Open Software Foundation, OSF, the OSF logo, OSF/I, OSF/Motif, and Motif are trademarks of Open Software Foundation, Inc.

UNIX is a registered trademark of UNIX Systems Laboratories, Inc. in the U.S. and other countries.

X/Open is a trademark of the X/Open Company Ltd. in the U.K. and other countries.

AT&T is a registered trademark of American Telephone & Telegraph Company in the U.S. and other countries.

BSD is a trademark of University of California, Berkeley.

DEC, DIGITAL, and VAX are registered trademarks of Digital Equipment Corporation

X Window System is a trademark of the Massachusetts Institute of Technology

MIPS is a trademark of Computer Systems, Inc.

Xerox is a registered trademark of Xerox Corporation

Sun Microsystems, Network File System, and NFS are trademarks of Sun Microsystems, Inc.

SMP, SMP+, and CMW+ are trademarks of SecureWare, Inc.

PostScript is a trademark of Adobe Systems, Inc.

Apple, the Apple Logo, Macintosh, AppleTalk, ImageWriter, and LaserWriter are registered trademarks of Apple Computer, Inc. A/UX is a trademark of Apple Computer.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>xiii</td>
</tr>
<tr>
<td>Audience</td>
<td>xiii</td>
</tr>
<tr>
<td>Applicability</td>
<td>xiv</td>
</tr>
<tr>
<td>Purpose</td>
<td>xiv</td>
</tr>
<tr>
<td>Document Usage</td>
<td>xiv</td>
</tr>
<tr>
<td>Related Documents</td>
<td>xiv</td>
</tr>
<tr>
<td>Typographic and Keying Conventions</td>
<td>xv</td>
</tr>
<tr>
<td>Problem Reporting</td>
<td>xvi</td>
</tr>
<tr>
<td>Permutted Index</td>
<td>xvii</td>
</tr>
<tr>
<td>Chapter 1 Functions</td>
<td></td>
</tr>
<tr>
<td>1.1 Organization of the Reference Pages</td>
<td>1-1</td>
</tr>
<tr>
<td>1.2 Error Numbers</td>
<td>1-2</td>
</tr>
<tr>
<td>abort</td>
<td>1-16</td>
</tr>
<tr>
<td>abs</td>
<td>1-17</td>
</tr>
<tr>
<td>accept</td>
<td>1-19</td>
</tr>
<tr>
<td>access</td>
<td>1-21</td>
</tr>
<tr>
<td>acct</td>
<td>1-23</td>
</tr>
<tr>
<td>adjtime</td>
<td>1-25</td>
</tr>
<tr>
<td>alarm</td>
<td>1-27</td>
</tr>
<tr>
<td>asinh</td>
<td>1-29</td>
</tr>
<tr>
<td>assert</td>
<td>1-30</td>
</tr>
<tr>
<td>async_daemon</td>
<td>1-32</td>
</tr>
<tr>
<td>atof</td>
<td>1-33</td>
</tr>
<tr>
<td>atoi</td>
<td>1-35</td>
</tr>
<tr>
<td>bcopy</td>
<td>1-39</td>
</tr>
<tr>
<td>bessel</td>
<td>1-41</td>
</tr>
<tr>
<td>bind</td>
<td>1-43</td>
</tr>
<tr>
<td>brk</td>
<td>1-45</td>
</tr>
<tr>
<td>bsearch</td>
<td>1-47</td>
</tr>
<tr>
<td>catclose</td>
<td>1-49</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
</tr>
<tr>
<td>catgets</td>
<td>1-51</td>
</tr>
<tr>
<td>catopen</td>
<td>1-53</td>
</tr>
<tr>
<td>cfgetispeed</td>
<td>1-55</td>
</tr>
<tr>
<td>cfgetospeed</td>
<td>1-56</td>
</tr>
<tr>
<td>cfsetispeed</td>
<td>1-57</td>
</tr>
<tr>
<td>cfsetospeed</td>
<td>1-58</td>
</tr>
<tr>
<td>chdir</td>
<td>1-59</td>
</tr>
<tr>
<td>chmod</td>
<td>1-61</td>
</tr>
<tr>
<td>chown</td>
<td>1-65</td>
</tr>
<tr>
<td>chroot</td>
<td>1-68</td>
</tr>
<tr>
<td>clearenv</td>
<td>1-70</td>
</tr>
<tr>
<td>clearerr</td>
<td>1-71</td>
</tr>
<tr>
<td>clock</td>
<td>1-72</td>
</tr>
<tr>
<td>close</td>
<td>1-73</td>
</tr>
<tr>
<td>connect</td>
<td>1-75</td>
</tr>
<tr>
<td>conv</td>
<td>1-78</td>
</tr>
<tr>
<td>ctermid</td>
<td>1-81</td>
</tr>
<tr>
<td>ctime</td>
<td>1-83</td>
</tr>
<tr>
<td>ctype</td>
<td>1-89</td>
</tr>
<tr>
<td>curses</td>
<td>1-92</td>
</tr>
<tr>
<td>cuserid</td>
<td>1-107</td>
</tr>
<tr>
<td>dbm</td>
<td>1-109</td>
</tr>
<tr>
<td>decode_mach_o_hdr</td>
<td>1-111</td>
</tr>
<tr>
<td>dn_comp</td>
<td>1-113</td>
</tr>
<tr>
<td>dn_expand</td>
<td>1-115</td>
</tr>
<tr>
<td>dn_find</td>
<td>1-117</td>
</tr>
<tr>
<td>dn_skipname</td>
<td>1-119</td>
</tr>
<tr>
<td>drand48</td>
<td>1-121</td>
</tr>
<tr>
<td>ecrvt</td>
<td>1-125</td>
</tr>
<tr>
<td>encode_mach_o_hdr</td>
<td>1-128</td>
</tr>
<tr>
<td>endhostent</td>
<td>1-130</td>
</tr>
<tr>
<td>endnetent</td>
<td>1-131</td>
</tr>
<tr>
<td>endprotoent</td>
<td>1-132</td>
</tr>
<tr>
<td>endservent</td>
<td>1-133</td>
</tr>
<tr>
<td>erf</td>
<td>1-134</td>
</tr>
<tr>
<td>exec</td>
<td>1-136</td>
</tr>
<tr>
<td>exec_with_loader</td>
<td>1-142</td>
</tr>
<tr>
<td>exit</td>
<td>1-145</td>
</tr>
<tr>
<td>exp</td>
<td>1-148</td>
</tr>
<tr>
<td>expacct</td>
<td>1-151</td>
</tr>
<tr>
<td>fclose</td>
<td>1-152</td>
</tr>
<tr>
<td>fcntl</td>
<td>1-155</td>
</tr>
<tr>
<td>feof</td>
<td>1-161</td>
</tr>
<tr>
<td>ferror</td>
<td>1-162</td>
</tr>
<tr>
<td>fileno</td>
<td>1-163</td>
</tr>
<tr>
<td>flock</td>
<td>1-164</td>
</tr>
<tr>
<td>flockfile</td>
<td>1-167</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
</tr>
<tr>
<td>floor</td>
<td>1-168</td>
</tr>
<tr>
<td>fopen</td>
<td>1-171</td>
</tr>
<tr>
<td>fork</td>
<td>1-176</td>
</tr>
<tr>
<td>fread</td>
<td>1-179</td>
</tr>
<tr>
<td>frexp</td>
<td>1-181</td>
</tr>
<tr>
<td>fseek</td>
<td>1-184</td>
</tr>
<tr>
<td>fsync</td>
<td>1-188</td>
</tr>
<tr>
<td>ftok</td>
<td>1-190</td>
</tr>
<tr>
<td>ftw</td>
<td>1-192</td>
</tr>
<tr>
<td>funlockfile</td>
<td>1-195</td>
</tr>
<tr>
<td>gamma</td>
<td>1-196</td>
</tr>
<tr>
<td>getaddressconf</td>
<td>1-198</td>
</tr>
<tr>
<td>getc</td>
<td>1-201</td>
</tr>
<tr>
<td>getclock</td>
<td>1-203</td>
</tr>
<tr>
<td>getcwd</td>
<td>1-205</td>
</tr>
<tr>
<td>getdirentries</td>
<td>1-207</td>
</tr>
<tr>
<td>getdiskbyname</td>
<td>1-209</td>
</tr>
<tr>
<td>getdtablesizel</td>
<td>1-210</td>
</tr>
<tr>
<td>getenv</td>
<td>1-211</td>
</tr>
<tr>
<td>gets</td>
<td>1-212</td>
</tr>
<tr>
<td>getsent</td>
<td>1-214</td>
</tr>
<tr>
<td>getsstat</td>
<td>1-216</td>
</tr>
<tr>
<td>getgid</td>
<td>1-218</td>
</tr>
<tr>
<td>getgrent</td>
<td>1-219</td>
</tr>
<tr>
<td>getgroups</td>
<td>1-222</td>
</tr>
<tr>
<td>gethostbyaddr</td>
<td>1-224</td>
</tr>
<tr>
<td>gethostbyname</td>
<td>1-226</td>
</tr>
<tr>
<td>gethostent</td>
<td>1-228</td>
</tr>
<tr>
<td>gethostid</td>
<td>1-230</td>
</tr>
<tr>
<td>gethostname</td>
<td>1-231</td>
</tr>
<tr>
<td>getitimer</td>
<td>1-232</td>
</tr>
<tr>
<td>getlogin</td>
<td>1-235</td>
</tr>
<tr>
<td>_getlong</td>
<td>1-237</td>
</tr>
<tr>
<td>getnetbyaddr</td>
<td>1-239</td>
</tr>
<tr>
<td>getnetbyname</td>
<td>1-241</td>
</tr>
<tr>
<td>getnetent</td>
<td>1-243</td>
</tr>
<tr>
<td>getopt</td>
<td>1-244</td>
</tr>
<tr>
<td>getpagesize</td>
<td>1-246</td>
</tr>
<tr>
<td>getpass</td>
<td>1-247</td>
</tr>
<tr>
<td>getpeernamet</td>
<td>1-249</td>
</tr>
<tr>
<td>getpid</td>
<td>1-251</td>
</tr>
<tr>
<td>getpriority</td>
<td>1-252</td>
</tr>
<tr>
<td>getprotobynamet</td>
<td>1-254</td>
</tr>
<tr>
<td>getprotobynumber</td>
<td>1-256</td>
</tr>
<tr>
<td>getprotoent</td>
<td>1-258</td>
</tr>
<tr>
<td>getpwent</td>
<td>1-259</td>
</tr>
<tr>
<td>getrlimit</td>
<td>1-262</td>
</tr>
</tbody>
</table>
getrusage
getservbyname
getservbyport
getservent
_getshort
getsockname
getsockopt
gettimeofday
gett imeofday
gettimeofday
getuid
getusershell
getutent
getwc
getwd
getws
hsearch
htoi
htons
hypot
inet_addr
inet_lnaof
inet_makeaddr
inet_netof
inet_network
inet_ntoa
initgroups
insque
ioctl
isnan
jctype
kill
ldr_entry
ldr_inq_module
ldr_inq_region
ldr_install
ldr_lookup_package
ldr_next_module
ldr_remove
ldr_xattach
ldr_xdetach
ldr_xentry
ldr_xload
ldr_xlookup_package
ldr_xunload
libPW
link
listen .................................................. 1-347
load ................................................... 1-349
localeconv ............................................. 1-351
lockf .................................................. 1-355
lsearch ............................................... 1-358
lseek ................................................... 1-360
madvise ................................................. 1-362
malloc .................................................. 1-364
mblen ................................................... 1-368
mbstowcs ............................................... 1-370
mbtowc .................................................. 1-372
memccpy ............................................... 1-374
mkdir .................................................. 1-378
mkfifo ................................................... 1-381
mknod ................................................... 1-383
mktemp ................................................. 1-386
mktimer ............................................... 1-388
mmap ..................................................... 1-390
mount .................................................... 1-395
mount .................................................... 1-400
mp ........................................................ 1-402
mprotect ................................................ 1-406
msem_init ............................................. 1-409
msem_lock ............................................. 1-411
msem_remove .......................................... 1-413
msem_unlock ......................................... 1-415
msgctl ................................................ 1-417
msgget ............................................... 1-420
msgrecv .............................................. 1-422
msgsnd ............................................... 1-425
msync ................................................ 1-428
munmap ............................................... 1-430
mvalid ............................................... 1-432
ndbm .................................................. 1-434
neg ..................................................... 1-437
nfssvc ............................................... 1-438
nice ................................................... 1-439
nl_langinfo .......................................... 1-441
ns_addr ................................................. 1-443
ntohl .................................................. 1-445
n-tohs ................................................. 1-446
open .................................................... 1-447
opendir ............................................... 1-453
pathconf ............................................... 1-458
pause .................................................. 1-462
pclose .................................................. 1-464
perror .................................................. 1-466
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipe</td>
<td>1-467</td>
</tr>
<tr>
<td>plock</td>
<td>1-469</td>
</tr>
<tr>
<td>poll</td>
<td>1-471</td>
</tr>
<tr>
<td>popen</td>
<td>1-474</td>
</tr>
<tr>
<td>printf</td>
<td>1-476</td>
</tr>
<tr>
<td>profil</td>
<td>1-483</td>
</tr>
<tr>
<td>pthread_attr_create</td>
<td>1-485</td>
</tr>
<tr>
<td>pthread_attr_delete</td>
<td>1-487</td>
</tr>
<tr>
<td>pthread_attr_getstacksize</td>
<td>1-488</td>
</tr>
<tr>
<td>pthread_attr_setstacksize</td>
<td>1-490</td>
</tr>
<tr>
<td>pthread_cancel</td>
<td>1-492</td>
</tr>
<tr>
<td>pthread_cleanup_pop</td>
<td>1-494</td>
</tr>
<tr>
<td>pthread_cleanup_push</td>
<td>1-496</td>
</tr>
<tr>
<td>pthread_cond_broadcast</td>
<td>1-498</td>
</tr>
<tr>
<td>pthread_cond_destroy</td>
<td>1-500</td>
</tr>
<tr>
<td>pthread_cond_init</td>
<td>1-502</td>
</tr>
<tr>
<td>pthread_cond_signal</td>
<td>1-504</td>
</tr>
<tr>
<td>pthread_cond_timedwait</td>
<td>1-506</td>
</tr>
<tr>
<td>pthread_cond_wait</td>
<td>1-508</td>
</tr>
<tr>
<td>pthread_condattr_create</td>
<td>1-510</td>
</tr>
<tr>
<td>pthread_condattr_delete</td>
<td>1-512</td>
</tr>
<tr>
<td>pthread_create</td>
<td>1-514</td>
</tr>
<tr>
<td>pthread_detach</td>
<td>1-516</td>
</tr>
<tr>
<td>pthread_equal</td>
<td>1-518</td>
</tr>
<tr>
<td>pthread_exit</td>
<td>1-519</td>
</tr>
<tr>
<td>pthread_getspecific</td>
<td>1-520</td>
</tr>
<tr>
<td>pthread_join</td>
<td>1-522</td>
</tr>
<tr>
<td>pthread_keycreate</td>
<td>1-524</td>
</tr>
<tr>
<td>pthread_mutex_destroy</td>
<td>1-526</td>
</tr>
<tr>
<td>pthread_mutex_init</td>
<td>1-528</td>
</tr>
<tr>
<td>pthread_mutex_lock</td>
<td>1-530</td>
</tr>
<tr>
<td>pthread_mutex_trylock</td>
<td>1-532</td>
</tr>
<tr>
<td>pthread_mutex_unlock</td>
<td>1-534</td>
</tr>
<tr>
<td>pthread_mutexattr_create</td>
<td>1-536</td>
</tr>
<tr>
<td>pthread_mutexattr_delete</td>
<td>1-538</td>
</tr>
<tr>
<td>pthread_once</td>
<td>1-539</td>
</tr>
<tr>
<td>pthread_self</td>
<td>1-541</td>
</tr>
<tr>
<td>pthread_setasynccancel</td>
<td>1-542</td>
</tr>
<tr>
<td>pthread_setcancel</td>
<td>1-545</td>
</tr>
<tr>
<td>pthread_setspecific</td>
<td>1-547</td>
</tr>
<tr>
<td>pthread_testcancel</td>
<td>1-549</td>
</tr>
<tr>
<td>pthread_yield</td>
<td>1-550</td>
</tr>
<tr>
<td>ptrace</td>
<td>1-551</td>
</tr>
<tr>
<td>putc</td>
<td>1-555</td>
</tr>
<tr>
<td>putenv</td>
<td>1-558</td>
</tr>
<tr>
<td>putlong</td>
<td>1-559</td>
</tr>
<tr>
<td>puts</td>
<td>1-560</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------</td>
<td>-------</td>
</tr>
<tr>
<td>putshort</td>
<td>1-562</td>
</tr>
<tr>
<td>putwc</td>
<td>1-564</td>
</tr>
<tr>
<td>putws</td>
<td>1-566</td>
</tr>
<tr>
<td>qsort</td>
<td>1-568</td>
</tr>
<tr>
<td>quotactl</td>
<td>1-570</td>
</tr>
<tr>
<td>raise</td>
<td>1-573</td>
</tr>
<tr>
<td>rand</td>
<td>1-574</td>
</tr>
<tr>
<td>random</td>
<td>1-577</td>
</tr>
<tr>
<td>rcmd</td>
<td>1-580</td>
</tr>
<tr>
<td>recomp</td>
<td>1-582</td>
</tr>
<tr>
<td>read</td>
<td>1-584</td>
</tr>
<tr>
<td>readlink</td>
<td>1-588</td>
</tr>
<tr>
<td>reboot</td>
<td>1-590</td>
</tr>
<tr>
<td>recv</td>
<td>1-593</td>
</tr>
<tr>
<td>recvfrom</td>
<td>1-595</td>
</tr>
<tr>
<td>recvmsg</td>
<td>1-598</td>
</tr>
<tr>
<td>regexp</td>
<td>1-601</td>
</tr>
<tr>
<td>reltimer</td>
<td>1-606</td>
</tr>
<tr>
<td>remove</td>
<td>1-608</td>
</tr>
<tr>
<td>rename</td>
<td>1-610</td>
</tr>
<tr>
<td>res_init</td>
<td>1-613</td>
</tr>
<tr>
<td>res_mkquery</td>
<td>1-615</td>
</tr>
<tr>
<td>res_send</td>
<td>1-618</td>
</tr>
<tr>
<td>rexec</td>
<td>1-620</td>
</tr>
<tr>
<td>rmdir</td>
<td>1-623</td>
</tr>
<tr>
<td>rmtimer</td>
<td>1-625</td>
</tr>
<tr>
<td>rresvport</td>
<td>1-626</td>
</tr>
<tr>
<td>ruserok</td>
<td>1-628</td>
</tr>
<tr>
<td>scandir</td>
<td>1-630</td>
</tr>
<tr>
<td>scanf</td>
<td>1-632</td>
</tr>
<tr>
<td>select</td>
<td>1-638</td>
</tr>
<tr>
<td>semctl</td>
<td>1-642</td>
</tr>
<tr>
<td>semget</td>
<td>1-646</td>
</tr>
<tr>
<td>semop</td>
<td>1-649</td>
</tr>
<tr>
<td>send</td>
<td>1-653</td>
</tr>
<tr>
<td>sendmsg</td>
<td>1-655</td>
</tr>
<tr>
<td>sendto</td>
<td>1-657</td>
</tr>
<tr>
<td>setbuf</td>
<td>1-660</td>
</tr>
<tr>
<td>setclock</td>
<td>1-662</td>
</tr>
<tr>
<td>setgid</td>
<td>1-664</td>
</tr>
<tr>
<td>setgroups</td>
<td>1-666</td>
</tr>
<tr>
<td>sethostid</td>
<td>1-668</td>
</tr>
<tr>
<td>sethostname</td>
<td>1-669</td>
</tr>
<tr>
<td>setjmp</td>
<td>1-670</td>
</tr>
<tr>
<td>setlocale</td>
<td>1-672</td>
</tr>
<tr>
<td>setnetent</td>
<td>1-676</td>
</tr>
<tr>
<td>setpgid</td>
<td>1-677</td>
</tr>
</tbody>
</table>
setprotoent .......................... 1-679
setquota ................................ 1-680
setregid ................................ 1-682
setreuid ................................ 1-683
setrgid ................................ 1-684
setruid ................................ 1-686
setsid ................................ 1-688
setsockopt ............................. 1-689
setuid ................................ 1-690
shmat ................................ 1-694
shmct1 ................................ 1-696
shmdt ................................ 1-699
shmget ................................ 1-701
shutdown ................................ 1-702
sigaction ................................ 1-705
sigblock ................................ 1-706
sigemptyset ............................ 1-710
siginterrupt ........................... 1-711
siglongjmp ................................ 1-714
sigpause ................................ 1-716
sigpending ............................ 1-718
sigprocmask ............................ 1-720
sigreturn ................................ 1-721
sigset ................................ 1-724
sigsetjmp ................................ 1-726
sigstack ................................ 1-729
sigsuspend ............................ 1-730
sigvec ................................ 1-732
sigwait ................................ 1-734
sin ...................................... 1-737
sinh ...................................... 1-739
sleep ................................ 1-742
socket ................................ 1-744
socketpair ................................ 1-745
sqrt ...................................... 1-748
stat ................................ 1-750
statfs ................................ 1-752
stime ................................ 1-754
strptime ................................ 1-756
string ...................................... 1-757
swab ...................................... 1-760
swapon ................................ 1-767
symlink ................................ 1-768
sync ................................ 1-770
sysconf ................................ 1-773
syslog ................................ 1-774
syslog ...................................... 1-776
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>1-780</td>
</tr>
<tr>
<td>t_accept</td>
<td>1-782</td>
</tr>
<tr>
<td>t_alloc</td>
<td>1-786</td>
</tr>
<tr>
<td>t_bind</td>
<td>1-790</td>
</tr>
<tr>
<td>t_close</td>
<td>1-795</td>
</tr>
<tr>
<td>t_connect</td>
<td>1-797</td>
</tr>
<tr>
<td>t_error</td>
<td>1-803</td>
</tr>
<tr>
<td>t_free</td>
<td>1-805</td>
</tr>
<tr>
<td>t_getinfo</td>
<td>1-808</td>
</tr>
<tr>
<td>t_getstate</td>
<td>1-812</td>
</tr>
<tr>
<td>t_listen</td>
<td>1-814</td>
</tr>
<tr>
<td>t_look</td>
<td>1-818</td>
</tr>
<tr>
<td>t_open</td>
<td>1-822</td>
</tr>
<tr>
<td>t_optmgmt</td>
<td>1-827</td>
</tr>
<tr>
<td>t_rcv</td>
<td>1-831</td>
</tr>
<tr>
<td>t_rcvconnect</td>
<td>1-834</td>
</tr>
<tr>
<td>t_rcvdis</td>
<td>1-838</td>
</tr>
<tr>
<td>t_rcvrel</td>
<td>1-842</td>
</tr>
<tr>
<td>t_rcvudata</td>
<td>1-844</td>
</tr>
<tr>
<td>t_rcvuderr</td>
<td>1-848</td>
</tr>
<tr>
<td>t_snd</td>
<td>1-851</td>
</tr>
<tr>
<td>t_snddis</td>
<td>1-855</td>
</tr>
<tr>
<td>t_sndrel</td>
<td>1-858</td>
</tr>
<tr>
<td>t_sndudata</td>
<td>1-860</td>
</tr>
<tr>
<td>t_sync</td>
<td>1-863</td>
</tr>
<tr>
<td>t_unbind</td>
<td>1-866</td>
</tr>
<tr>
<td>tcdrain</td>
<td>1-868</td>
</tr>
<tr>
<td>tcfget</td>
<td>1-869</td>
</tr>
<tr>
<td>tcflow</td>
<td>1-870</td>
</tr>
<tr>
<td>tcflush</td>
<td>1-872</td>
</tr>
<tr>
<td>tcgetattr</td>
<td>1-874</td>
</tr>
<tr>
<td>tcgetpgrp</td>
<td>1-876</td>
</tr>
<tr>
<td>tcsendbreak</td>
<td>1-878</td>
</tr>
<tr>
<td>tcssetattr</td>
<td>1-880</td>
</tr>
<tr>
<td>tcssetpgrp</td>
<td>1-882</td>
</tr>
<tr>
<td>time</td>
<td>1-884</td>
</tr>
<tr>
<td>times</td>
<td>1-885</td>
</tr>
<tr>
<td>tmpfile</td>
<td>1-887</td>
</tr>
<tr>
<td>tmpnam</td>
<td>1-888</td>
</tr>
<tr>
<td>truncate</td>
<td>1-890</td>
</tr>
<tr>
<td>tsearch</td>
<td>1-893</td>
</tr>
<tr>
<td>ttyname</td>
<td>1-896</td>
</tr>
<tr>
<td>ttyslot</td>
<td>1-898</td>
</tr>
<tr>
<td>ulimit</td>
<td>1-899</td>
</tr>
<tr>
<td>umask</td>
<td>1-901</td>
</tr>
<tr>
<td>umount</td>
<td>1-902</td>
</tr>
<tr>
<td>uname</td>
<td>1-904</td>
</tr>
<tr>
<td>ungetc</td>
<td>1-906</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------</td>
<td>--------</td>
</tr>
<tr>
<td>unlink</td>
<td>1-908</td>
</tr>
<tr>
<td>unload</td>
<td>1-910</td>
</tr>
<tr>
<td>unlocked_getc</td>
<td>1-912</td>
</tr>
<tr>
<td>unlocked_putchar</td>
<td>1-913</td>
</tr>
<tr>
<td>usleep</td>
<td>1-914</td>
</tr>
<tr>
<td>utime</td>
<td>1-915</td>
</tr>
<tr>
<td>varargs</td>
<td>1-918</td>
</tr>
<tr>
<td>vprintf</td>
<td>1-921</td>
</tr>
<tr>
<td>wait</td>
<td>1-923</td>
</tr>
<tr>
<td>wcstombs</td>
<td>1-928</td>
</tr>
<tr>
<td>wchar</td>
<td>1-930</td>
</tr>
<tr>
<td>write</td>
<td>1-932</td>
</tr>
<tr>
<td>wsprintf</td>
<td>1-937</td>
</tr>
<tr>
<td>wscanf</td>
<td>1-939</td>
</tr>
<tr>
<td>wstring</td>
<td>1-941</td>
</tr>
</tbody>
</table>

Chapter 2 Files

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar</td>
<td>2-1</td>
</tr>
<tr>
<td>core</td>
<td>2-2</td>
</tr>
<tr>
<td>ctab</td>
<td>2-3</td>
</tr>
<tr>
<td>dir</td>
<td>2-4</td>
</tr>
<tr>
<td>disklabel</td>
<td>2-9</td>
</tr>
<tr>
<td>disktab</td>
<td>2-10</td>
</tr>
<tr>
<td>en</td>
<td>2-12</td>
</tr>
<tr>
<td>exports</td>
<td>2-15</td>
</tr>
<tr>
<td>fd</td>
<td>2-18</td>
</tr>
<tr>
<td>fs</td>
<td>2-20</td>
</tr>
<tr>
<td>group</td>
<td>2-21</td>
</tr>
<tr>
<td>icmp</td>
<td>2-24</td>
</tr>
<tr>
<td>idp</td>
<td>2-25</td>
</tr>
<tr>
<td>inet</td>
<td>2-27</td>
</tr>
<tr>
<td>ip</td>
<td>2-30</td>
</tr>
<tr>
<td>lo</td>
<td>2-32</td>
</tr>
<tr>
<td>lvm</td>
<td>2-34</td>
</tr>
<tr>
<td>msqid_ds</td>
<td>2-35</td>
</tr>
<tr>
<td>netintro</td>
<td>2-56</td>
</tr>
<tr>
<td>ns</td>
<td>2-58</td>
</tr>
<tr>
<td>nsip</td>
<td>2-64</td>
</tr>
<tr>
<td>null</td>
<td>2-66</td>
</tr>
<tr>
<td>OSF/ROSE</td>
<td>2-67</td>
</tr>
<tr>
<td>passwd</td>
<td>2-68</td>
</tr>
<tr>
<td>protocols</td>
<td>2-96</td>
</tr>
<tr>
<td>pty</td>
<td>2-98</td>
</tr>
<tr>
<td>resolver</td>
<td>2-99</td>
</tr>
<tr>
<td>route</td>
<td>2-102</td>
</tr>
<tr>
<td>semid_ds</td>
<td>2-104</td>
</tr>
<tr>
<td>services</td>
<td>2-106</td>
</tr>
<tr>
<td>shells</td>
<td>2-109</td>
</tr>
<tr>
<td></td>
<td>2-110</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>shmid_ds</td>
<td>2-111</td>
</tr>
<tr>
<td>signal</td>
<td>2-113</td>
</tr>
<tr>
<td>spp</td>
<td>2-118</td>
</tr>
<tr>
<td>stab</td>
<td>2-120</td>
</tr>
<tr>
<td>tar</td>
<td>2-123</td>
</tr>
<tr>
<td>tcp</td>
<td>2-125</td>
</tr>
<tr>
<td>terminfo</td>
<td>2-127</td>
</tr>
<tr>
<td>termios</td>
<td>2-139</td>
</tr>
<tr>
<td>tty</td>
<td>2-151</td>
</tr>
<tr>
<td>udp</td>
<td>2-165</td>
</tr>
<tr>
<td>ascii</td>
<td>3-1</td>
</tr>
<tr>
<td>end</td>
<td>3-2</td>
</tr>
<tr>
<td>environ</td>
<td>3-4</td>
</tr>
<tr>
<td>hier</td>
<td>3-5</td>
</tr>
<tr>
<td>hostname</td>
<td>3-7</td>
</tr>
</tbody>
</table>

Chapter 3 Miscellaneous Functions
List of Tables

Table 1-1. OSF/1 Errnos .......................... 1-3
The *OSF/1 Programmer's Reference* contains reference pages for OSF/1™ system calls, library routines, file formats, and special files.

**Audience**

This book is for application programmers who want to use the application programming interface provided with the OSF/1 operating system. The book assumes that the reader is a programmer familiar with the C programming language.
Applicability

This book applies to Release 1.0 of the OSF/1 operating system.

Purpose

The purpose of this book is to provide a complete reference to all features of the operating system’s application programming interface.

Document Usage

This document is organized into three chapters.

• *Chapter 1* is a reference to functions in OSF/1, both system and library calls. It contains reference pages from both the man2 and man3 directories, sorted alphabetically.

• *Chapter 2* is a reference to files in OSF/1. It contains reference pages from both the man4 and man7 directories, sorted alphabetically.

• *Chapter 3* is a reference to miscellaneous facilities, found in the man5 directory.

Related Documents

The following documents are also included with the OSF/1 documentation set:

• *OSF/1 Applications Programmer’s Guide*
• *OSF/1 Security Features Programmer’s Guide*
• *OSF/1 System Programmer’s Reference Volume 1*
• *OSF/1 System Programmer’s Reference Volume 2*
• *OSF/1 Command Reference*
• *OSF/1 System and Network Administrator’s Reference*

• *OSF/1 Network Applications Programmer’s Guide*

• *Application Environment Specification - Operating System/Programming Interfaces Volume*

## Typographic and Keying Conventions

This document uses the following typographic conventions:

**Bold**

*Bold* words or characters represent system elements that you must use literally, such as commands, flags, and pathnames.

**Italic**

*Italic* words or characters represent variable values that you must supply.

**Constant width**

Examples and information that the system displays appears in the constant width typeface.

**[]**

Brackets enclose optional items in format and syntax descriptions.

**l**

A vertical bar separates items in a list of choices.

**...**

A horizontal ellipsis indicates that you can repeat the preceding item one or more times. A vertical ellipsis indicates that you can repeat the preceding line one or more times.
Problem Reporting

If you have any problems with the software or documentation, please contact your software vendor's customer service department.
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/initstate, setstate</td>
<td>Generates &quot;better&quot; pseudo-random numbers</td>
</tr>
<tr>
<td>/Converts an unsigned short</td>
<td>(16-bit) integer from host-byte</td>
</tr>
<tr>
<td>/Converts an unsigned long</td>
<td>(16-bit) integer from Internet/</td>
</tr>
<tr>
<td>htonl:</td>
<td>Converts an unsigned long</td>
</tr>
<tr>
<td>ntohl:</td>
<td>Converts an unsigned long</td>
</tr>
<tr>
<td>semaphore ID semget:</td>
<td>Returns (and possibly creates) a</td>
</tr>
<tr>
<td>msgget:</td>
<td>Converts a message queue</td>
</tr>
<tr>
<td>shmget:</td>
<td>Returns a shared memory/</td>
</tr>
<tr>
<td>htonl(3)</td>
<td>(and possibly creates) the ID for msgget(2)</td>
</tr>
<tr>
<td>ntohs(3)</td>
<td>(and possibly creates) the ID for shmget(2)</td>
</tr>
<tr>
<td>htonl(3)</td>
<td>(library) file format</td>
</tr>
<tr>
<td>htons(3)</td>
<td>(local) address component</td>
</tr>
<tr>
<td>lvm(7)</td>
<td>Logical Volume Manager (LVM) programming interface</td>
</tr>
<tr>
<td>isnan(3)</td>
<td>Tests for NaN (Not a Number)</td>
</tr>
<tr>
<td>mbstowcs(3)</td>
<td>Converts a multibyte (single-byte or double-byte)/</td>
</tr>
<tr>
<td>mbstowcs(3)</td>
<td>Xerox sequenced packet protocol</td>
</tr>
<tr>
<td>udp(7)</td>
<td>Internet user datagram protocol</td>
</tr>
<tr>
<td>udp(7)</td>
<td>/etc/protocols file</td>
</tr>
<tr>
<td>getprotoent(3)</td>
<td>/etc/protocols file getprotoent:</td>
</tr>
<tr>
<td>/etc/services file entry</td>
<td>/etc/services file entry</td>
</tr>
<tr>
<td>2-byte Internet network integer</td>
<td>2-byte Internet network integer</td>
</tr>
<tr>
<td>htns(3)</td>
<td>2-byte Internet network integer</td>
</tr>
<tr>
<td>ldr_xattach(3)</td>
<td>/etc/protocols file getprotoent:</td>
</tr>
<tr>
<td>ldr_xattach(3)</td>
<td>/etc/services file entry</td>
</tr>
<tr>
<td>ldr_xattach(3)</td>
<td>2-byte Internet network integer</td>
</tr>
<tr>
<td>getdvagent(3)</td>
<td>Xerox sequenced packet protocol</td>
</tr>
<tr>
<td>getdvagent(3)</td>
<td>Internet user datagram protocol</td>
</tr>
<tr>
<td>getdvagent(3)</td>
<td>/etc/protocols file getprotoent:</td>
</tr>
<tr>
<td>getdvagent(3)</td>
<td>/etc/services file entry</td>
</tr>
<tr>
<td>getdvagent(3)</td>
<td>/etc/services file entry</td>
</tr>
<tr>
<td>getdvagent(3)</td>
<td>2-byte Internet network integer</td>
</tr>
<tr>
<td>getdvagent(3)</td>
<td>2-byte Internet network integer</td>
</tr>
<tr>
<td>getdvagent(3)</td>
<td>2-byte Internet network integer</td>
</tr>
</tbody>
</table>
exit, atexit, _exit: Terminates a process ................................... exit(2)
_getlong: Retrieves long ....................................... _getlong(3)
_getshort: Retrieves short ...................................... _getshort(3)
_translates/ toascii, tolower, _tolower, toupper, _toupper: ................................. conv(3)
_abort: Generates a software abort(3)
_abs, div, labs, ldiv: Computes abs(3)
_absolute value and division of/ abs(3)
_Remainder and floating-point distance function and complex on a socket
accept: Accepts a new connection accept(2)
acceptable_password: Determines acceptable_password(3)
socket accept: Accepts a new connection on a accept(2)
_UTIME, UTIMES: Sets file access and modification times utime(2)
_record basis audit: Open and access audit session data on a audit(3)
_macilb: Mandatory access control and information macilb(4)
_Mandatory: Mandatory access control databases mandatory(4)
_functions acl: Access control list conversion acl(3)
_policy databases acl: Access control list discretionary acl(4)
_chacl: Changes the access control list of a file chacl(3)
_satacl: Retrieves the access control list of a file statacl(3)
_unix_chacl: Manipulates access control lists on/ ipc_acl(3)
_setgroups: Sets the group access list setgroups(2)
_chmod, fchmod: Changes file access permissions chmod(2)
_mapping mprotect: Modifies access protections of memory mprotect(2)
_sectlink, sectlabel: Format of the additional security attributes section_label(3)
_security independent disk inode access routines disk: disk(3)
eaccess: Determines effective access to a file eaccess(3)
access: Determines the access(2)
_getpwent, setpwent, endpwent: Accesses the basic user getpwent(3)
_getgrnam, setgrnt, endgrnt: Accesses the basic group/ getgrnt(3)
_setutent, endutent, utmpname: Accesses utmp file entries getutent(3)
_access: Determines the accessibility of a file access(2)
/getgrnm, setgrnt, endgrnt: Accesses the basic group/ getgrnt(3)
_getpwent, setpwent, endpwent: Accesses the basic user getpwent(3)
_setutent, endutent, utmpname: Accesses utmp file entries getutent(3)
_access: Determines the accessibility of a file access(2)
/Internet/ /Translates an Internet address integer into its host (local)/ inet_makeaddr(3)
_integer into its network ns_addr, ns_ntoa: Xerox NS inet_lnaof(3)
/address into its network inet_makeaddr(3)
/address into its network inet_lnaof(3)
/address conversion routines ns_addr(3)
/address conversion routines ns_addr(3)
/address into its host inet_lnaof(3)
/address into its network inet_lnaof(3)
async_daemon: Creates a local NFS async_daemon(2)

cmdauth: Format of Command Authorizations Definition file cmdauth(7)

/setpwent, endpwent: Accesses the basic user information in the getpwent(3)

/socket connections and limits the backlog of incoming connections listen(2)

/gets the alphanumeric username associated with the current/ cuserid(3)

closes a condition variable attributes object pthread_condattr_create(3)

/Enables or disables the asynchronous cancelability of the permission pthread_setasynccancel(3)

/Converts a character string to a character string to/ atoi, atol, strtol, strtoul: atoi(3)

/tcgetattr: Gets the parameters associated with the terminal tcgetattr(3)

/Creates a mutex attributes object pthread_mutexattr_create(3)

/close: Closes the file close(2)

/setpriv: Sets kernel authorizations and privileges setpriv(3)

/getpiv: Gets privilege or authorization sets associated with a file descriptor getpiv(3)

/setgrent, endgrent: Accesses the basic group information in the getgrent(3)

/getpriv: Gets privilege or authorization sets associated with this process getpriv(3)

/setpwent, endpwent: Accesses the user information in the getpwent(3)

/cfgetospeed: Gets output baud rate for a terminal cfgetospeed(3)

/setuid, setgid, getpiv: Gets privilege or authorization sets associated with this process getpiv(3)

/setgrent, endgrent: Accesses the basic group information in the getgrent(3)

/cfsetospeed: Sets output baud rate for a terminal cfsetospeed(3)

/setgrent, endgrent: Accesses the basic group information in the getgrent(3)

/setpwent, endpwent: Accesses the user information in the getpwent(3)
and byte string/bcopy, bcmp, bzero, ffs: Performs bit operations

bcopy, bcmp, bzero, ffs: Performs bit operations

of a process' expected paging

j0, j1, jn, y0, y1, yn: Computes Bessel functions

ffs: Performs bit

bind: Binds a name to a socket

bind: Binds a thread-specific value to

Atomically changes the set of blocked signals and waits for a

Bound to a key

privileges: Perform privilege bracketing

bzero, ffs: Performs

data line tcsendbreak: Sends a break on an asynchronous serial

size

setbuffer, setlinebuf: Assigns buffering to a stream

stream putlong: Places long byte quantities into the byte

stream putshort: Places short byte quantities into the byte

Retrieves long quantities from a byte stream

Retrieves short quantities from a byte stream

bzero, ffs: Performs bit and

/swab: Swaps

string operations bcopy, bcmp, bzero, ffs: Performs bit and byte

function and complex/hypot, hypot

top of the cleanup stack of the pthread_exit: Terminates the thread

a cancellation point in the asynchronous cancelability of the thread

to the cleanup stack of the thread

Returns the ID of the thread

alloca/malloc, free, realloc, pthread_once:

/or enables the asynchronous

/Enables or disables the general

pthread_testcancel: Creates a cancellation point in the calling thread

cancellation point in the calling thread

canonical form/file header

decode_mach_o_hdr: Converts the canonical header from an OSF/ROSE format

terminfo: Describes terminals by locale character classification,

Closes a specified message

Retrieves a message from a specified message catalog

catclose: Closes a specified message catalog
OSF/1 Programmer's Reference

- a catalog
  message catalog
  cube root functions sqrt, cbrt
  floating-point numbers to/ for a terminal:
    floor, rint, fmod: Rounds
    for a terminal:
    rate for a terminal:
  list of a file
    brk, sbrk:
    chmod, fchmod:
    truncate, fruncate:
    storage fsync:

- entire current locale/ setlocale:
  process nice:
  of a file chac1:
  chdir, fchdir:
  directory chroot:
  a file chilabel:
  signals/ sigsuspend:

- timers alarm, ualarm:
  pipe:
  ungetc, ungetwc:

- conversion, and/ ctab:

- hexadecimial, and decimal ASCII
  wcstombs:
  wctombs:
  mbtowc:
  mblen:
  a wide character to a wide
  a multi-byte character to a wide
  a wide character into a multibyte

- Retrieves file implementation
  isjspace, isjpunct: Classifies
  iscntrl, isascii: Classifies
  toupper, _toupper: Translates
  current directory
  tod:

- catgets: Retrieves a message from
  catopen: Opens a specified
  sqrt, cbrt: Computes square root and
  floor(3)
  for a terminal:
  rate for a terminal:
  for a terminal:
  rate for a terminal:
  changes in a file to permanent
  changes or queries the program's
  nice:
  changes the access control list
  changes the current directory
  changes the effective root
  changes the information label of
  changes the owner and group IDs
  changes the sensitivity label of
  changes the set of blocked
  changes the timeout of interval
  creates an interprocess
  character back into input stream
  character classification, case
  character from an input stream
  character into a multibyte
  character into a wide
  character or a word to a stream
  character or a word to a stream
  character or word from an input/
  character or word from an input/
  character sets ascii: Octal,
  character string into a/
  character string to a wide/
  character string to a/:
  character string to the specified/
  character string to the specified/
  character string to an integer

- isctype:
  isjxdigit, isjalnum, ...
  isprint, isgraph,
  _toupper, _tolower:
  Replaces:
  Checks:
  Changes:
  Changes:
  Changes:
  Checks:
  Checks:
  Checks:
  Checks:
  Checks:
mvalid: Checks memory region for validity .......................... mvalid(2)
multilevel ismultdir: Checks to see if a directory is .................. ismultdir(3)
identity: Gets or checks user or group IDs ................................ identity(3)
label of a file chilabel: Changes the information .................. chilabel(3)
times: Gets process and child process times .......................... times(3)
wait, waitpid, wait3: Waits for a child process to stop or/ ............ wait(2)
ptrace: Traces the execution of a child process ........................ ptrace(2)
access permissions chmod, fchmod: Changes file permissions ........... chmod(2)
and group IDs of a file chown, fchown: Changes the owner ........... chown(2)
chpriv: Sets file privileges .............................................. chpriv(3)
root directory chroot: Changes the effective root directory .......... chroot(2)
label of a file chslabel: Changes the sensitivity of a file ........... chslabel(3)
and/ etab: Locale character classification, case conversion, .......... etab(4)
isjalnum, isjspace, isjpunct: Classifies characters ..................... jctype(3)
isgraph, iscntrl, isascii: Classifies characters ......................... ctype(3)
thread /pushes a routine onto the cleanup stack of the calling .......... pthread_cleanup_push(3)
/a routine from the top of the cleanup stack of the calling/ .......... pthread_cleanup_pop(3)
clearance: Clearance functions .......................................... clearance(3)
environment clearenv: Clears the process environment ............... clearenv(3)
stream clearerr: Clears indicators on a stream ........................ clearerr(3)
clearerr: Clears indicators on a stream .................................... clearenv(3)
Allows servers to authenticate clients ruserok: ......................... ruserok(3)
synchronization of the system clock /corrects the time to allow .......... adjtime(2)
Gets current value of system-wide clock getclock: ................. getclock(3)
Sets value of system-wide time-of-day clock getclock: .............. getclock(3)
Sets the system-wide time-of-day clock stime: ......................... stime(3)
clock: Reports CPU time used ............................................. clock(3)
with a file descriptor close: Closes the file associated with a ...... close(2)
/telldir, seekdir, reindir, the system log syslog, openlog, closelog, setlogmask: Controls syslog(3)
$pclose: Closes a pipe to a process ...................................... pclose(3)
catalog catclose: Closes a specified message .......................... catclose(3)
t close: Closes a transport endpoint ..................................... t_close(3)
fclose, fflush: Closes or flushes a stream ............................. fclose(3)
endprotoent: Closes the /etc/protocols file ............................. endprotoent(3)
entry endservent: Closes the /etc/services file ......................... endservent(3)
file descriptor close: Closes the file associated with a .......... close(2)
endnetent: Closes the networks file ...................................... endnetent(3)
support routines cmdauth: Command authorization ...................... cmdauth(3)
Authorizations Definition file/ routines cmdauth: Format of Command ................................ cmdauth(7)
/file and/ cmdauth: Format of Command Authorization Definition .......... cmdauth(7)
host rexec: Allows execution on a remote host ........................ rexec(3)
system: Executes a shell command ....................................... system(3)
rcmd: Allows execution of socket: Creates an end point for .......... rcmd(3)
Generates a standard interprocess communication and returns a/ ......... socket(2)
communication key ftok: .................................................. ftok(3)
communication objects ..................................................... ipc_la_label(3)
OSF/1 Programmer's Reference

labels on interprocess communication objects .......................................... ipc_slabel(3)
control lists on interprocess communication objects /access ............................ ipc_ac1(3)

pthread_equal: Compares two thread identifiers .......................................... pthread_equal(3)
sigprocmask/ sigblock: Provides a compatibility interface to the sigprocmask(2)
sigpause/ sigpause: Provides a compatibility interface to the sigpause(3)
sigaction( )/ sigvec: Provides a compatibility interface for sigaction(2)
s/sighold, sigrelse, sigignore: Compatibility interfaces for sighold(2)
/Library: Provides functions for sigset(2)
the terminal interface for POSIX
compile, step: Regular-expression compile and match routines ................................... regcomp(3)
compile and match/ advance, erfc: Computes the error and erf(3)
tcdrain: Waits for output to complex absolute value ......................................... tcdrain(3)
/Euclidean distance function and component /address integer ................................... hypot(3)
integer into its host (local) address component integer into its network address ................................... inet_lnaof(3)
dn_expand: Expands a compressed domain name ..................................... dn_expand(3)
dn_skipname: Skips over a compressed domain name ..................................... dn.skipname(3)
dn_comp: Compresses a domain name ................................................. dn_comp(3)
dn_comp: Computes the error and erf(3)
gamma function lgamma, gamma: Computes the logarithm of the gamma(3)
/to floating-point integers, or computes the Modulo Remainder and/ floot(3)
/tan, asin, acos, atan, atan2: Computes the trigonometric and/ sin(3)
initgroups: Initializes a concurrent group set ............................................. initgroups(3)
object /Creates a condition variable attributes .............................................. pthread.condattr_create(3)
object /Deletes a condition variable attributes ............................................. pthread.condattr_delete(3)
 Specifies period of/ /Waits on a condition variable for a ....................................... pthread.cond_timedwait(3)
pthread_cond_destroy: Destroys a condition variable ...................................... pthread.cond_destroy(3)
pthread_cond_init: Creates a condition variable ............................................. pthread.cond_init(3)
pthread_cond_wait: Waits on a condition variable /Wakes up ................................ pthread.cond_broadcast(3)
all threads that are waiting on a condition variable /Wakes ..................................... pthread.cond_signal(3)
up a thread that is waiting on a condition variable /Wakes ..................................... pthread.cond_signal(3)
descriptors poll: Monitors conditions on multiple file poll(2)
sysconf: Gets configurable system variables ................................................. sysconf(3)
resolver: Resolver configuration file .................................................. resolver(4)
about system address space configuration /Gets information ......................... getaddressconf(2)
t_rcvconnect: Receives the connect: Connects two sockets ..................................... connect(2)
t_sndrel: Initiates an endpoint connect sockets ................................................. recv(2)
t_accept: Accepts a connected sockets .................................................. socketpair(2)
t_listen: Listens for a connected sockets .................................................. socketpair(2)
Receives the confirmation from a connected sockets ................................................. socketpair(2)
permuted index

accept: Accepts a new connection on a socket .......................................... accept(2)
user t_connect: Establishes a connection /Receives normal .................................. t_connect(3)
data or expedited data on a data or expedited data over a listen: Listens for socket limits the backlog of incoming
connect: Connects two sockets ............................................ connect(2)

mktemp, mkstemp: Constructs a unique filename ................................ mktemp(3)
temporary file tmpnam, tempnam: Constructs the name for a temporary file tmpnam(3)
Controls maximum system resource variables used by/ signal.h: Contains definitions and
Scans or sorts directory contents scandir, alphasort: scandir(3)
restores the current execution context /longjmp: Saves and
Sets and gets signal stack context sigstack: sigstack(2)
macilb: Mandatory access /putprfinam: Manipulate file control database entry getprfient(3)
/acl: Access/putprlnam: Manipulate printer control database entry getprlent(3)
putprtnam: Manipulate terminal control database entry getprtcent(3)
mandatory: Mandatory access control databases mandatory(4)
tcflow: Performs flow control functions tcflow(3)
acl: Access list conversion functions acl(3)
databases acl: Access control list discretionary policy acl(4)
chacl: Changes the access control list of a file ................................................. chacl(3)
statacl: Retrieves the access control list of a file statacl(3)
shm_chacl: Manipulates access control lists on interprocess/ipc_acl(3)
icmp: Internet Control Message Protocol icmp(7)
msgctl: Performs message control operations .................................................. msgct1(2)
semctl: Performs semaphore operations .................................................. semct1(2)
shmctl: Performs shared memory operations .................................................. shmct1(2)
tcp: Internet transmission protocol .................................................. tcp(7)
Generates the pathname for the controlling terminal ctermid: ctermid(3)
windowing curses Library: Controls cursor movement and curses(3)
octl: Controls devices .................................................. ioctl(2)
getlimit, setlimit: Controls maximum system resource/ ipc_acl(3)
fcntl, dup, dup2: Controls open file descriptors .................................................. fcnt1(2)
lockf: Controls open file descriptors .................................................. lockf(3)
openlog, closelog, setlogmask: Controls the system log syslog, syslog(3)
en: Locale country conversion tables .................................................. en(4)
acl: Access control list conversion routines ns_addr, ns_addr(3)
ns_ntoa: Xerox NS address conversion routines ns_addr, ns_addr(3)
/character classification, case conversion routines ns_addr, ns_addr(3)
double-precision/ atof, strtod: Converts a character string to a atof(3)
the/ atoi, atol, strtol, stroul: Converts a character string to atoi(3)
to a string evct, fcvt, gcvt: Converts a floating-point number evct(3)
or double-byte/ mbstowcs: Converts a multibyte (single-byte mbstowcs(3)
a wide character mbtowc: Converts a multibyte character to mbtowc(3)
a regular/ mrmultdir: Converts a multilevel directory mrmultdir(3)
a multilevel/ mkmultdir: Converts a regular directory into mkmultdir(3)
multibyte character wctomb: Converts a wide character to wctomb(3)
into a/ westombs: Converts a wide character string westombs(3)
header from/ encode_mach_o_hdr: Converts an OSF/ROSE object file encode_mach_o_hdr(3)
(32-bit) integer from htonl: Converts an unsigned long
(32-bit) integer from ntohl: Converts an unsigned long
(16-bit) integer from htons: Converts an unsigned short
(16-bit) integer from ntohs: Converts an unsigned short
strftime: Converts date and time to string
scanf, fscanf, sscanf: Converts formatted input
wscanf: Converts formatted input
from an decode_mach_o_hdr: Converts the canonical header
localtime_r, mktime, tzset: Converts time units /localtime,
/enddvagent, putdvagnam, copydvagent: Manipulate device/
core: Specifies the format of the
memory image file
synchronization of the/ adjtime: Corrects the time to allow
atan2: Computes the/ sin, cos, tan, asin, acos, atan,
functions sinh, cosh, tanh: Computes hyperbolic
en: Locale country convention tables
clock: Reports CPU time used
or writing open, the calling/ pthread_testcancel: Creates a cancellation point in
pthread_cond_init: Creates a condition variable
pthread_condattr_create: Creates a condition variable/
kdir: Creates a directory
mkfifo: Creates a FIFO
I/O server async_daemon: Creates a local NFS asynchronous
pthread_mutexattr_create: Creates a mutex attributes object
pthread_mutex_init: Creates a mutex
fork, vfork: Creates a new process
sockets socketpair: Creates a pair of connected
nfssv: Creates a remote NFS server
tmpfile: Creates a temporary file
object pthread_attr_create: Creates a thread attributes
pthread_create: Creates a thread
entry for an existing file/ link: Creates an additional directory
communication and/ socket: Creates an end point for
mknod: Creates an FIFO or special file
pipe: Creates an interprocess channel
masks /sigdelset, sigismember: Creates and manipulates signal
semget: Returns (and possibly
msgget: Returns (and possibly
shmget: Returns (and possibly
and gets the value of the file
classification, case conversion,/ for the controlling terminal
gmtime_r, asctime, asctime_r,
asctime, asctime_r, ctime,
cbrt: Computes square root and
getwd: Gets
chdir, fchdir: Changes the
getcwd: Gets the pathname of the
endpoint t_look: Looks at the
/longjmp: Saves and restores the
Permuted Index

\textbf{t_unbind}: Disables a transport endpoint \( \text{t_unbind}(3) \)

Binds an address to a transport endpoint \( \text{t_bind}(3) \)

the current event on a transport endpoint \( \text{t_look}(3) \)

protocol options for a transport endpoint \( \text{t_optmgmt}(3) \)

\( /\text{getprdfnam}, \text{setprdfent}, \text{endprdfent}, \text{putprdfnam}: / \)

\( /\text{getprfient}, \text{setprfent}, \text{endprfent}, \text{putprfent}: / \)

\( /\text{getprlpnam}, \text{setprlpent}, \text{endprlpent}, \text{putprlpnam}: / \)

\( /\text{getprpwnam}, \text{setprpwent}, \text{endprpwent}, \text{putprpwent}: / \)

\( /\text{getprtcnam}, \text{setprtcent}, \text{endprtcent}, \text{putprtcnam}: / \)

\( /\text{getpwnam}, \text{putpwent}, \text{setpwent}, \text{endpwent}, \text{putpwent}: / \)

entries endhostent: \( \text{endhostent}(3) \)

...everywhere...

endservent: Closes the endhostent(3)

user/ getusershell, setusershell, getutline, pututline, setutent, setutsid, setutxgetutent(3)

Changes or queries the program's getdirentries: Gets directory getdirentries(2)

utmpname: Accesses utmp file getutent(3)

Ents retrieval of network host endhostent(3)

gethostbyaddr: Gets network gethostbyaddr(3)

gethostbyname: Gets network gethostbyname(3)

getservbyname: Get service getservbyname(3)

getprotobynumber: Gets a protocol getprotobynumber(3)

getservbyport: Gets service getservbyport(3)

/Creates an additional directory entry getprotoent: Gets protocol getprotoent(3)

/entry point for a loaded module ldr_entry(3)

getnetent: Gets network getnetent(3)

getnetbyname: Gets network getnetbyname(3)

getservbyname: Get service getservbyname(3)

getprotobyname: Gets protocol getprotobyname(3)

getservbyport: Gets service getservbyport(3)

/Creates an additional directory entry getprotoent: Gets protocol getprotoent(3)

entry from the /etc/protocols getprotoent(3)

link(2)

Manipulate file control database Manipulate terminal control database

system default database

printer control database

protected password database

Closes the /etc/services file endservent: endservent(3)

execve, execclp, execvp: Executes/ execv(2)

 executes/ environ, execl, execv, execle, environ(5)

getenv: Returns the value of an environment variable getenv(3)

putenv: Sets an environment variable putenv(3)

clearenv: Clears the process clearenv(3)

environ: User environment environ(5)

feof: Tests EOF on a stream feof(3)

mrand48, jrand48, rrand48, erand48, erand48, nrand48, stdrand48, drand48,
complementary error functions erf, erfc: Computes the error and erf(3)
Xerox Network Systems protocol family ns: ............................................................... ns(7)

directory chdir, chmod, chown: Changes the current directory chdir(2)
permissions chmod, chown: Changes file access chmod(2)
group IDs of a file chown, a stream file descriptors
fchown: Changes the owner and file descriptors
fchdir: Changes the current directory chdir(2)
chmod, fchdir: Changes file access chmod(2)
file descriptors
chown, fchown: Changes the owner and file descriptors
chdir(2)
permissions chmod, fchmod: Changes file access permissions chmod(2)

floating-point number to a/ descriptors
ecvt, fcvt, gcvt: Converts a floating-point number to a/ descriptors
ecvt(3)

fdopen, fopen, descriptors
fdopen: Opens a stream descriptors
fopen(3)

feof: Tests EOF on a stream descriptors
feof(3)
ferror: Tests the error indicator descriptors
ferror(3)

nextkey, forder:/ descriptors
/dbminit, fetch, store, delete, firstkey, descriptors
/dbm(3)
extkey, forder:/ descriptors
/dbminit, fetch, store, delete, firstkey, descriptors
/dbm(3)
operations bcopy, bcmp, bzero, character or word from an/ descriptors
fgetc, getchar, getw: Gets a character or word from an/ descriptors
getc(3)
getchar(3)
getw(3)

stream gets, character or word from an/ descriptors
fseek, rewind, ftell, descriptors
fseek(3)
ftell(3)

getwc, fgetwc, getwchar: Gets a character or word from an/ descriptors
getwc(3)
fgetwc(3)
getwchar(3)

mknod: Creates an mknod(3)
mkfifo: Creates a file descriptor...
mknod(3)

Command Authorization Definition descriptor...
/setutent, getutent(3)
data descriptor close: Closes the descriptor close:
data descriptor close: Closes the descriptor close:
data descriptor close: Closes the descriptor close:

getservent: Gets services descriptor...
getservent(3)

setservent: Gets service descriptor...
setservent(3)

endservent: Closes the /etc/services descriptor endservent:
endservent(3)

open, creat: Opens a file file descriptor...
open(2)

/finds the slot in the utmp file descriptor...
open(2)

OSF/ROSE: Object translators descriptors
OSF/ROSE(4)

ar: Archive (library) file format...
ar(4)
tar: Tape archive file format...
tar(4)

getfh: Gets a file handle descriptors getfh(2)

form/ /Converts an OSF/ROSE object file format...
encode_mach_o_hdr(3)

pathconf, fpathconf: Retrieves file implementation/ pathconf(3)
/statilabel: Retrieve a file information label statilabel(3)

truncate, ftruncate: Changes file length truncate(2)

lseek: Moves read-write file offset lseek(2)

getfh: Gets a file handle descriptors getfh(2)

form/ /Converts an OSF/ROSE object file format...
encode_mach_o_hdr(3)

pathconf, fpathconf: Retrieves file implementation/ pathconf(3)
/statilabel: Retrieve a file information label statilabel(3)

tr truncate, ftruncate: Changes file length truncate(2)

lseek: Moves read-write file offset lseek(2)

getfh: Gets a file handle descriptors getfh(2)

form/ /Converts an OSF/ROSE object file format...
encode_mach_o_hdr(3)

pathconf, fpathconf: Retrieves file implementation/ pathconf(3)
/statilabel: Retrieve a file information label statilabel(3)

tr truncate, ftruncate: Changes file length truncate(2)

lseek: Moves read-write file offset lseek(2)

getfh: Gets a file handle descriptors getfh(2)

form/ /Converts an OSF/ROSE object file format...
encode_mach_o_hdr(3)

pathconf, fpathconf: Retrieves file implementation/ pathconf(3)
/statilabel: Retrieve a file information label statilabel(3)

tr truncate, ftruncate: Changes file length truncate(2)

lseek: Moves read-write file offset lseek(2)

getfh: Gets a file handle descriptors getfh(2)
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chpriv</td>
<td>Sets file privileges</td>
</tr>
<tr>
<td>/fstatslabel</td>
<td>Retrieve a file sensitivity label</td>
</tr>
<tr>
<td>memory mmap</td>
<td>Maps file system object into virtual</td>
</tr>
<tr>
<td>stasts, fstasts, usat</td>
<td>Get file system statistics</td>
</tr>
<tr>
<td>Specifies the format of the</td>
<td>Mounts a file system</td>
</tr>
<tr>
<td>mount</td>
<td>Unmounts a file system</td>
</tr>
<tr>
<td>Gets information about a</td>
<td>Mounts or unmounts a file system</td>
</tr>
<tr>
<td>umount</td>
<td>a directory or a file within a file system</td>
</tr>
<tr>
<td>Enables or disables quotas on</td>
<td>a file system</td>
</tr>
<tr>
<td>a hierarchy</td>
<td>Provides information about a file system</td>
</tr>
<tr>
<td>sync</td>
<td>Updates all file systems</td>
</tr>
<tr>
<td>Gets list of all mounted</td>
<td>file systems</td>
</tr>
<tr>
<td>fsync</td>
<td>Writes changes in a header from an OSF/ROSE object</td>
</tr>
<tr>
<td>ftw</td>
<td>Walks a file tree</td>
</tr>
<tr>
<td>exec_with_loader</td>
<td>Executes a file with a loader</td>
</tr>
<tr>
<td>rename</td>
<td>Renames a directory or a file within a file system</td>
</tr>
<tr>
<td>disktab</td>
<td>Disk description</td>
</tr>
<tr>
<td>endnetent</td>
<td>Closes the networks</td>
</tr>
<tr>
<td>group</td>
<td>Group</td>
</tr>
<tr>
<td>mknod</td>
<td>Creates an FIFO or special file</td>
</tr>
<tr>
<td>msync</td>
<td>Synchronizes a mapped file</td>
</tr>
<tr>
<td>read, readv</td>
<td>Reads from a file</td>
</tr>
<tr>
<td>remove</td>
<td>Removes a file</td>
</tr>
<tr>
<td>resolver</td>
<td>Resolver configuration</td>
</tr>
<tr>
<td>rmdir</td>
<td>Removes a directory</td>
</tr>
<tr>
<td>tmpfile</td>
<td>Creates a temporary file</td>
</tr>
<tr>
<td>write, writev</td>
<td>Writes to a file</td>
</tr>
<tr>
<td>conversion, and collating</td>
<td>file /classification, case</td>
</tr>
<tr>
<td>execlp, execvp</td>
<td>Executes a file</td>
</tr>
<tr>
<td>ftw</td>
<td>Walks a file tree</td>
</tr>
<tr>
<td>access</td>
<td>Accessibility of a file</td>
</tr>
<tr>
<td>the access control list of a</td>
<td>Changes</td>
</tr>
<tr>
<td>the information label of a</td>
<td>Changes</td>
</tr>
<tr>
<td>the owner and group IDs of a</td>
<td>Changes</td>
</tr>
<tr>
<td>the sensitivity label of a</td>
<td>Changes</td>
</tr>
<tr>
<td>the format of the memory image</td>
<td>Specifies</td>
</tr>
<tr>
<td>core</td>
<td></td>
</tr>
<tr>
<td>Determine effective access to</td>
<td></td>
</tr>
<tr>
<td>a file</td>
<td></td>
</tr>
<tr>
<td>closes the /etc/protocols</td>
<td></td>
</tr>
<tr>
<td>an advisory lock on an open</td>
<td></td>
</tr>
<tr>
<td>sethost</td>
<td>Mounts network host</td>
</tr>
<tr>
<td>entry from the /etc/protocols</td>
<td></td>
</tr>
<tr>
<td>Opens and rewinds the networks</td>
<td></td>
</tr>
<tr>
<td>Provides information about a</td>
<td></td>
</tr>
<tr>
<td>the access control list of a</td>
<td></td>
</tr>
<tr>
<td>Stopt further I/O to a special</td>
<td></td>
</tr>
<tr>
<td>Makes a symbolic link to a</td>
<td></td>
</tr>
<tr>
<td>the name for a temporary</td>
<td></td>
</tr>
</tbody>
</table>

**Indices**

- `chpriv(3)`
- `fstatslabel(3)`
- `mmap(2)`
- `statfs(2)`
- `fs(4)`
- `mount(3)`
- `umount(3)`
- `rename(2)`
- `setquota(2)`
- `sethostent(3)`
- `setprotoent(3)`
- `ftw(3)`
- `exec_with_loader(2)`
- `rename(2)`
- `disktab(4)`
- `endnetent(3)`
- `group(4)`
- `mknod(2)`
- `msync(2)`
- `read(2)`
- `remove(3)`
- `resolver(4)`
- `rmdir(2)`
- `tmpfile(3)`
- `write(2)`
- `ctab(4)`
- `execlp, execvp(2)`
- `access(2)`
- `chpriv(3)`
- `flock(2)`
- `fhostent(3)`
- `mgetprotoent(3)`
- `setprotoent(2)`
- `setstat(2)`
- `symlink(2)`
- `tmpnam(3)`
- `endprotoent(3)`
- `flock(2)`
- `gethostent(3)`
- `getprotoent(3)`
- `setstat(2)`
- `symlink(2)`
- `tmpnam(3)`
Permuted Index

/the structure of the termios file, which provides the terminal/termios(4)

mcstemp: Constructs a unique file descriptor mktemp, mktemp(3)

passwd: Password files passwd(4)

for the current user ttyslot: Finds the slot in the utmp file ttyslot(3)

dbminit, fetch, store, delete, vector getopt: Gets floating-point numbers to floor, ceil, rint, fmod, fabs: Rounds

frexp, ldexp, modf: Manipulates floating-point numbers

string to a double-precision

advisory lock on an open file

Rounds floating-point numbers to/ floor, ceil, rint, fmod, fabs:

data or nonread input/ tcflush: Flushes nontransmitted output

numbers to/ floor, ceil, rint, omin, stream

/store, delete, firstkey, nextkey, tcsetpgrp: Sets

process

file header from native, readable

OSF/ROSE object file to readable

readable form to canonical

OSF/ROSE: Object file

Definition file and/ cmdauth:

form to canonical form /object encode_mach_o_hdr(3)

form /canonical header from an decode_mach_o_hdr(3)

form /file header from native, encode_mach_o_hdr(3)

format for output from OSF/1/ OSF/ROSE(4)

dir: Format of Command Authorizations cmdauth(7)

Format of directories dir(4)

attributes added to/ Inode: Format of the additional security inode(7)

fs, inode: Specifies the format of the file system volume fs(4)

core: Specifies the format of the memory image file core(4)

format ar(4)

format in a file-systemindependent file.

format /Gets directory entries getdirenteress(2)

for/ vprintf, fprintf, vsprintf:

Formats a varargs parameter list vsprintf(3)

scanf, fscanf, sscanf: Converts formatted input scanf(3)

wscanf: Converts wscanf(3)

printf, fprintf, sprintf: Prints printf(3)

formatted output wpprintf(3)

formatted output

/Retrieves locale-dependent localeconv(3)

ROUTE: Kernel packet forwarding database route(7)

implementation/ pathconf, formatted output printf, or a word to a/ putc, putchar, formatted output printf, or a word to a/ putc, putchar, stream puts, word to a/ putwc, putwchar,
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>putws</td>
<td>Writes a string to a file</td>
</tr>
<tr>
<td>fread, fwrite</td>
<td>Performs input/output operations</td>
</tr>
<tr>
<td>mallinfo, malloc, t_free</td>
<td>Frees a library structure</td>
</tr>
<tr>
<td>rmtimer</td>
<td>Frees a per-process timer</td>
</tr>
<tr>
<td>fopen, fptew</td>
<td>Opens a stream</td>
</tr>
<tr>
<td>fs, inode</td>
<td>Specifies the format of a file system volume</td>
</tr>
<tr>
<td>scanf, fscanf</td>
<td>Converts formatted input</td>
</tr>
<tr>
<td>fseek, rewind, ftell, fgetpos, fsetpos</td>
<td>Repositions a file</td>
</tr>
<tr>
<td>stat, fstat &amp; fstatfs</td>
<td>Provides information about a file system volume</td>
</tr>
<tr>
<td>frexp, ldexp</td>
<td>Manipulates floating-point numbers</td>
</tr>
<tr>
<td>fsetpos</td>
<td>Repositions the file</td>
</tr>
<tr>
<td>ftell, ftell, ftell, ftell, ftell</td>
<td>Frees a library structure</td>
</tr>
<tr>
<td>ftok</td>
<td>Generates a standard key</td>
</tr>
<tr>
<td>ftruncate</td>
<td>Changes file length</td>
</tr>
<tr>
<td>hypot, cabs</td>
<td>Computes Euclidean distance</td>
</tr>
<tr>
<td>perror</td>
<td>Writes a message explaining a function error</td>
</tr>
<tr>
<td>sigblock, sigpause, sigvec</td>
<td>Enables or disables interprocess communication</td>
</tr>
<tr>
<td>gamma, lgamma</td>
<td>Computes the logarithm of the gamma function</td>
</tr>
<tr>
<td>erf, erfc</td>
<td>Computes the error and complementary error functions</td>
</tr>
<tr>
<td>asinh, acosh, atanh</td>
<td>Computes inverse hyperbolic functions</td>
</tr>
<tr>
<td>floor, sqrt, cbrt</td>
<td>Computes the square root and cube root functions</td>
</tr>
<tr>
<td>exp</td>
<td>Computes the exponential function</td>
</tr>
<tr>
<td>pthread_setcancel</td>
<td>Enables or disables the general cancelability of signals</td>
</tr>
<tr>
<td>tgamma, gamma</td>
<td>Computes the gamma function</td>
</tr>
<tr>
<td>gcd, invert</td>
<td>Computes the greatest common divisor</td>
</tr>
<tr>
<td>ecvt, fcvt, gcvt</td>
<td>Converts numerical data</td>
</tr>
<tr>
<td>stopio</td>
<td>Further I/O to a special file</td>
</tr>
<tr>
<td>siginterrupt</td>
<td>Allows signals to interrupt functions</td>
</tr>
<tr>
<td>cosh, sinh</td>
<td>Computes hyperbolic functions</td>
</tr>
<tr>
<td>sqrt, cbrt</td>
<td>Computes the square root and cube root functions</td>
</tr>
<tr>
<td>log10, pow</td>
<td>Computes the logarithm and power functions</td>
</tr>
<tr>
<td>sin</td>
<td>Computes the trigonometric functions</td>
</tr>
<tr>
<td>funlockfile</td>
<td>Unlocks a file</td>
</tr>
<tr>
<td>fsetpos</td>
<td>Repositions the file</td>
</tr>
<tr>
<td>stopio</td>
<td>Further I/O to a special file</td>
</tr>
<tr>
<td>fread, fwrite</td>
<td>Performs input/output operations</td>
</tr>
<tr>
<td>tgamma, gamma</td>
<td>Computes the gamma function</td>
</tr>
<tr>
<td>gcd, invert, mp</td>
<td>Computes the greatest common divisor</td>
</tr>
<tr>
<td>ecvt, fcvt, gcvt</td>
<td>Converts numerical data</td>
</tr>
<tr>
<td>general cancelability of the</td>
<td>Enables or disables the general cancelability of signals</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>tty</strong></td>
<td>General terminal interface</td>
</tr>
<tr>
<td><strong>randomword</strong></td>
<td>Generate random passwords</td>
</tr>
<tr>
<td><strong>random</strong>, <strong>initstate</strong>, <strong>setstate</strong></td>
<td>Generates &quot;better&quot;/ random</td>
</tr>
<tr>
<td><strong>end the current process abort</strong></td>
<td>Generates a software signal to</td>
</tr>
<tr>
<td><strong>communication key ftok</strong></td>
<td>Generates a standard interprocess</td>
</tr>
<tr>
<td><strong>rand</strong>, <strong>rand_r</strong>, <strong>srand</strong></td>
<td>Generates pseudo-random numbers</td>
</tr>
<tr>
<td><strong>controlling terminal ctermid</strong></td>
<td>Generates the pathname for the</td>
</tr>
<tr>
<td><strong>/srand48</strong>, <strong>seed48</strong>, <strong>lcong48</strong></td>
<td>Generates uniformly distributed/</td>
</tr>
<tr>
<td><strong>about system address space/ a character or word from an/ or word from an/ getc, fgetc, getchar, getw</strong></td>
<td>Gets a character</td>
</tr>
<tr>
<td><strong>system-wide clock</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getch</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getc</strong>, <strong>fgetc</strong>, <strong>getchar</strong>, <strong>getw</strong></td>
<td>Gets a character</td>
</tr>
<tr>
<td><strong>process's sensitivity/ getlabel, current directory entries in a/ description using a disk name</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getgfile</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getdtablesize</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getdvagent</strong>, <strong>enddvagent</strong>, <strong>getdvagent</strong>, <strong>getdvagnam</strong></td>
<td></td>
</tr>
<tr>
<td><strong>setdvagent</strong>, <strong>enddvagent</strong>, <strong>getdvagent</strong>, <strong>getdvagnam</strong></td>
<td></td>
</tr>
<tr>
<td><strong>group IDs</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getenv</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getenv</strong></td>
<td></td>
</tr>
<tr>
<td><strong>geteuid</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getuid</strong></td>
<td></td>
</tr>
<tr>
<td><strong>gethostid</strong></td>
<td></td>
</tr>
<tr>
<td><strong>gethostname</strong></td>
<td></td>
</tr>
<tr>
<td><strong>geti1abel</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getitimer</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getlogin</strong>, <strong>getlogin_r</strong>, <strong>setlogin</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getluid</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getnetbyaddr</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getnetbyname</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getnetent</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getopt</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getpagesize</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getpass</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getpeername</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getppid</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getpgrp</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getpwent</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getpuser</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getpwnam</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getpwuid</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getpwcre</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getuid</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getuser</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getuserent</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getuserpw</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getuserpwuid</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getpwent</strong></td>
<td></td>
</tr>
<tr>
<td><strong>getpwuser</strong></td>
<td></td>
</tr>
</tbody>
</table>
the process ID, process group/
getpid, getpid, getppid, getpgid, GETPID, GETPPID, GETPGID: Gets ........................................... getpid(2)
process group/ getpid, getpgid,
setpgid, endpgid, /
setpgid, endpgid, /
setpgid, endpgid, /
setpgid, endpgid, /
setpgid, endpgid, /
setpgid, endpgid, /
entry by protocol name
getprotobyname: Gets protocol
getprotobynumber: Gets a protocol entry by number
from the /etc/protocols file
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprototclnt: Gets protocol entry
getprotoc
Permuted Index

address gethostbyaddr: Gets network host entry by gethostbyaddr(3)

identity: Gets or checks user or group IDs identity(3)

getpriority, setpriority: Gets or sets process scheduling/ priority getpriority(2)
terminal cfgetospeed: Gets output baud rate for a terminal cfgetospeed(3)

sets associated with/ getpriv: Gets privilege or authorization getpriv(3)
times times: Gets process and child process times(3)

name getprotobyname: Gets protocol entry by protocol getprotobyname(3)

/etc/protocols file getprotoent: Gets protocol entry from the /etc/protocols file getprotoent(3)

information t_getinfo: Gets protocol-specific information t_getinfo(3)

getservbyport: Gets service entry by port getservbyport(3)

setservent: Gets service file entry setservent(3)

getservent: Gets services file entry getservent(3)
getsockopt: Gets socket options getsockopt(2)

associated with the/ cuserid: Gets the alphanumeric username cuserid(3)

information label getilabel: Gets the current process’s label getilabel(3)

getslabel, getclmce: Gets the current process’s label getslabel(3)

transport provider t_getstate: Gets the current state of the transport provider t_getstate(3)

getdtablesize: Gets the descriptor table size getdtablesize(2)
ttyname, isatty: Gets the name of a terminal ttyname(3)
system rename: Gets the name of the current system rename(2)

gethostname: Gets the name of the local host gethostname(2)

getpeername: Gets the name of the peer socket getpeername(2)

with the terminal tcgetatt: Gets the parameters associated with the terminal tcgetatt(3)
directory getcwd: Gets the pathname of the current directory getcwd(3)

getgid, getegid: Gets the process group IDs getgid(2)

group/ getgroupld, getgroupld: Gets the process ID, process getgroupld(2)
effective user/ getuid, geteuid: Gets the process’ real or effective user getuid(2)

gethostname: Gets the socket name gethostname(2)

getgroups: Gets the supplementary group set getgroups(2)

getpagesize: Gets the system page size getpagesize(2)
current host gethostid: Gets the unique identifier of the current host gethostid(2)

creation mask umask: Sets and gets the value of the file’s creation mask umask(2)
time: Gets time time(3)

ulimit: Sets and gets user limits ulimit(3)

stream gets, fgets: Gets a string from a stream gets(3)

by name getservbyname: Get service entry getservbyname(3)

by port getservbyport: Gets service entry getservbyport(3)

getservent: Gets services file entry getservent(3)
tcgetpgrp: GetsForeground process group ID tcgetpgrp(3)

current process’s sensitivity/ t_getinfo: Gets the current process’s sensitivity t_getinfo(3)

getsockname: Gets the socket name getsockname(2)

getsockopt: Gets socket options getsockopt(2)

time: Gets and sets date and time gettimeofday, settimeofday, time(3)

gmtime: Gets date and time getmtime(2)

gftime: Gets date and time strftime(3)

process’ real or effective user/ getuid, geteuid: Gets the process’ real or effective user getuid(2)

getsusershell, setusershell, getsusershell(3)

getutent, getutid, getutline, getutent, getutid, getutline: Gets names of endusershell: Gets names of a stream getutent(3)

pututline: Gets names of getutent: pututline(3)

setutent, endutent:/ getutent, getutid, getutline: Gets names of setutent, endutent:/ getutent: endutent(3)

from an/ getc, fgetc, getchar, getw: Gets a character or word getc(3)
character or word from an input/ getwc, fgetwc, getwchar: Gets a
word from an/ getwc, fgetwc,
pathname
a stream/getwc, fgetwc, getwchar: Gets a string from
ctime, ctime_r, difftime,
/time, ctime_r, difftime, gmtime_r, gmtime
Sets jump point for a nonlocal
group/ /pw_idtoname, gr_nametoid,
pw_nametoid, pw_idtoname,
setgroups: Sets the group
getwd: Gets current directory
getws, fgetws: Gets a string from
ctime, ctime_r, difftime, gmtime, gmtime_r, localtime,
gmtime_r, localtime, localtime_r/
siglongjmp: Nonlocal
group/ /pw_idtoname, gr_nametoid,
pw_nametoid, gr_idtoname: Map between userand
setgroups: Sets the group access list
setgid: Sets the group ID
setgid: Sets the group ID
tcgetpgrp: Gets foreground process
setpgrp: Sets the process group ID
Sets the real and effective
groups: Sets the supplementary group set
plates hsearch, hsearch, hcreate, hcreate,
file to/ /Converts the canonical
character sets ascii: Octal,
/Internet address integer into its
an Internet address and
Ends retrieval of network
gethostbyaddr: Gets network
gethostbyname: Gets network
sethostent: Opens network
unique identifier of the current
Gets the name of the local
execution of commands on a remote
getwc, fgetwc, getwchar: Gets a
getwc, fgetwc, getwchar: Gets a character or
getwd: Gets current directory
gets, fgetws: Gets a string from
gmtime, gmtime_r, localtime,
gmtime_r, localtime, localtime_r/
goto with signal handling
getwc(3)
goto sigsetjmp:
sigsetjmp(3)
setgroups(3)
getwc(3)
getwd(3)
getws(3)
ctime(3)
ctime(3)
siglongjmp(3)
sigsetjmp(3)
pw_mapping(3)
pw_mapping(3)
ctime(3)
ctime(3)
ctime(3)
siglongjmp(3)
setgroups(2)
setgid(2)
setsid(2)
tcgetpgrp(3)
setpgrp(2)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgrp(3)
tcgetpgr
Permuted Index

command execution on a remote host
unique identifier of the current host
Sets the name of the current host
/short (16-bit) integer from host-byte order to 2-byte/long (32-bit) integer from host-byte order to Internet network-byte order to host-byte order
hostname: Hostname resolution description
Manages hash tables
(32-bit) integer from host-byte/short (16-bit) integer from host-byte
sinh, cosh, tanh: Computes hyperbolic functions
acosh, atanh: Computes inverse hyperbolic functions
hypot, cabs: Computes Euclidean distance function and complex
security attributes added to i-nodes
select: Synchronous Protocol
/create a local NFS asynchronous server
stopio: Stop further I/O to a special file
Protoocol (and possibly creates) the ID for a message queue
/Returns the next module
/(and possibly creates) the pthread_self: Returns the getuid: Gets the calling thread
setgid: Sets the group
setuid: Sets the user ID
process group ID, parent process ID
(process' real or effective user
a module and returns the module possibly creates) a semaphore
setpgid: Sets the process group
Sets the real and effective group
Getsforeground process group
Sets foreground process group
Sets real and effective user
the process ID, process group
/getppid: Gets the process ID, getentropy: Gets the
gethostid: Gets the unique identifier of the current host
sethostid: Sets the unique
Compares two thread group IDs
Protocol
Changes the owner and group between user and group names and
getegid: Gets the process group
Gets or checks user or group
setegid: Sets the process group
seteuid: Sets the process user IDs

the format of the memory
/pathconf: Retrieves file and limits the backlog of
routines disk: Security receipt of an orderly release
Receives a unit data error
ferror: Tests the error
clearerr: Clears

Internet network address string/inet: Internet Protocol family
Internet address integer into/inet_addr: Translates an
Internet address and host/inet_lnaof: Translates an
Internet address integer into/inet_makeaddr: Translates an
Internet integer address into/inet_netof: Translates an
Internet dot-formatted address/inet_network: Translates an
Internet integer address into/inet_ntoa: Translates an

setfsent, endfsent: Gets information about a file system
stat, fstat, lstat: Provides information about a file
ldr_inq_module: Returns
ldr_inq_region: Returns
getusage, vtimes: Gets
space /getaddressconf: Gets
/endpwent: Accesses the basic user

chilabel: Changes the
fstatilabel: Retrieve a file
Gets the current process's
Sets the current process's
Mandatory access control and
shm_chilabel: Manipulates
nl_langinfo: Language
/t_getinfo: Gets protocol-specific
/t_rcvdis: Retrieves disconnect
concurrent group set
pthread_once: Calls an
file system lmount:
mapped file or shared/ msem_init:

initgroups: Initializes
popen: Initiates a pipe to a process
orderly release t_sndrel:
pthread_cancel: Initiates termination of a thread
disk: Security independent disk
security attributes added to
the file system volume fs,
cfgetispeed: Gets
output data or nonread
case conversion, and collating
Gets a character or word from an
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>getc</td>
<td>Gets a character from an input stream</td>
<td>getc(3)</td>
</tr>
<tr>
<td>unlocked_getc</td>
<td>Gets a character from an unlocked input stream</td>
<td>unlocked_getc(3)</td>
</tr>
<tr>
<td>ungetc</td>
<td>Pushes a character back into an input stream</td>
<td>ungetc(3)</td>
</tr>
<tr>
<td>ungetwc</td>
<td>Inserts or removes an element in a queue</td>
<td>ungetwc(3)</td>
</tr>
<tr>
<td>wsscanf</td>
<td>Converts formatted input</td>
<td>wsscanf(3)</td>
</tr>
<tr>
<td>sscanf</td>
<td>Converts formatted input scansn, fscanf, fgets</td>
<td>scanf(3)</td>
</tr>
<tr>
<td>fwrite</td>
<td>Performs input/output</td>
<td>fwrite(3)</td>
</tr>
<tr>
<td>insque</td>
<td>Inserts or removes an element in a queue</td>
<td>insque(3)</td>
</tr>
<tr>
<td>ldr_install</td>
<td>Installs a module in the current process' private</td>
<td>ldr_install(3)</td>
</tr>
<tr>
<td>pthread_yield</td>
<td>Instead of the current one /the scheduler to run another thread</td>
<td>pthread_yield(3)</td>
</tr>
<tr>
<td>inet_ntoa</td>
<td>Translates an Internet address integer into a host</td>
<td>inet_ntoa(3)</td>
</tr>
<tr>
<td>ntohs</td>
<td>Translates an Internet address and host address into an Internet byte-ordered address</td>
<td>ntohs(3)</td>
</tr>
<tr>
<td>htonl</td>
<td>Converts an Internet network address string to an Internet network integer</td>
<td>htonl(3)</td>
</tr>
<tr>
<td>inet_addr</td>
<td>Converts an Internet network address string to an Internet address string</td>
<td>inet_addr(3)</td>
</tr>
<tr>
<td>labs</td>
<td>Computes the absolute value and division of integers</td>
<td>labs(3)</td>
</tr>
<tr>
<td>floor</td>
<td>Computes the floor of floating-point integers</td>
<td>floor(3)</td>
</tr>
<tr>
<td>nsip</td>
<td>Software network interfaces encapsulating NS</td>
<td>nsip(7)</td>
</tr>
<tr>
<td>termios</td>
<td>Provides the terminal</td>
<td>termios(4)</td>
</tr>
<tr>
<td>sigvec</td>
<td>Provides a compatibility</td>
<td>sigvec(2)</td>
</tr>
<tr>
<td>sigblock</td>
<td>Provides a compatibility</td>
<td>sigblock(2)</td>
</tr>
<tr>
<td>sigpause</td>
<td>Provides a compatibility</td>
<td>sigpause(3)</td>
</tr>
<tr>
<td>lo</td>
<td>Software loopback network</td>
<td>lo(7)</td>
</tr>
<tr>
<td>tty</td>
<td>General terminal</td>
<td>tty(7)</td>
</tr>
<tr>
<td>lvm</td>
<td>Logical Volume Manager (LVM)</td>
<td>lvm(7)</td>
</tr>
<tr>
<td>sigset</td>
<td>Compatibility compatibility</td>
<td>sigset(3)</td>
</tr>
<tr>
<td>swapon</td>
<td>Adds a swap device for interleaved paging and swapping</td>
<td>swapon(2)</td>
</tr>
<tr>
<td>inet_makeaddr</td>
<td>Translates an Internet address and host</td>
<td>inet_makeaddr(3)</td>
</tr>
<tr>
<td>inet_lnaof</td>
<td>Translates an Internet address and host into an Internet byte-ordered address</td>
<td>inet_lnaof(3)</td>
</tr>
<tr>
<td>inet_netof</td>
<td>Translates an Internet address and host into an Internet network address</td>
<td>inet_netof(3)</td>
</tr>
<tr>
<td>/network address string</td>
<td>Internet address integer into its network address</td>
<td>inet_netof(3)</td>
</tr>
<tr>
<td>for a default domain name</td>
<td>Internet address integer into its network address</td>
<td>inet_netof(3)</td>
</tr>
<tr>
<td>/address and host address</td>
<td>Internet byte-ordered address</td>
<td>inet_makeaddr(3)</td>
</tr>
<tr>
<td>icmp</td>
<td>Internet Control Message Protocol</td>
<td>icmp(7)</td>
</tr>
<tr>
<td>idp</td>
<td>Xerox Internet Datagram Protocol</td>
<td>idp(7)</td>
</tr>
<tr>
<td>inet_network</td>
<td>Internet dot-formatted address</td>
<td>inet_network(3)</td>
</tr>
<tr>
<td>inet_ntoa</td>
<td>Internet integer address into a host</td>
<td>inet_ntoa(3)</td>
</tr>
<tr>
<td>inet_addr</td>
<td>Internet network address string</td>
<td>inet_addr(3)</td>
</tr>
<tr>
<td>/integer</td>
<td>Internet network integer into a host</td>
<td>inet_addr(3)</td>
</tr>
<tr>
<td>/to</td>
<td>Internet network-byte order to</td>
<td>ntohl(3)</td>
</tr>
<tr>
<td>/from host-byte order to</td>
<td>Internet network-byte order to</td>
<td>ntohl(3)</td>
</tr>
<tr>
<td>/integer</td>
<td>Internet network-byte order to</td>
<td>ntohl(3)</td>
</tr>
</tbody>
</table>
Permuted Index

privilege sets for/ statpriv: Get kernel authorizations or ................. statpriv(3)

ROUTE: key to be used with/ pthread_keycreate(3)

pthread_keycreate: Creates a interprocess communication

Returns the value bound to a thread-specific value to a
or to a group of processes

installed module from the private
in the current process’ private
the current process’s sensitivity

ilb: Information

mand: Sensitivity

lmount: Initializes a

chilabel: Changes the information

chslabcl: Changes the sensitivity

ilb: Information

Retrieve a file information

the current process’s information

the current process’s information

access control and information

communication/ /information
communication/ /sensitivity
value and division of/ abs, div,

nl_langinfo: Language information

hier: Layout of file systems

/jrand48, srand48, seed48,
floating-point numbers frexp,
division of/ abs, div, labs,
point for a loaded module
information about a loaded/
information about a region in a/
the current process’s private/
address of a symbol name in a/
module ID for a process
module from the private known/
process to permit/
attached process
point for a module loaded in/
another process and returns the/
address of a symbol name within a/
previously loaded in another/
endusershell: Gets names of
character mblen: Determines the
truncate, fntrunc: Changes file
Determines minimum password
getopt: Gets flag
and update lsearch,
logarithm of the gamma function
_t alloc: Allocates a

Kernel packet forwarding database ................. route(7)

key ftok: Generates a standard ................. ftok(3)

key pthread_getspecific: ................. pthread_getspecific(3)

key pthread_setspecific: Binds ................. pthread_setspecific(3)

kill: Sends a signal to a process ................. kill(2)

known package table /Removes an ................. ldr_remove(3)
known packageable /a module ................. ldr_install(3)
label and clearance /Gets ................. getslabel(3)

label functions .................................. ilb(3)

label mount of a file system ................. lmount(3)

label of a file .................................. chilabel(3)

label of a file .................................. chslabcl(3)

label or clearance /Sets ................. setslabel(3)

label .................................. disklabel(4)

label /lstatilabel, fstatilabel: ................. statilabel(3)

label /lstatilabel, fstatilabel: ................. statilabel(3)

label getilabel: Gets ................. getilabel(3)

label setilabel: Sets ................. setilabel(3)

labeling macilb: Mandatory ................. macilb(4)

labels on interprocess ................. ipc_label(3)

labels on interprocess ................. ipc_label(3)

labs, ldiv: Computes absolute ................. abs(3)

getilabel: Gets ................. getilabel(3)

setilabel: Sets ................. setilabel(3)

getilabel: Gets ................. getilabel(3)

setilabel: Sets ................. setilabel(3)

setslabel(3)

setslabel(3)

setslabel(3)

setslabel(3)

setslabel(3)

ldiv: Computes absolute value and ................. abs(3)

ldr_entry: Returns the entry ................. ldr_entry(3)

ldr_inq_module: Returns ................. ldr_inq_module(3)

ldr_inq_region: Returns module ................. ldr_inq_region(3)

ldr_install: Installs a module in ................. ldr_install(3)

ldr_lookup_package: Returns the ................. ldr_lookup_package(3)

ldr_next_module: Returns the next ................. ldr_next_module(3)

ldr_remove: Removes an installed ................. ldr_remove(3)

ldr_xattach : Attaches to another ................. ldr_xattach(3)

ldr_xdetach: Detaches from an ................. ldr_xdetach(3)

ldr_xentry: Returns the entry ................. ldr_xentry(3)

ldr_xload: Loads a module in ................. ldr_xload(3)

ldr_xlookup_package: Returns the ................. ldr_xlookup_package(3)

ldr_xunload: Unloads a module ................. ldr_xunload(3)

legal user shells /setusershell, ................. getusershell(3)

length in bytes of a multibyte ................. mblen(3)

length ................. truncate(2)

length passlen: ................. passlen(3)

letters from the argument vector ................. getopt(3)

lfind: Performs a linear search ................. lsearch(3)

lgamma, gamma: Computes the ................. gamma(3)

library structure ................................ t alloc(3)
t_free: Frees a library structure ......................................... t_free(3)
t_sync: Synchronizes transport and windowing curses
Programmers Workbench
/library: Controls cursor movement ....................... curses(3)
library: Provides functions for ......................... libPW(3)
limits the backlog of incoming/ ...................... listen(2)
limits ...................................................... ulimit(3)
link to a file ............................................. symlink(2)
link_readlink: ......................................... readlink(2)
link: Creates an additional ............................ link(2)
list conversion functions ............................. acl(3)
list discretionary policy ............................... acl(4)
list of a file ............................................. chacl(3)
list of all mounted file systems .................... getfsstat(2)
list varargs: Handles ................................. varargs(3)
listen: Listens for socket .............................. listen(2)
listen: Listens for socket connections ............. listen(2)
listens for a connect request ........................ t_listen(3)
lists on interprocess.................................. ipc_ac1(3)
lo: Software loopback network ...................... lo(7)
load: Loads a module and returns .................... load(3)
load: Loads a module in another process .......... load(3)
load: Unloads a previously loaded module ........ load(3)
load module ............................................. ldr_inq_module(3)
load module ldr_xentry: ............................. ldr_entry(3)
load module ldr_inq_module: ......................... ldr_module(3)
exec_with_loader: ..................................... exec_with_loader(2)
loading/unloading of modules in ...................... ldr_xattach(3)
loads a module in another process ................ ldr_xload(3)
gethostname: Gets the name of the
local host .............................................. gethostname(2)
gethostname: Creates a case conversion, and/ctab:
Locale character classification, .................... ctab(4)
Locale country convention tables ................ en(4)
locale or portions thereof ......................... setlocale(3)
locale-dependent formatting .................. localeconv(3)
localeconv, localeconv_r: ........................ localeconv(3)
localeconv_r: Retrieves ............................. localeconv(3)
localtime, localtime_r, mktime, ...
time(3)
localtime, localtime_r, mktime, tsetime ........ ctime(3)
live a program end, .................................. end(5)
live a mutex ........................................... pthread_mutex_trylock(3)
live an open file flock: ............................. flock(2)
lockf: Controls open file ........................... lockf(3)
locking ............................................... tod(3)
/tries once to

LOCAL HOST................. async_daemon(2)
local NFS asynchronous I/O server ................ async_daemon(2)
localeconv, localeconv_r: Retrieves
localeconv(3)
localeconv_r: Retrieves ............................. localeconv(3)
localtime, localtime_r, mktime, /
time(3)
localtime_r, mktime, tsetime ........ ctime(3)
live a program end, .................................. end(5)
live a mutex ........................................... pthread_mutex_trylock(3)
live an open file flock: ............................. flock(2)
lockf: Controls open file ........................... lockf(3)
locking ............................................... tod(3)
/tries once to

LOCAL HOST................. async_daemon(2)
local NFS asynchronous I/O server ................ async_daemon(2)
localeconv, localeconv_r: Retrieves
localeconv(3)
localeconv_r: Retrieves ............................. localeconv(3)
localtime, localtime_r, mktime, /
time(3)
localtime_r, mktime, tsetime ........ ctime(3)
live a program end, .................................. end(5)
live a mutex ........................................... pthread_mutex_trylock(3)
live an open file flock: ............................. flock(2)
lockf: Controls open file ........................... lockf(3)
locking ............................................... tod(3)
/tries once to

LOCAL HOST................. async_daemon(2)
local NFS asynchronous I/O server ................ async_daemon(2)
localeconv, localeconv_r: Retrieves
localeconv(3)
localeconv_r: Retrieves ............................. localeconv(3)
localtime, localtime_r, mktime, /
time(3)
localtime_r, mktime, tsetime ........ ctime(3)
live a program end, .................................. end(5)
live a mutex ........................................... pthread_mutex_trylock(3)
live an open file flock: ............................. flock(2)
lockf: Controls open file ........................... lockf(3)
locking ............................................... tod(3)
/tries once to
a message from a socket using a message structure sendmsg: Sends message to a message queue action sendmsg(2)
mmsgnd: Sends a message to a message queue action sendmsg(2)
t_error: Produces error message action t_error(3)
res_mkquery: Makes query messages for name servers action res_mkquery(3)
recv: Receives messages from connected sockets action recv(2)
recvfrom: Receives messages from sockets action recvfrom(2)
send: Sends messages on a socket action send(2)
sendto: Sends messages through a socket action sendto(2)
invert, rpow, msqrt, mcmp, move, passlen: Determines minimum password length action passlen(3)
directory into a multilevel/file naming mkmultidir: Converts a regular file action mkmultidir(3)
filename mktemp, mkstemp: Constructs a unique filename action mktemp(3)
mkdir: Creates a directory action mkdir(2)
mkfifo: Creates a FIFO action mkfifo(3)
mktemp, mktemp: Constructs a unique filename action mktemp(3)
mktimer: Allocates a per-process timer action mktimer(3)
directory into virtual memory action mld(3)
mmap: Maps file system object action mmap(2)
modf: Manipulates floating-point numbers action modf(3)
umtimes: Sets file access and modification times action utime(2)
mprotect: Modifies access protections of memory mapping action mprotect(2)
load: Loads a module and returns the module ID action load(3)
ldr_remove: Removes an installed module from the private known/ action ldr_remove(3)
ldr_next_module: Returns the next module ID /Loads a module in another process and returns the module ID load(3)
Loads a module and returns the module ID /Loads a module in another process and returns the module ID load(3)
returns the/ ldr_xload: Loads a module in another process and returns the module ID load(3)
private/ ldr_install: Installs a module in a/ action ldr_install(3)
in a/ ldr_inq_region: Returns module information about a region action ldr_inq_region(3)
/Returns the entry point for a region in a loaded module as a file system action ldr_xunload: Unloads a module previously loaded /to permit loading/unloading of modules in that process' address space/ action ldr_xattach(3)
/integers, or computes the Modulo Remainder and/ action floor(3)
poll: Monitors conditions on multiple file descriptors action poll(2)
Imount: Initializes a label for a file system action Imount(3)
requests exports: Defines remote mount points for NFS mount action exports(4)
remote mount points for NFS mount requests exports: Defines remote mount points for NFS mount action exports(4)
getfsstat: Gets list of all mount points for a file system action getfsstat(2)
mount, umount: Mounts or unmounts a file system action mount(2)
mount: Mounts a file system action mount(3)
mounted file systems action getfsstat(2)
Mounts a file system action mount(3)
Mounts or unmounts a file system action mount(2)
mount, umount: Mounts or unmounts a file system action mount(2)
mout, omout, fmout, m_out, sdiv, / action mp(3)
move, min, omin, fmin, m_in, / action mp(3)
movement and windowing action curses(3)
OSF/1 Programmer’s Reference

lseek: Moves read-write file offset .................................................. lseek(2)
protections of memory mapping mprotect: Modifies access ................................ mprotect(2)
/erand48, lrand48, nrand48,.................................................................... erand48(3)
semaphore in a mapped file or/ msem_init: Initializes a .......................... msem_init(3)
sem_lock: Locks a semaphore ............................................................... msem_lock(3)
sem_remove: Removes a semaphore .................................................. msem_remove(3)
sem_unlock: Unlocks a semaphore ..................................................... msem_unlock(3)

msem_chacl,/ msg_statacl,................................................................. msem_chacl(3)
msg_statilabel,............................................................ msg_statilabel(3)
msg_clabel,............................................................... msg_clabel(3)
sem_chacl, sem_chacl,/................................................................. sem_chacl(3)
sem_chilabel,............................................................. sem_chilabel(3)
sem_chacl, sem_chacl,/................................................................. sem_chacl(3)
sem_chilabel,sem_chilabel,/....................................................... sem_chilabel(3)
operations msgctl: Performs message control .............................. msgctl(2)
creates) the ID for a message/ msgget: Returns (and possibly .............. msgget(2)
message queue msgrcv: Receives a message from a ..................... msgrcv(2)
message queue

/mdiv, pow, gcd, invert, rpow,...................................................... mdiv(3)
invert, rpow, msqrt, mcmp,/ madd,.............................................. invert(3)
rpow, msqrt, mcmp,/ madd, msub,.................................................. msqrt(3)
mbstowcs: Converts a multibyte (single-byte or/.......................... mbstowcs(3)
character mbtowc: Converts a multibyte character to a wide .............. mbtowc(3)
the length in bytes of a multibyte character /Determines ................ mblen(3)
Converts a wide character into a multibyte character wctomb: ................ wctomb(3)
a wide character string into a multibyte character string .................. wcstombs(3)
regular/ rmmultdir: Converts a................................................. rmmultdir(3)
mld: Traverse a regular directory into a........................................... mld(3)
Checks to see if a directory is a regular directory into a ................... mld(3)
poll: Monitors conditions on......................................................... poll(3)
/select: Synchronous I/O ............................................................... poll(3)

/mkdir, pw, gdc, / madd,...................................................... mkdir(3)
msgid_ds: Defines a message queue .............................................. msgid_ds(3)
msgsnd: Sends a message to a ..................................................... msgsnd(3)
msgqget: Returns (and possibly .............................................. msgqget(3)
msgqrcv: Receives a message from a ................................. msgqrcv(3)
semaphore in a mapped file ..................................................... msgqrcv(3)

/mdiv, pow, gcd, invert, rpow,...................................................... mdiv(3)
multibyte single-byte or/.......................................................... mdiv(3)
multibyte character to a wide .................................................. mdiv(3)
multibyte character /Determines .............................................. mdiv(3)
/multibyte character wctomb: .................................................. multibyte character wctomb(3)
a wide character string ......................................................... wctomb(3)
/a wide character string into a multibyte character string .................. wcstombs(3)
regular/ rmmultdir: Converts a................................................. rmmultdir(3)
mb: Traverse a regular directory into a........................................... mb(3)
Checks to see if a directory is a regular directory into a ................... mb(3)
poll: Monitors conditions on......................................................... poll(3)
/select: Synchronous I/O ............................................................... poll(3)

/mkdir, pw, gdc, / madd,...................................................... mkdir(3)
multibyte single-byte or/.......................................................... mdiv(3)
multibyte character to a wide .................................................. mdiv(3)
multibyte character /Determines .............................................. mdiv(3)
/multibyte character wctomb: .................................................. multibyte character wctomb(3)
a wide character string ......................................................... wctomb(3)
/a wide character string into a multibyte character string .................. wcstombs(3)
regular/ rmmultdir: Converts a................................................. rmmultdir(3)
mb: Traverse a regular directory into a........................................... mb(3)
Checks to see if a directory is a regular directory into a ................... mb(3)
poll: Monitors conditions on......................................................... poll(3)
/select: Synchronous I/O ............................................................... poll(3)

/creates a mutex attributes object.......................................... phrthread_mutexattr_create(3)
/Delegates a mutex attributes object........................................ phrthread_mutexattr_delete(3)
 pthread_mutex_destroy: Deletes a............................................ pthread_mutex_destroy(3)
 pthread_mutex_init: Creates a.................................................. pthread_mutex_init(3)
 pthread_mutex_lock: Locks a..................................................... pthread_mutex_lock(3)
 pthread_mutex_unlock: Unlocks a.............................................. pthread_mutex_unlock(3)
 Tries once to lock a validity.............................................. pthread_mutex_trylock(3)
/Validates a memory region for............................................ pthread_mutex_valid(3)
/Validates a memory region for............................................ pthread_mutex_valid(3)
/Validates a memory region for............................................ pthread_mutex_valid(3)
 name and Internet address ................................................. res_init(3)
/Validates a memory region for............................................ res_init(3)
 name database ......................................................... protocols(4)
 services: Service name database ................................................ services(4)
tmpnam, tempnam: Constructs the............................................. tempnam(3)
/Validates a memory region for............................................ tempnam(3)
 name in a package ......................................................... ldr_lookup_package(3)
/Validates a memory region for............................................ ldr_lookup_package(3)
 name of a terminal ......................................................... ttyname(3)
/Validates a memory region for............................................ ttyname(3)
 name of the current host .................................................... sethostname(2)
/Validates a memory region for............................................ sethostname(2)
 name of the current system .................................................. uname(2)
/Validates a memory region for............................................ uname(2)
nl_langinfo: Language information ...................... nl_langinfo(3)
Nonlocal goto with signal ..................................... siglongjmp(3)
nonlocal goto .................................................... siglongjmp(3)
nonread input data /Flushes .................................... tcflush(3)
nontransmitted output data or ................................. tcflush(3)
normal data or expedited data on ............................ t_rv(3)
normal data or expedited data ................................. t_snd(3)
nrand48, mrand48, jrand48, i_ran(3)
NS address conversion routines ............................... ns_addr(3)
NS packets in IP packets ....................................... nsip(7)
n: Xerox Network Systems ..................................... ns(7)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS ..................................... ns_addr(3)
n_addr, ns_ntoa: Xerox NS address .......................... ns_addr(3)
nsip: Converts an unsigned long ......................... ntohl(3)
nsip: Converts an unsigned long ......................... ntohl(3)
nsip: Converts an unsigned long ......................... ntohl(3)
nip: Data sink ................................................... null(7)
network interface encapsulating protocol family
address conversion routines ns_addr, ns_ntoa:
encapsulating NS packets in IP/(32-bit) integer from Internet/
short (16-bit) integer from/
distributed pseudo-random gcvt: Converts a floating-point
Gets a protocol entry by
isnan: Tests for NaN (Not a
fmod, fabs: Rounds floating-point
modf: Manipulates floating-point
rand: Generates pseudo-random from OSF/1 translators OSF/ROSE:
readable/ /Converts an OSF/ROSE
canonical header from an OSF/ROSE
mmap: Maps file system
a condition variable attributes
a condition variable attributes
or privilege sets for an
attribute of a thread attributes
attribute of a thread attributes
Creates a thread attributes
Deletes a thread attributes
Creates a mutex attributes
Deletes a mutex attributes
on interprocess communication
on interprocess communication
on interprocess communication
ASCII character sets ascii:
iseek: Moves read-write file
/pow, msqrt, mcnp, move, min,
/min, omin, fmin, m_in, mout,
 pthread_mutex_trylock: Tries calling thread /Pushes a routine
data on a record basis audit:
fcntl, dup, dup2: Controls
lockf: Controls
or removes an advisory lock on an
reading or writing

OSF/1 Programmer’s Reference
ROUTE: Kernel packet forwarding database .................................. route(7)
spp: Xerox sequenced packet protocol (SPP) ........................................ spp(7)
interface encapsulating NS packets in IP packets /network .................. nsip(7)
encapsulating NS packets in IP packets /network interface .................. nsip(7)
getpagesize: Gets the system page size ........................................... getpagesize(2)
a swap device for interleaved paging and swapping /swapon: Adds .......... swapon(2)
the system of a process' expected socketpair: Creates a pair of connected sockets /socketpair: Adds ........... swapon(2)
s/ v/multipathconf: Formats a varargs parameter list for output ............ vsprintf(3)
Handles a variable-length parameter list /varargs: ................................ varargs(3)
terminal tcgetattr: Gets the parameters associated with the process ID, process group ID, /tcgetattr(3)
terminal tcsetattr: Sets the parameters associated with the process ID, password length /tcsetattr(3)
locale-dependent formatting /localeconv(3)
the process ID, process group ID, /getppid: Gets .................. getpid(2)
password: Determines minimum password length ................................ passlen(3)
passwd: Password files .................................................................. passwd(4)
passlen: Determines minimum password length ................................ passlen(3)
/putprpwent: Manipulate protected passwords .................................... passwd(4)
passwd: Password files .................................................................. passwd(4)
/putprpwent: Manipulate protected passwords .................................... passwd(4)
passlen: Determines minimum password length ................................ passlen(3)
requirements /Determines if a password meets deduction ...................... acceptable_password(3)
getpass: Reads a password .................................................................. getpass(3)
randomword: Generate random passwords .......................................... randomword(3)
file implementation/ pathconf, fpathconf: Retrieves ...................... pathconf(3)
terminal ctermid: Generates the pathname for the controlling process /ctermid(3)
getcwd: Gets the pathname of the current directory ........................ getcwd(3)
getwd: Gets current directory pathname ............................................ getwd(3)
signal is received pause: Suspends a process until a signal is received .......... pause(3)
process getpeername: Gets the name of the peer socket .................... getpeername(2)
sigpending: Examines pending signals ............................................ sigpending(2)
msktimer: Allocates a per-process timer ........................................... mktimer(3)
rmtimer: Frees a per-process timer .................................................. rmtimer(3)
timeout intervals of a per-process timer /Establishes privileges: ................................ privileges(3)
update bsearch: Performs a binary search ......................................... bsearch(3)
lssearch, lfind: Performs a linear search and ................................. lssearch(3)
bcopy, bcmp, bzero, ffs: Performs bit and byte string/ .................. bcopy(3)
tcfflow: Performs flow control functions ......................................... tcflow(3)
freed, fwrite: Performs input/output ............................................ fread(3)
/memcmp, memcpy, memset, memmove: Performs memory operations .......... memccpy(3)
operations msgsctl: Performs message control .................................. msgsctl(2)
/omout, fmout, m_out, sdiv, itom: Performs multiple precision/ operations mp(3)
/strtoc, strtof, r, strxfm: Performs operations on strings .................. string(3)
/wstrchr, wstrspn, wstrtok: Performs operations on wide/ .................. wstring(3)
seekdir, rewinddir, closedir: Performs operations on/ /telldir, opendir(3)
operations semct: Performs semaphore control .................................. semctl(2)
semop: Performs semaphore operations ........................................... semop(2)
operations shmct: Performs shared memory control .......................... shmctl(2)
variable for a specified period of time /on a condition .................. pthread_cond_timedwait(3)
Writes changes in a file to permanent storage fsync: ...................... fsync(2)
fchmod: Changes file access permissions chmod, .......................... chmod(2)
/: Attaches to another process to permit loading/unloading of/
    explaining a function error
    pclose: Closes a
    popen: Initiates a
        channel
    qsort: Sorts a table in
        the byte stream putlong:
        the byte stream putshort:
            and/or data segments in memory
    ldr_entry: Returns the entry
    ldr_xentry: Returns the entry
        sigsetjmp: Sets jump
    returns a/ socket: Creates an end
        /Creates a cancellation
    fsetpos: Repositions the file
    fileno: Maps stream pointer to file descriptor
    exports: Defines remote mount
    routines spdbm: Security
    Access control list discretionary
    multiple file descriptors
        process
    Gets service entry by
    entire current locale or
    the terminal interface for
    semget: Returns (and
    message/ msgget: Returns (and
    shared/ shmat: Returns (and
    mcmp, madd, msub, mdiv,
    logarithm, and/ exp, log, log10,
    exponential, logarithm, and
    /sddiv, itom: Performs multiple
    ldr_xunload: Unloads a module
    unload: Unloads a
    /putprlpnam: Manipulate
        formatted output
    printf, fprintf, sprintf:
        wprintf:
    nice: Changes scheduling
    Gets or sets process scheduling
    /an installed module from the
    /a module in the current process’
    privileges: Perform
    associated with/ getpriv: Gets
    /Get kernel authorizations or
    Retrieves a socket with a
    chpriv: Sets file
    Sets kernel authorizations and
    bracketing
    acct: Enables and disables
    times: Gets
    /Loads a module in another
    permit loading/unloading of/
    perror: Writes a message
    pipe to a process
    pipe to a process
    pipe: Creates an interprocess
        place
    Places long byte quantities into
    Places short byte quantities into
    block: Locks a process’ text
    point for a loaded module
    point for a module loaded in/
    point for a nonlocal goto
    point for communication and
    point in the calling thread
    fsetpos, /fgetpos, fseek
    pointer to file descriptor
    pointer to a stream
    points for NFS mount requests
    policy database management
    policy databases acl:
    poll: Monitors conditions on
    poll
    popen: Initiates a pipe to a
    portions thereof /the program’s
    POSIX compatibility /provides
    possibly creates a semaphore ID
    possibly creates the ID for a
    possibly creates the ID for a
    mcmp, gcd, invert, rpow, msqrt,
    logarithm, and/ exp, log, log10,
    exponential, logarithm, and
    /sdiv, itom: Performs multiple
    ldr_xunload: Unloads a module
    unload: Unloads a
    /putprlpnam: Manipulate
        formatted output
    printf, fprintf, sprintf:
        wprintf:
    nice: Changes scheduling
    Gets or sets process scheduling
    /an installed module from the
    /a module in the current process’
    privileges: Perform
    associated with/ getpriv: Gets
    /Get kernel authorizations or
    Retrieves a socket with a
    chpriv: Sets file
    Sets kernel authorizations and
    bracketing
    acct: Enables and disables
    times: Gets
    /Loads a module in another
    priority /setpriority:
    private known package table
    private known packagetable
    privilege bracketing
    privilege or authorization sets
    privilege sets for an object
    privilege address resvport:
    privileges
    privileges setpriv:
    privileges: Perform privilege
    process accounting
    process and child process times
    process and returns the module ID
    ldr_xload(3)
    ldr_xattach(3)
    perror(3)
    pclose(3)
    popen(3)
    pipe(2)
    qsort(3)
    putlong(3)
    putshort(3)
    plock(2)
    ldr_entry(3)
    ldr_xentry(3)
    sigsetjmp(3)
    pthread_testcancel(3)
    fsetpos(3)
    fileno(3)
    exports(4)
    spdbm(3)
    acl(4)
    poll(2)
    popen(3)
    getServbyport(3)
    setlocale(3)
    termios(4)
    semget(2)
    msgget(2)
    shmat(2)
    mp(3)
    exp(3)
    mp(3)
    ldr_xunload(3)
    unload(3)
    getprlpent(3)
    printfs(3)
    prinft(3)
    printf(3)
    wprintf(3)
    nice(3)
    getpriority(2)
    ldr_remove(3)
    ldr_install(3)
    privileges(3)
    getpriv(3)
    statpriv(3)
    rresvport(3)
    chpriv(3)
    privileges(3)
    setpriv(3)
    privileges(3)
    acct(2)
    times(3)
    ldr_xload(3)
<table>
<thead>
<tr>
<th>Provides functions for/</th>
<th>Programmers Workbench Library: libPW(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lvm: Logical Volume Manager (LVM)</td>
<td>programming interface lvm(7)</td>
</tr>
<tr>
<td>for compatibility with existing</td>
<td>programs</td>
</tr>
<tr>
<td>/putprpname: Manipulate</td>
<td>protected password database entry getprpwent(3)</td>
</tr>
<tr>
<td>name subsys_real_name: Map a</td>
<td>protected subsystem group to its subsys_real_name(3)</td>
</tr>
<tr>
<td>mprotect: Modifies access</td>
<td>protections of memory mapping mprotect(2)</td>
</tr>
<tr>
<td>spp: Xerox sequenced packet</td>
<td>protocol (SPP) spp(7)</td>
</tr>
<tr>
<td>udp: Internet user datagram</td>
<td>protocol (UDP) udp(7)</td>
</tr>
<tr>
<td>getprotobynumber: Gets a</td>
<td>protocol entry by number getprotobynumber(3)</td>
</tr>
<tr>
<td>getprotobynname: Gets</td>
<td>protocol entry by protocol name getprotobynname(3)</td>
</tr>
<tr>
<td>/etc/protocols/ getprotoent: Gets</td>
<td>protocol entry from the getprotoent(3)</td>
</tr>
<tr>
<td>inet: Internet</td>
<td>Protocol family inet(7)</td>
</tr>
<tr>
<td>ns: Xerox Network Systems</td>
<td>protocol family ns(7)</td>
</tr>
<tr>
<td>protocols:</td>
<td>Protocol name database protocols(4)</td>
</tr>
<tr>
<td>t_optmgmt: Manages</td>
<td>protocol name getprotobynname: getprotobynname(3)</td>
</tr>
<tr>
<td>endpoint</td>
<td>protocol options for a transport t_optmgmt(3)</td>
</tr>
<tr>
<td>icmp: Xerox Internet Datagram</td>
<td>Protocol icmp(7)</td>
</tr>
<tr>
<td>idp: Xerox Internet Datagram</td>
<td>Protocol idp(7)</td>
</tr>
<tr>
<td>ip: Internet</td>
<td>Protocol ip(7)</td>
</tr>
<tr>
<td>Internet transmission control</td>
<td>protocol tcp: tcp(7)</td>
</tr>
<tr>
<td>t_getinfo: Gets</td>
<td>protocol-specific information t_getinfo(3)</td>
</tr>
<tr>
<td>current state of the transport</td>
<td>protocols: Protocol name database protocols(4)</td>
</tr>
<tr>
<td>interface to the/ sigblock:</td>
<td>provider t_getstate: Gets the t_getstate(3)</td>
</tr>
<tr>
<td>interface to the/ sigpause:</td>
<td>Provides a compatibility sigblock(2)</td>
</tr>
<tr>
<td>interface to the/ sigvec:</td>
<td>Provides a compatibility sigpause(3)</td>
</tr>
<tr>
<td>/malloc, mallocinfo, alloc:</td>
<td>Provides a memory allocator malloc(3)</td>
</tr>
<tr>
<td>Programs Workbench Library:</td>
<td>Provides functions for/ libPW(3)</td>
</tr>
<tr>
<td>stat, fstat, lstat:</td>
<td>Provides information about a file stat(2)</td>
</tr>
<tr>
<td>for/ of the termios file, which</td>
<td>provides the terminal interface termios(4)</td>
</tr>
<tr>
<td>pty:</td>
<td>Pseudo terminal driver pty(7)</td>
</tr>
<tr>
<td>Generates uniformly distributed</td>
<td>pseudo-random number sequences drand48(3)</td>
</tr>
<tr>
<td>rand, rand_r, srand: Generates</td>
<td>pseudo-random numbers rand(3)</td>
</tr>
<tr>
<td>thread attributes object</td>
<td>pthread_attr_create: Creates a pthread_attr_create(3)</td>
</tr>
<tr>
<td>thread attributes object</td>
<td>pthread_attr_delete: Deletes a pthread_attr_delete(3)</td>
</tr>
<tr>
<td>Returns the value of the stack/</td>
<td>pthread_attr_getstacksize: pthread_attr_getstacksize(3)</td>
</tr>
<tr>
<td>the value of the stack size/</td>
<td>pthread_attr_setstacksize: Sets pthread_attr_setstacksize(3)</td>
</tr>
<tr>
<td>termination of a thread</td>
<td>pthread_cancel: Initiates pthread_cancel(3)</td>
</tr>
<tr>
<td>routine from the top of the/</td>
<td>pthread_cleanup_pop: Removes a pthread_cleanup_pop(3)</td>
</tr>
<tr>
<td>routine onto the cleanup stack/</td>
<td>pthread_cleanup_push: Pushes a pthread_cleanup_push(3)</td>
</tr>
<tr>
<td>all threads that are waiting on/</td>
<td>pthread_cond_broadcast: Wakes up pthread_cond_broadcast(3)</td>
</tr>
<tr>
<td>condition variable</td>
<td>pthread_cond_destroy: Destroys a pthread_cond_destroy(3)</td>
</tr>
<tr>
<td>condition variable</td>
<td>pthread_cond_init: Creates a pthread_cond_init(3)</td>
</tr>
<tr>
<td>thread that is waiting on a/</td>
<td>pthread_cond_signal: Wakes up pthread_cond_signal(3)</td>
</tr>
<tr>
<td>a condition variable for a/</td>
<td>pthread_cond_timedwait: Waits on pthread_cond_timedwait(3)</td>
</tr>
<tr>
<td>condition variable</td>
<td>pthread_cond_wait: Waits on a pthread_cond_wait(3)</td>
</tr>
<tr>
<td>a condition variable attributes/</td>
<td>pthread_condattr_create: Creates pthread_condattr_create(3)</td>
</tr>
<tr>
<td>a condition variable attributes/</td>
<td>pthread_condattr_delete: Deletes pthread_condattr_delete(3)</td>
</tr>
<tr>
<td>thread identifiers</td>
<td>pthread_create: Creates a thread pthread_create(3)</td>
</tr>
</tbody>
</table>

| thread identifiers | pthread_detach: Detaches a thread pthread_detach(3) |

| thread identifiers | pthread_equal: Compares two pthread_equal(3) |
calling thread
value bound to a key
to terminate
to be used with thread-specific/
mutex
to lock a mutex
a mutex attributes object
a mutex attributes object
initialization routine
the calling thread
or disables the asynchronous/
disables the general/
thread-specific value to a key
cancellation point in the/
scheduler to run another thread/
child process
input stream ungetc, ungetwc:
stack of/ pthread_cleanup_push:
Writes a character or a word to:/ character or a word to a/ putc,
/setdvagent, enddvagent,
variable
quantities into the byte stream
default/ /setprfent, endprfent,
control/ /setprfent, endprfent,
control/ /setprfent, endprlent,
password/ /setprlent, endprlent,
control/ /setprlent, endprlent,
getpwent, getpwuid, getpwnam,
stream
quantities into the byte stream
getutent, getutid, getutline,
word to a/ putc, putchar, fputc,
character or a word to a stream
character or a word to a/ putwc,
a stream
gd_idtoname: Map/ pw_nametoid,
gd_nametoid, gd_idtoname: Map/
_getlong: Retrieves long
_getshort: Retrieves short
putlong: Places long byte
putshort: Places short byte
current/ setlocale: Changes or
res_mkquery: Makes
retrieves a/ res_send: Sends a
msqid_ds: Defines a message

pthread_exit: Terminates the
pthread_getspecific: Returns the
pthread_join: Waits for a thread
pthread_keycreate: Creates a key
pthread_mutex_destroy: Deletes a
pthread_mutex_init: Creates a
pthread_mutex_lock: Locks a mutex
pthread_mutex_trylock: Tries once
pthread_mutex_unlock: Unlocks a
pthread_mutexattr_create: Creates
pthread_mutexattr_delete: Deletes
pthread_mutexattr_destroy: Deletes
pthread_mutex_destroy: Deletes
pthread_mutex_init: Creates
pthread_mutex_lock: Locks
pthread_mutex_trylock: Tries
pthread_mutex_unlock: Unlocks
pthread_once: Calls an
pthread_self: Returns the ID of
pthread_setspecific: Binds a
pthread_setspecific_value: Binds
pthread_testcancel: Creates a
pthread_yield: Allows the
ptrace: Traces the execution of a
pty: Pseudo terminal driver
ungetc: Pushes a character back into
pthread_cleanup_push: Pushes a routine onto the cleanup
putc, putchar, fputc, putw: Writes a
putchar, fputc, putw: Writes a
putenv: Sets an environment
putenv: Sets an environment
putlong: Places long byte
putlong: Places long byte
putshort: Places short byte
putshort: Places short byte
pututent, getutent, getutline,
putcw, putwchar, fputwc: Writes a
putcw, putwchar, fputwc: Writes a
putcw, putwchar, fputwc: Writes a
putcw, putwchar, fputwc: Writes a
putcw, putwchar, fputwc: Writes a
putcw, putwchar, fputwc: Writes a
getpwent, getpwuid, getpwnam,
setlocale: Changes or
res_mkquery: Makes
retrieves a/ res_send: Sends a
msqid_ds: Defines a message

qsort: Sorts a table in place
qsort: Sorts a table in place
qsort: Sorts a table in place
qsort: Sorts a table in place
qsort: Sorts a table in place
qsort: Sorts a table in place
qsort: Sorts a table in place
qsort: Sorts a table in place
qsort: Sorts a table in place
Permuted Index

creates) the ID for a message queue /Returns (and possibly msgget(2)
or removes an element in a message queue

Receives a message from a queue

Sends a message to a message queue

setquota: Enables or disables quotas on a file system

quotactl: Manipulates disk quotas

raise: Sends a signal to the executing program

rand, rand_r, randn: Generates pseudo-random numbers

randomword: Generates random passwords

cfgetispeed: Gets input baud rate for a terminal

cfgetospeed: Gets output baud rate for a terminal

cfsetispeed: Sets input baud rate for a terminal

cfsetospeed: Sets output baud rate for a terminal

commands on a remote host

cmd: Allows execution of expressions

re_comp, re_exec: Handles regular expressions

read, readv: Reads from a file

link: Connects to a symbolic link

getpass: Reads a password

geteuid: Gets the process' real or effective user ID

mallinfo, alloca: Allocates memory

reboot: Reboots system or halts


readable form to canonical form

encode_mach_o_hdr(3)

decode_mach_o_hdr(3)

rewinddir, closedir, opendir, readdir, telldir, seekdir, open, creat: Opens a file for reading or writing

readlink: Reads the value of a symbolic link

read, readv: Reads from a file

link: Connects to a symbolic link

getpass: Reads a password

geteuid: Gets the process' real or effective user ID

mallinfo, alloca: Allocates memory

reboot: Reboots system or halts


readable form /canonical header

decode_mach_o_hdr(3)

open(2)

read(2)

malloc(3)

reboot(2)

recv(2)

recvfrom(2)

recvmsg(2)

t_rcvudata(3)

recv(2)

recv(2)

recvfrom(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)

recv(2)
osp/1 programmer's reference

allows execution of commands on a remote mount points for nfs mount

shmid_ds: Defines a shared memory region
the ID for a shared memory domain name and internet address
in a mapped file or shared memory
multilevel/ mkmultdir: Converts a multilevel directory into a regular directory /converts rmmultdir(3)
re_comp, re_exec: handles regular expressions t_comp(3)
re_timer: establishes timeout t_timer(3)
remote host rcmd: rcmd(3)
remote host rexec: rexec(3)
remote mount points for nfs mount exports(4)
remote nfs server nfssvc(2)
remove: Removes a file remove(3)
unlink: Removes a directory entry unlink(2)
rmdir: Removes a directory file rmdir(2)
remove: Removes a file remove(3)

the cleanup/ pthread_cleanup_pop: Removes a routine from the top of pthread_cleanup_pop(3)
msm_remove: Removes a semaphore msm_remove(3)
open file flock: applies or removes an advisory lock on a flock(2)
insque, remque: inserts or removes an element in a queue insque(3)
the private known/ ldr_remove: Removes an installed module from ldr_remove(3)
element in a queue insque, remque: inserts or removes an insque(3)
file within a file system
within a file system rename:
clock: Reports CPU time used clock(3)
/rewind, ftell, fgetpos, fseekpos:

Resources send: Sends a query to a name server
name: Hostname
resolver: Resolver configuration file resolver(4)

controls maximum system resource utilization getrusage, getrusage(2)
vtimes: Gets information about resource consumption /setrlimit: getrlimit(2)
to a name server and retrieves a response res_send: Sends a query res_send(3)
setjmp, longjmp: Saves and restores the current execution.
endhostent: Ends retrieval of network host entries.
/statilabel, fstatilabel: Retrieve a file information label.
/statatslabel, fstatatslabel: Retrieve a file sensitivity label.
catalog catgets: Retrieves a message from a.
a query to a name server and retrieves a response.
resvport: Sends a privileged address.
	_t_rcvdis: Retrieves disconnect information.
pathconf, fpathconf: Retrieves file implementation.
localeconv, localeconv_r: Retrieves locale-dependent.
byte stream _getlong: Retrieves long quantities from a.
byte stream _getshort: Retrieves short quantities from a.
of a file statacl: Retrieves the access control list.
semaphore ID semget: Returns (and possibly creates) a.
the ID for a message/ msgget: Returns (and possibly creates) a.
the ID for a shared/ shmget: Returns (and possibly creates) a.
end point for communication and
	_sigreturn: Returns from signal.
loaded module ldr_inq_module: Returns information about a.
a region in a/ ldr_inq_region: Returns module information about a.
name in a/ ldr_lookup_package: Returns the address of a symbol.
loaded module ldr_xlookup_package: Returns the address of a.
module loaded in/ ldr_xentry: Returns the entry point for a.
thread pthread_self: Returns the ID of the calling.
load: Loads a module and
	returns the module ID./Loads.
a module in another process and
	process ldr_next_module: Returns the next module ID for a.
pthread_getspecific: Returns the value bound to a key.
environment variable getenv: Returns the value of an.
settimetype, getittimer: Sets or
	operand x neg: Negates and
	size/ pthread_attr_getstacksize: Returns the value of the stack.
Repositions the file/ fseek, /readdir, toldir, seekdir, setproctent: Opens and
setnetent: Opens and

on a remote host
floating-point/ floor, ceil, rint, fmod, fabs: Rounds

directory into a regular/timer

sqrt, cbrt: Computes square
chroot: Changes the effective
Computes square root and cube
floor, ceil, rint, fmod, fabs:
database

pthread_cleanup_pop: Removes a
pthread_cleanup_push: Pushes a
Calls an initialization compile and match

root and cube root functions
root directory
root functions sqrt, cbrt:
Rounds floating-point numbers to
ROUTER: Kernel packet forwarding
routine from the top of the/
routine onto the cleanup stack of/
routine pthread_once:
routines /Regular-expression
# OSF/1 Programmer's Reference

<table>
<thead>
<tr>
<th>Command authorization support routines</th>
<th>cmdauth(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>independent disk inode access routines</td>
<td>disk(3)</td>
</tr>
<tr>
<td>Xerox NS address conversion routines</td>
<td>ns_addr(3)</td>
</tr>
<tr>
<td>policy database management routines</td>
<td>spdbm(3)</td>
</tr>
<tr>
<td>/mult, mdiv, pow, gcd, invert, with a privileged address routines</td>
<td>mp(3)</td>
</tr>
<tr>
<td>current/ /Allows the scheduler to authenticate clients</td>
<td>rresvport(3)</td>
</tr>
<tr>
<td>execution/ setjmp, longjmp:</td>
<td>ruserok(3)</td>
</tr>
<tr>
<td>brk, scantabs, scandir, alphasort:</td>
<td>setjmp(3)</td>
</tr>
<tr>
<td>formatted input routines</td>
<td>scanf(3)</td>
</tr>
<tr>
<td>scandir, alphasort:</td>
<td>scandir(3)</td>
</tr>
<tr>
<td>pthread_yield: Allows the scheduler to run another thread/</td>
<td>pthread_yield(3)</td>
</tr>
<tr>
<td>nice: Changes scheduling priority</td>
<td>getpriority(2)</td>
</tr>
<tr>
<td>setpriority: Gets or sets process</td>
<td>getpriority(2)</td>
</tr>
<tr>
<td>/m_in, mout, omitout, omout, !search, !find:</td>
<td>lsearch(3)</td>
</tr>
<tr>
<td>delete, twalk: Manages binary search trees</td>
<td>bsearch(3)</td>
</tr>
<tr>
<td>bsearch: Performs a binary search</td>
<td>bsearch(3)</td>
</tr>
<tr>
<td>name and Internet/ res_init:</td>
<td>res_init(3)</td>
</tr>
<tr>
<td>name dn_find:</td>
<td>dn_find(3)</td>
</tr>
<tr>
<td>Inode: Format of the additional authcap:</td>
<td>inode(7)</td>
</tr>
<tr>
<td>security attributes added to/</td>
<td>authcap(7)</td>
</tr>
<tr>
<td>access routines disk:</td>
<td>disk(3)</td>
</tr>
<tr>
<td>management routines spdbm:</td>
<td>spdbm(3)</td>
</tr>
<tr>
<td>/mrand48, jrand48, srand48, seed48, lcong48: Generates/</td>
<td>drand48(3)</td>
</tr>
<tr>
<td>opendir, readdir, telldir,</td>
<td>opendir(3)</td>
</tr>
<tr>
<td>brk, sbrk: Changes data segment size</td>
<td>brk(2)</td>
</tr>
<tr>
<td>Locks a process' text and/or data multiplexing</td>
<td>select(2)</td>
</tr>
<tr>
<td>semctl: Performs semaphore control operations</td>
<td>semctl(2)</td>
</tr>
<tr>
<td>Creates (and possibly creates) a semaphore ID</td>
<td>semget(2)</td>
</tr>
<tr>
<td>shared/ msem_init: Initializes a semaphore in a mapped file or</td>
<td>semget(2)</td>
</tr>
<tr>
<td>semop: Performs semaphore operations</td>
<td>semop(2)</td>
</tr>
<tr>
<td>semid_ds: Defines a semaphore set</td>
<td>semid_ds(4)</td>
</tr>
<tr>
<td>msem_lock: Locks a semaphore</td>
<td>msem_lock(3)</td>
</tr>
<tr>
<td>msem_remove: Removes a semaphore</td>
<td>msem_remove(3)</td>
</tr>
<tr>
<td>msem_unlock: Unlocks a semaphore</td>
<td>msem_unlock(3)</td>
</tr>
<tr>
<td>control operations</td>
<td>semctl(2)</td>
</tr>
<tr>
<td>creates) a semaphore ID operations</td>
<td>semget(2)</td>
</tr>
<tr>
<td>shutdown: Shuts down socket send and receive operations</td>
<td>shutdown(2)</td>
</tr>
<tr>
<td>socket using a message structure send: Sends messages on a socket</td>
<td>send(2)</td>
</tr>
<tr>
<td>serial data line tcsendbreak: sendmsg: Sends a message from a</td>
<td>sendmsg(2)</td>
</tr>
<tr>
<td>Creates (and possibly creates) a semaphore ID operations</td>
<td>semid_ds(4)</td>
</tr>
<tr>
<td>semop: Performs semaphore operations</td>
<td>semop(2)</td>
</tr>
<tr>
<td>sendsmsg: Sends a message from a</td>
<td>sendsmsg(2)</td>
</tr>
<tr>
<td>Sends a break on an asynchronous</td>
<td>tcsendbreak(3)</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>send</td>
<td>Sends messages on a socket</td>
</tr>
<tr>
<td>sendto</td>
<td>Sends messages through a socket</td>
</tr>
<tr>
<td>sendudata</td>
<td>Sends a data unit</td>
</tr>
<tr>
<td>sendmsg</td>
<td>Sends a message from a socket</td>
</tr>
<tr>
<td>msgsnd</td>
<td>Sends a message to a message</td>
</tr>
<tr>
<td>res_send</td>
<td>Sends a query to a name server</td>
</tr>
<tr>
<td>queue</td>
<td>Sends a query to a name server</td>
</tr>
<tr>
<td>kill</td>
<td>Sends a signal to a process or to a group of processes</td>
</tr>
<tr>
<td>raise</td>
<td>Sends a signal to the executing process</td>
</tr>
<tr>
<td>sendto</td>
<td>Sends messages through a socket</td>
</tr>
<tr>
<td>t_snd</td>
<td>Sends normal data or expedited</td>
</tr>
<tr>
<td>t_snddis</td>
<td>Sends user-initiated disconnect</td>
</tr>
<tr>
<td>socket</td>
<td>/Gets the current process's sensitivity label and clearance</td>
</tr>
<tr>
<td>/getslabel</td>
<td>Sensitivity label functions</td>
</tr>
<tr>
<td>/chslabel</td>
<td>Changes the sensitivity label of a file</td>
</tr>
<tr>
<td>/fstatslabel</td>
<td>Sensitivity label or clearance</td>
</tr>
<tr>
<td>/shm_chslabel</td>
<td>Manipulates sensitivity labels on/</td>
</tr>
<tr>
<td>/spp</td>
<td>Xerox distributed pseudo-random number</td>
</tr>
<tr>
<td>tcsegments</td>
<td>Sends a break on an asynchronous process</td>
</tr>
<tr>
<td>res_send</td>
<td>Sends a query to a name server</td>
</tr>
<tr>
<td>nfssvc</td>
<td>Creates a remote NFS server</td>
</tr>
<tr>
<td>/async daemon</td>
<td>Creates an NFS asynchronous I/O server</td>
</tr>
<tr>
<td>/ruserok</td>
<td>Allows authentication</td>
</tr>
<tr>
<td>/local</td>
<td>Makes query messages for name</td>
</tr>
<tr>
<td>/getservbyname</td>
<td>Get service entry by name</td>
</tr>
<tr>
<td>/getservbyport</td>
<td>Get service entry by port</td>
</tr>
<tr>
<td>/setservent</td>
<td>Gets service file entry</td>
</tr>
<tr>
<td>/services</td>
<td>Service name database</td>
</tr>
<tr>
<td>/getfsspec</td>
<td>Gets file system entry</td>
</tr>
<tr>
<td>/setgroups</td>
<td>Sets the group access</td>
</tr>
<tr>
<td>/sethostent</td>
<td>Opens network host</td>
</tr>
<tr>
<td>/setilabel</td>
<td>Sets the current information label</td>
</tr>
<tr>
<td>/setgid</td>
<td>Sets the group ID</td>
</tr>
<tr>
<td>/setgrgid</td>
<td>Accesses the/</td>
</tr>
<tr>
<td>/getgrgid</td>
<td>Sets the group access</td>
</tr>
<tr>
<td>/getgrnm</td>
<td>Sets the unique</td>
</tr>
<tr>
<td>/gethostent</td>
<td>Opens network host</td>
</tr>
<tr>
<td>/gethostid</td>
<td>Sets the name of the host</td>
</tr>
<tr>
<td>/sethostname</td>
<td>Sets the name of the host</td>
</tr>
<tr>
<td>/setilabel</td>
<td>Sets the current information label</td>
</tr>
</tbody>
</table>
returns the value of interval/ returns the value of interval/
restores the current execution/ setitimer, getitimer: Sets or
setbuf, setvbuf, setbuffer, getitimer (2)
program’s entire current locale/ setitimer (2)
getlogin, getlogin_r, log syslog, openlog, closeog,
log syslog, openlog, closeog,
networks file
setlocale: Changes or queries the
process group ID
setlocale (3)
setlocale: Changes or queries the
ID setgid,
scheduling priority getpriority,
scheduling priority getpriority,
authorizations and privileges
getprlpen, getprlpmam,
the /etc/protocols file
getprpwuid, getprpmnam,
getprpwuid, getprpmnam,
quotas on a file system
getpwuid, getpwnam, putpwent,
quotas on a file system
getpwuid, getpwnam, putpwent,
effective group ID
getprlpen, getprlpmam,
user ID’s
getprlpen, getprlpmam,
process group IDs
getprlpen, getprlpmam,
scheduling priority getpriority,
scheduling priority getpriority,
authorizations and privileges
getprlpen, getprlpmam,
the /etc/protocols file
getprpwuid, getprpmnam,
process user IDs
getprpwuid, getprpmnam,
context sigstack:
getpwuid, getpwnam, putpwent,
context sigstack:
file creation mask umask:
file creation mask umask:
ulimit:
ulimit:
settimeofday, ftime: Gets and
settimeofday, ftime: Gets and
times utime, utimes:
times utime, utimes:
chpriv:
chpriv:
statpriv:
statpriv:
terminal cfsetispeed:
terminal cfsetispeed:
goto sigsetjmp:
goto sigsetjmp:
priviligties setpriv:
priviligties setpriv:
getlogin_r, setlogin: Gets and
getlogin_r, setlogin: Gets and
setuid:
setuid:
interval timers alarm, ualarm:
interval timers alarm, ualarm:
setterm, getterm:
setterm, getterm:
terminal cfsetospeed:
terminal cfsetospeed:
getpriority, setpriority: Gets or
getpriority, setpriority: Gets or
setreuid:
setreuid:
sigprocmask, sigsetmask:
sigprocmask, sigsetmask:
setgroups:
setgroups:
setgid:
setgid:
setterm, getterm:
setterm, getterm:
terminal cfsetospeed:
terminal cfsetospeed:
setpriority, setpriority: Gets or
setpriority, setpriority: Gets or
setreuid:
setreuid:
sigprocmask, sigsetmask:
sigprocmask, sigsetmask:
setgroups:
setgroups:
setgid:
setgid:
setterm, getterm:
setterm, getterm:
terminal cfsetospeed:
terminal cfsetospeed:
getpriority, setpriority: Gets or
getpriority, setpriority: Gets or
setreuid:
setreuid:
sigprocmask, sigsetmask:
sigprocmask, sigsetmask:
setgroups:
setgroups:
setgid:
setgid:
setterm, getterm:
setterm, getterm:
terminal cfsetospeed:
terminal cfsetospeed:
getpriority, setpriority: Gets or
getpriority, setpriority: Gets or
setreuid:
setreuid:
sigprocmask, sigsetmask:
sigprocmask, sigsetmask:
setgroups:
setgroups:
setgid:
setgid:
setterm, getterm:
setterm, getterm:
terminal cfsetospeed:
terminal cfsetospeed:
getpriority, setpriority: Gets or
getpriority, setpriority: Gets or
setreuid:
setreuid:
sigprocmask, sigsetmask:
sigprocmask, sigsetmask:
setgroups:
setgroups:
setgid:
setgid:
setterm, getterm:
setterm, getterm:
terminal cfsetospeed:
terminal cfsetospeed:
getpriority, setpriority: Gets or
getpriority, setpriority: Gets or
setreuid:
setreuid:
sigprocmask, sigsetmask:
sigprocmask, sigsetmask:
setgroups:
setgroups:
setgid:
setgid:
sethostname: Sets the name of the current host ........................................ sethostname(2)

with the terminal tcsetattr: Sets the parameters associated ....................... tcsetattr(3)

setpgid, setpgrp: Sets the process group ID ...................................... setpgid(2)

setpgid: Sets the process group ID ........................................ setpgid(3)

setuid, seteuid: Sets the process user IDs ....................................... setuid(3)

ID setregid: Sets the real and effective group ...................................... setregid(2)

clock stime: Sets the system-wide time-of-day ..................................... stime(3)

current host sethostid: Sets the unique identifier of the .................. sethostid(2)

setuid: Sets the user ID ........................................................ setuid(2)

pthread_attr_setstacksize: Sets the value of the stack size.................. pthread_attr_setstacksize(3)

setclock: Sets value of system-wide clock ................................. setclock(3)

and decimal ASCII character sets ascii: Octal, hexadecimal, ............................. ascii(S)

entry setservent: Gets service file ................................... setservent(3)

sets: Sets the process group ID ........................................ setsid(2)

setregid: Sets the real and effective group ..................................... setregid(2)

clock stime: Sets the system-wide time-of-day ..................................... stime(3)

current host sethostid: Sets the unique identifier of the .................. sethostid(2)

setuid: Sets the user ID ........................................................ setuid(2)

names of legal/ getusershell, /getutid, getutline, pututline, .......................... getusershell(3)

Assigns buffering to a/ setbuf, ......................................... setbuf(3)

shmctl: Performs shared memory control operations ....................... shmctl(2)

setvbuf, setlinebuf: .................................................. setvbuf(3)

smat: Attaches a shared memory region ......................................... smat(2)

shmctl: Detaches a shared memory region ......................................... shmctl(2)

shm: Defines a shared memory region ................................................. shm(2)

possibly creates) the ID for a shared memory region ......................... shmget(2)

control/ /sem_chacl, shm-stattacl, ...................................... sem_chacl(3)

information/ /shm_statlabel, .................................................. shm_statlabel(3)

sensitivity/ /shm_statlabel, .................................................. shm_statlabel(3)

sem_stattacl, sem_chacl, .................................................. sem_chacl(3)

sem_statlabel, sem_chilabel, .......................... sem_chilabel(3)

sem_statslabel, sem_chilabel, .......................... sem_chilabel(3)

region control operations.................................................. sem_chilabel(3)

region creates) the ID for a shared/ .................................................. shmget(2)

region hton: Converts an unsigned ................................................. htonl(3)

ntohs: Converts an unsigned .................................................. ntoh(3)

byte stream putshort: Places ................................................. _getshort(3)

stream _getshort: Retrieves .................................................. _getshort(3)

and receive operations shutdown: Shuts down socket send ....................... shutdown(2)

receive operations shutdown: Shuts down socket send and ....................... shutdown(2)

a compatibility interface to the sigaction( ) function /Provides ............. sigvec(2)
action to take upon delivery of sigaction, signal: Specifies the action to take upon delivery of a signal.
sigemptyset, sigfillset, sigaddset, sigdelset:
compatibility interface to the sigemptyset, sigfillset, sigaddset, sigdelset, sigismember:
Compatibility interfaces/ sigset, sigset, sighold, sigrelse, siginterrupt:
interrupt functions sigpending:
actions taken upon delivery of sigaction, signal: Specifies the
and sigaction(2) sigemptyset(3)
sigblock: Provides a sigemptyset(2)
sigdelset, sigismember: Creates sigemptyset(3)
sigemptyset, sigfillset, sigaddset, sigdelset,
sigismember: / sigemptyset, Compatibility interfaces/ sigset,
sigset, sighold, sigrelse, siglongjmp:
nonlocal goto with sigemptyset(2)
signal handling
compatibility interface to the / sigfillset, sigaddset, sigdelset, sigismember:
and sigemptyset, sigfillset, sigaddset, sigdelset, sigismember:
Compatibility interfaces/ sigset,
sigset, sighold, sigrelse, sigignore:
definitions and variables used by sigset, sighold, sigrelse, sigignore:
signal handling
definitions and variables used by signal functions /Contains sigemptyset(3)
sigemptyset, sigfillset, sigaddset, sigdelset, sigismember:
Compatibility interfaces/ sigset,
sigset, sighold, sigrelse, sigignore:
signal handling
signal is received pause(3)
signal /changes the set of sigsuspendsignal(2)
signal/signal: Specifies the signal(2)
signal.h: Contains definitions signal(4)
signal: Specifies the action to sigaction(2)
signals and waits for a signal siginterrupt(3)
signal /changes the set of sigsuspend(2)
signal to a process or to a group kill(2)
signal to end the current process abort(3)
signal to the executing program raise(3)
signal: /Changes the set of sigsuspend(2)
signal mask sigprocmask, sigprocmask(2)
signal masks /sigismember: sigemptyset(3)
signal stack context sigreturn(2)
siginterrupt: Allows siginterrupt(3)
sigpending: Examines pending sigpending(2)
signals sigpending(2)
signal functions /Contains signal(4)
signal to a process or to a group kill(2)
signal to end the current process abort(3)
signal to the executing program raise(3)
signal mask sigprocmask, sigprocmask(2)
signal /changes the set of sigsuspend(2)
signal /signal: Specifies the signal(2)
signal.h: Contains definitions signal(4)
signal: Specifies the action to sigaction(2)
signals and waits for a signal sigsuspend(2)
signals to interrupt functions siginterrupt(3)
signals
sigpause: Provides a sigpause(3)
sigpending: Examines pending sigpending(2)
signal mask sigprocmask, sigprocmask(2)
sigprocmask function /Provides sigblock(2)
sigsetmask: Sets the current sigprocmask(2)
sigprocmask, sigsetmask: Sets the sigprocmask(2)
sigrelse, sigignore: sigset(3)
sigreturn: Returns from signal sigreturn(2)
sigset, sighold, sigrelse, sigignore: sigset(3)
sigsetjmp: Sets jump point for a sigsetjmp(3)
sigsetmask: Sets the current sigprocmask(2)
sigstack: Sets and gets signal sigstack(2)
sigreturn function /Provides sigpause(3)
sigreturn: Returns from signal sigreturn(2)
sigset, sighold, sigrelse, sigignore: sigset(3)
sigsetjmp: Sets jump point for a sigsetjmp(3)
sigsetmask: Sets the current sigprocmask(2)
sigstack: Sets and gets signal sigstack(2)
sigreturn function /Provides sigpause(3)
sigreturn: Returns from signal sigreturn(2)
sigset, sighold, sigrelse, sigignore: sigset(3)
sigsetjmp: Sets jump point for a sigsetjmp(3)
sigsetmask: Sets the current sigprocmask(2)
sigstack: Sets and gets signal sigstack(2)
sigreturn function /Provides sigpause(3)
Permuted Index

name dn_skipname: Skips over a compressed domain dn_skipname(3)
interval
current user ttyslot: Finds the ttyslot(3)
backlog of listen: Listens for listen(2)
getsockname: Gets the socket name getsockname(2)
getsockopt: Gets socket options getsockopt(2)
sendmsg: Sends a message from a socket sendmsg(2)
sendto: Sends messages through a socket sendto(2)
Accepts a new connection on a socket accept(2)
connect: Connects two sockets connect(2)
recvfrom: Receives messages from sockets recvfrom(2)
recv: Receives messages from connected sockets recv(2)
listensocket: Creates an end point for socket(2)
socketpair: Creates a pair of connected sockets socketpair(2)
bind: Binds a name to a socket bind(2)
send: Sends messages on a socket send(2)
sendto: Sends messages through a socket sendto(2)
setsockopt: Sets socket options setsockopt(2)
shutdown: Shuts down socket send and receive shutdown(2)
listen: Listens for socket connections and limits the listen(2)
socket: Creates an end point for socket(2)
accept: Accepts a new connection on a socket accept(2)
socketgetpeername: Gets the name of the peer socket getpeername(2)
recvmsg: Receives messages from a socket recvmsg(2)
socketsocketpair: Creates a pair of connected sockets socketpair(2)
socketpair: Creates a pair of connected sockets socketpair(2)
lo: Software loopback network lo(7)
nsip: Software network interface nsip(7)
abort: Generates a software signal to end the abort(3)
qsort: Sorts a table in place qsort(3)
sortedirectorycontents: Gets space configuration/scandir(3)
modulesin that process' address space ldr_xattach(3)
management routines spdbm: Security policy database spdbm(3)
mknod: Creates an FIFO or special file mknod(2)
stopio: Stop further I/O to a special file stopio(3)
/a character string to the specified integer data type atoi(3)
catclose: Closes a file catclose(3)
catopen: Opens a file catopen(3)
address of a symbol name within a module ldr_xlookup_package(3)
on a condition variable for a delivery of a sigaction, signal: pthread_cond_timedwait(3)
specifies the action to take upon sigaction(2)
system volume fs, inode: fs(4)
special file: spdbm: Security policy database spdbm(3)
special file: mknod(2)
stopio: Stop further I/O to a special file stopio(3)
atsymbol name/various externals specified integer data type atoi(3)
specific message catalog catclose(3)
specific message catalog catopen(3)
specific package in another/the ldr_xlookup_package(3)
specified period of time / Waits pthread_cond_timedwait(3)
spdp: Xerox sequenced packet spp(7)
sprintf: Prints formatted output printf(3)
sqrt, cbrt: Computes square root sqrt(3)
sqrt, cbrt: Computes square root sqrt(3)
square root and cube root sqrt(3)
numbers rand, rand_r srand48(3), seed48, lcong48, rand(): Generates pseudo-random rand(3)
scanf: Converts formatted input scanf(3)
stab: Symbol table types stab(4)
signals stack context sigstack(2)
stack of the calling thread and: pthread_cleanup_pop(3)
OSF/1 Programmer’s Reference

is received pause: Suspending a process until a signal pause(3)
interval sleep: Suspending execution for an interval sleep(3)
usleep: Swaps bytes swab(3)
swab device for interleaved paging and swapping swab(3)
swab: Swaps bytes swab(3)
/Returns the address of a symbol name in a package ldr_lookup_package(3)
stab: Symbol table types stab(4)
symlink: Makes a symbolic link to a file symlink(3)
readlink: Reads the value of a symbolic link readlink(2)
package/ /Returns the address of a symbol name within a specified package ldr_xlookup_package(3)
sync: Updates all file systems sync(2)
clock /Corrects the time to allow synchronization of the system adjtime(2)
msync: Synchronizes a mapped file msync(2)
t_sync: Synchronizes transport library t_sync(3)
select: Synchronous I/O multiplexing select(2)
variables syslog: Gets configurable system syslog(3)
setlogmask: Controls the system log /openlog, closelog, syslog(3)
/setlogmask: Controls the system to allow synchronization of the system adjtime(2)
putprdfnam: Manipulate system default database entry getprdfent(3)
setlogmask: Controls the system log /openlog, closelog, syslog(3)
mmap: Maps file system object into virtual memory mmap(2)
paging/ madvise: Advise the system of a process’ expected behavior madvise(2)
reboot: Reboots the system or halts processor reboot(2)
getpagesize: Gets the system page size getpagesize(2)
/setrlimit: Controls maximum resource consumption getrlimit(2)
staffs, fstaffs, ustaff: Gets file system statistics staffs(2)
sysconf: Gets configurable system variables sysconf(3)
Specifies the format of the file system volume fs, inode: fs(4)
mount: Mounts a file system mount(3)
unmount: Unmounts a file system umount(3)
an existing file on current file system /directory entry for link(2)
Get the name of the current directory or a file within a file system rename: Renames a file rename(2)
or disables quotas on a file system setquota: Enables setquota(2)
setquota: Enables and disables quotas on a file system setquota(2)
gets the name of the current directory or a file within a file getname(2)
getclock: Gets current value of the system-wide clock getclock(3)
system-wide time-of-day clock stime(3)
setclock: Sets value of system-wide time-of-day clock setclock(3)
system-wide clock system-wide clock setclock(3)
settime: Sets the system-wide time-of-day clock stime(3)
system: Executes a shell command system(3)
system: Programs protocol family ns(7)
hier: Layout of file systems hier(5)
sync: Updates all file systems sync(2)
Gets list of all mounted file systems getfsstat(2)
request t_accept: Accepts a connect t_accept(3)
structure
transport endpoint
connection with another/
state of the transport provider
options for a transport endpoint
expedited data on a connection
confirmation from a connect/
error indication
expedited data over a connection
disconnect request
connect orderly release
library
qsort:
from the private known package
en:
#include:
Functions
output data or nonread input/
associated with the terminal
control protocol
asynchronous serial data line
associated with the terminal
process group ID
search trees
/ putptcnam:
pty:
termios file, which provides the
tty: General terminal interface .................................tty(7)

Gets input baud rate for a terminal cfgetispeed: ........................................... cfgetispeed(3)

Gets output baud rate for a terminal cfgetispeed: ........................................... cfgetispeed(3)

Sets input baud rate for a terminal cfsetispeed: ........................................... cfsetispeed(3)

Sets output baud rate for a terminal cfsetispeed: ........................................... cfsetispeed(3)

the pathname for the controlling
parameters associated with the terminal tcgetattr: Gets the ......................... tcgetattr(3)

parameters associated with the terminal tcgetattr: Sets the ............................ tcgetattr(3)

isatty: Gets the name of a terminal ttcname, (3)

terminfo: Describes terminfo: Describes terminals by .................................... terminfo(3)

for a child process to stop or terminate /waitpid, wait3: Waits ......................... wait(2)

Waits for a thread to terminate pthread_join: ............................................. pthread_join(3)

exit, atexit, _exit: Terminates a process exit(2)

pthread_exit: Terminates the calling thread pthread_exit(3)

pthread_cancel: Initiates termination of a thread pthread_cancel(3)

cancellation point of the termios file, which provides the/termios.h : Defines the structure termios(4)

of the termios file, which/termios(4)

feof: Tests EOF on a stream feof(3)

isnan: Tests for NaN (Not a Number) isnan(3)

stream ferror: Tests the error indicator on a ferror(3)

memory plock: Locks a process' text and/or data segments in plock(2)

binary search trees tsearch, thereof /or queries the program's setlocale(3)

element current locale or portions thread and optionally executes it pthread_cleanup_pop(3)

of the cleanup stack of the calling thread attributes object pthread_attr_create(3)

 pthread_attr_create: Creates a thread attributes object pthread_attr_create(3)

 pthread_attr_delete: Deletes a thread attributes object pthread_attr_delete(3)

pthread_attr_getstacksize: thread attributes object /value pthread_attr_getstacksize(3)

of the stack size attribute of a pthread_attr_setstacksize: thread attributes object /value pthread_attr_setstacksize(3)

pthread_equal: Compares two thread identifiers pthread_equal(3)

due to the scheduler to run another thread instead of the current one pthread_yield(3)

pthread_cond_signal: Wakes up a thread that is waiting on a/ pthread_cond_signal(3)

pthread_join: Waits for a thread to terminate pthread_join(3)

pthread_create: Creates a thread pthread_create(3)

pthread_detach: Detaches a thread pthread_detach(3)

sigwait: Suspends a calling thread sigwait(3)

cancellation point in the calling thread /Creates a thread that is waiting on a/ pthread_testcancel(3)

cancelability of the calling thread /disables the asynchronous pthread_setasynccancel(3)

cancelability of the calling thread /disables the general pthread_setcancel(3)

the cleanup stack of the calling thread /Pushes a routine onto pthread_cleanup_push(3)

Initiates termination of a thread pthread_cancel: ......................................... pthread_cancel(3)

Terminates the calling thread pthread_exit: ............................................. pthread_exit(3)

pthread_self: pthread_self(3)

time-of-day clock stime(3)

time: Gets time time(3)

time: Gets time time(3)

time: Gets time time(3)

timeout intervals of a/timeout of interval timers alarm(3)

/ualarm: Sets or changes the

tod: Check tod(3)

tod: Check tod(3)

tod: Check tod(3)

tod: Check tod(3)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mktimer</td>
<td>Allocates a per-process timer</td>
</tr>
<tr>
<td>rmtimer</td>
<td>Frees a per-process timer</td>
</tr>
<tr>
<td>intervals of a per-process</td>
<td>or returns the value of interval</td>
</tr>
<tr>
<td>gets process and child process</td>
<td>changes the timeout of interval</td>
</tr>
<tr>
<td>Sets file access and modification process times</td>
<td>name for a temporary file</td>
</tr>
<tr>
<td>tmpfile</td>
<td>Creates a temporary file</td>
</tr>
<tr>
<td>tmpnam, tempnam</td>
<td>Constructs the</td>
</tr>
<tr>
<td>toascii, tolower, _tolower</td>
<td>Translates characters</td>
</tr>
<tr>
<td>conv</td>
<td>Translates OSF/ROSE: Object</td>
</tr>
<tr>
<td>tcp</td>
<td>Internet</td>
</tr>
<tr>
<td>t_bind</td>
<td>Binds an address to a transport endpoint</td>
</tr>
<tr>
<td>t_close</td>
<td>Closes a transport endpoint</td>
</tr>
<tr>
<td>t_open</td>
<td>Establishes a transport endpoint</td>
</tr>
<tr>
<td>t_unbind</td>
<td>Disables a transport endpoint</td>
</tr>
<tr>
<td>t_look</td>
<td>Looks at the current event on a transport endpoint</td>
</tr>
<tr>
<td>t_optmgmt</td>
<td>Manages protocol options for a transport endpoint</td>
</tr>
<tr>
<td>t_sync</td>
<td>Synchronizes transport library</td>
</tr>
<tr>
<td>t_getstate</td>
<td>Gets the current state of the transport provider</td>
</tr>
<tr>
<td>t_connect</td>
<td>Establishes a connection with another transport user</td>
</tr>
<tr>
<td>ftw</td>
<td>Walks a file</td>
</tr>
<tr>
<td>tsearch, tfind, tdelete</td>
<td>Manages binary search</td>
</tr>
<tr>
<td>pthread_mutex_trylock</td>
<td>Tries once to lock a mutex</td>
</tr>
<tr>
<td>acos, atan, atan2</td>
<td>Computes the trigonometric and inverse/ /asin, trigonometric and inverse</td>
</tr>
<tr>
<td>/the trigonometric and inverse</td>
<td>length</td>
</tr>
<tr>
<td>tsearch, tfind, tdelete, twalk</td>
<td>Manages binary search</td>
</tr>
<tr>
<td>tty</td>
<td>General terminal interface</td>
</tr>
<tr>
<td>ttyname, isatty</td>
<td>Gets the name of</td>
</tr>
<tr>
<td>ttyslot</td>
<td>Finds the slot in the</td>
</tr>
<tr>
<td>tty</td>
<td>Manages binary search</td>
</tr>
<tr>
<td>type /Converts a character string</td>
<td>trigonometric functions.</td>
</tr>
<tr>
<td>stub: Symbol table</td>
<td>Manages binary search</td>
</tr>
<tr>
<td>/localtime, localtime_r, rkt ime, timeout of interval/ alarm, protocol (UDP)</td>
<td>a terminal</td>
</tr>
<tr>
<td>localtime</td>
<td>Changes the time units</td>
</tr>
<tr>
<td>ct ime</td>
<td>Converts time units</td>
</tr>
<tr>
<td>alarm</td>
<td>Sets or changes the</td>
</tr>
<tr>
<td>alarm</td>
<td>Internet user datagram</td>
</tr>
</tbody>
</table>
ulimit: Sets and gets user limits .................................. ulimit(3)
unmask: Sets and gets the value of ............................. umask(2)
unmount: Mounts or unmounts a file .......................... umount(2)
unmount: Unmounts a file system ............................... umount(3)
uname: Gets the name of the ................................. uname(2)
ungetc, ungetwc: Pushes a character back ungetc(3)
ungetc: Pushes a character back ............................... ungetc(3)
ungetc: Pushes a character back ungetc(3)
unique file name.................................................. mktemp(2)
unique identifier of the current user gethostid(2)
unique identifier of the current user gethostid(2)
unit data error indication........................................ t_rcvuderr(3)
unit................................................................. t_rcvuderr(3)
unit................................................................. t_rcvuderr(3)
unit /localtime, localtime_r, ctime(3)
unlink: Removes a directory entry ............................ unlink(2)
unload: Unloads a previously loaded module previously unload(3)
Unloads a previously loaded module previously unload(3)
unlocked_getc, unlocked_getchar: ................................... unlocked_getc(3)
unlocked_getc: Gets a character from an unlocked_getc(3)
stream...................................................... unlocked_getc(3)
unlocked_putc, unlocked_putchar: .................................. unlocked_putc(3)
unlocked_putc: Writes a character to a stream unlocked_putc(3)
Unlocks a semaphore ........................................... msem_unlock(3)
Unlocks a semaphore ........................................... msem_unlock(3)
Unlocks a dataset .............................................. funlockfile(3)
Unlocks a dataset .............................................. funlockfile(3)
Unmaps a mapped region ...................................... munmap(2)
Unmaps a file system .......................................... umount(3)
Unmounts a file system ........................................ umount(3)
unsigned long (32-bit) integer ............................... htonl(3)
unsigned long (32-bit) integer ............................... htonl(3)
unsigned short (16-bit) integer .............................. htons(3)
unsigned short (16-bit) integer .............................. htons(3)
pause: Suspends a process........................................ pause(3)
Perform a linear search and update ........................ lsearch(3)
Perform a linear search and update ........................ lsearch(3)
Update all file systems........................................ sync(2)
Update all file systems........................................ sync(2)
sync: / Specifies the action to take upon delivery of a signal sigaction(2)
sync: / Specifies the action to take upon delivery of a signal sigaction(2)
user database / Accesses the user database / Accesses the getgrent(3)
user database / Accesses the user database / Accesses the getgrent(3)
user datagram protocol (UDP) .................................. udp(7)
User environment .............................................. environ(5)
User environment .............................................. environ(5)
user ID ......................................................... getuid(3)
user ID ......................................................... getuid(3)
user ID ......................................................... getuid(3)
user ID ......................................................... getuid(3)
user ID ......................................................... getuid(3)
user ID getsuid, geteuid: Gets ............................... geteuid(3)
user ID getuid, geteuid: Gets ............................... geteuid(3)
user ID's ..................................................... seteuid(2)
user ID's ..................................................... seteuid(2)
user IDs setreuid, ............................................ setreuid(2)
user IDs setreuid, ............................................ setreuid(2)
user information in the user/ getpwent(3)
user limits .................................................... ulimit(3)
user or group IDs .......................................... identity(3)
user or group IDs .......................................... identity(3)
user shells /setusershell, ..................................... getusershell(3)
user shells /setusershell, ..................................... getusershell(3)
connection with another transport in the utmp file for the current user ttyslot: Finds the slot ttyslot(3)

user t_connect: Establishes a t_connect(3)
user-initiated disconnect request t_snddis(3)

_gr_idtoname: Map between user and group names and IDs pw_mapping(3)
cuserid: Gets the alphanumeric username associated with the/ cuserid(3)

Gets disk description using a disk name getdiskbyname: getdiskbyname(3)
/Sends a message from a socket using a message structure sendmsg(2)

interval usleep: Suspends execution for an usleep(3)

statistics labs, fstatfs, getrusage, vtimes: getrusage(2)
and modification times utime, utimes: utime(2)

endutent, utmpname: Accesses utmp file entries /setutent, getutent(3)
ttyslot: Finds the slot in the utmp file for the current user ttyslot(3)

/pututline, setutent, endutent, utmpname: Accesses utmp file/ getutent(3)
mvalid: Checks memory region for validity mvalid(2)
/labs, ldv: Computes absolute value and division of integers abs(3)

pthread_getspecific: Returns the value bound to a key pthread_getspecific(3)

and floating-point absolute value functions /Modulo Remainder floor(3)
readlink: Reads the value of a symbolic link readlink(2)
/getenv: Returns the value of an environment variable getenv(3)

/getitimer: Sets or returns the value of interval timers getitimer(2)
setclock: Sets value of system-wide clock setclock(3)

neg: Negates and returns the value of the double operand x neg(3)

umask: Sets and gets the value of the file creation mask umask(2)
of a thread/ /Returns the value of the stack size attribute pthread_attr_getstacksize(3)
of a thread attributes/ /Sets the value of the stack size attribute pthread_attr_setstacksize(3)

/Binds a thread-specific a double-precision floating-point function and complex absolute value /Euclidean distance hypot(3)
value /a character string to atof(3)

/vfprintf, vsprintf: Formats a variable-length parameter/ varargs parameter list for output vprintf(3)
/Creates a condition variable attributes object pthread_condattr_create(3)
/Deletes a condition variable attributes object pthread_condattr_delete(3)
of time /Waits on a condition variable for a specified period pthread_cond_timedwait(3)
putenv: Sets an environment variable putenv(3)

that is waiting on a condition variable /Wakes up a thread pthread_cond_signal(3)
that are waiting on a condition variable /Wakes up all threads pthread_cond_broadcast(3)
the value of an environment variable getenv: Returns getenv(3)
Destroys a condition variable pthread_cond_destroy: pthread_cond_destroy(3)
Creates a condition variable pthread_cond_init: pthread_cond_init(3)
 waits on a condition variable pthread_cond_wait: pthread_cond_wait(3)

varargs: Handles a variable-length parameter list varargs(3)
/Contains definitions and variables used by signal/ signal(4)
sysconf: Gets configurable system variables sysconf(3)
flag letters from the argument vector getopt: Gets getopt(3)
fork, vfork: Creates a new process fork(2)
/varargs parameter list/ vprintf, vsprintf: Formats a vprintf(3)
Maps file system object into interface lvm: Logical Volume Manager (LVM) programming lvm(7)
the format of the file system
volume fs, inode: Specifies ................................................. fs(4)
Formats a varargs parameter list/ vprintf, vfprintf, vsprintf: ........................................ vprintf(3)
resource utilization getusage, child process to stop or/ to stop or/ wait, waitpid, wait wait3: Waits for a process to stop or/ wait, waitpid, wait3: Waits for a child process, wait3: Waits for a child process waiting on a condition variable waiting on a condition variable on a/ wait, wait, waitpid, wait3: Waits for a child process to stop or/ wait, wait, waitpid, wait3: Waits for a child process to stop or/ wait, wait, waitpid, wait3: Waits for a signal /changes
the set of blocked signals and pthread_join: Waits for a thread to terminate pthread_join(3)
pthread_cond_timedwait(3) waits for a varargs parameter
pthread_cond_timedwait: Waits on a condition variable for
pthread_cond_timedwait(3)
Wakes up a thread that is waiting
Wakes up all threads that are
Wakes up all threads that are
ftw: Walks a file tree ftw(3)
character string into a/ into a multibyte character character wctomb: Converts a wide character to a multibyte character wctomb(3)
/character string to a character wctomb: Converts a wide character to a multibyte character wctomb(3)
wstok: Performs operations on wide character strings to a multibyte character wstok(3)
Controls cursor movement and windowing curses Library: ................................... curses(3)
renames a directory or a file within a specified package in/ ldr_xlookup_package(3)
/getw: Gets a character or word from an input stream getc(3)
/gewchar: Gets a character or putw: Writes a character or a character string to a character wctomb: Converts a wide character to a multibyte character wctomb(3)
unlocked_putc, unlocked_putchar: Writes a character or a fputwc: Writes a character or a function error perror: Writes a message explaining a function error perror(3)
permanent storage fsync: Writes changes in a file to fsync(2)
write, writew: Writes to a file write(2)
write: Writes to a file write(2)
Opens a file for reading or writing open, creat: ............................................... open(2)
stream putwc, putwchar, fputwc: Writes a character or a word to a fputwc(3)
unlocked_putc, unlocked_putchar: Writes a character to a stream unlocked_putc(3)
function error perror: Writes a message explaining a function error perror(3)
 permanent storage fsync: Writes changes in a file to fsync(2)
write, writew: Writes to a file write(2)
writing open, creat: .................................................. open(2)
wsprintf: Prints formatted output wsscanf(3)
wscanf: Converts formatted input wscanf(3)
This chapter contains reference pages for OSF/1 functions. The reference pages from the `man2` and `man3` directories are sorted alphabetically in this chapter.

1.1 Organization of the Reference Pages

The manual pages for functions in this volume use the following format:

- **Purpose**: This section describes the general purpose of the programming interface.
- **Synopsis**: This section describes the appropriate syntax for using the programming interface, including any headers, and the types of all arguments.
**Description**  This section describes the behavior of the interface, including the conditions or permissions required for its successful use, the domain of legal values for all arguments, and the interface's effects on the state of processes or files.

**Return Value**  
This section specifies the return values for successful or unsuccessful completion of the invoked function.

**Errors**  
This section describes the error conditions under which the invoked function will or may fail to complete successfully, and the value of `errno` associated with each.

**Related Information**  
This section provides cross-references to related interfaces and headers described within this document.

### 1.2 Error Numbers

This section summarizes and describes the error codes ("errnos") returned by functions. Some error codes represent more than one type of error. For example, [E2BIG] can indicate that the specified argument size has exceeded the system limit of ARG_MAX, or that the specified number of sembuf structures has exceeded a predefined limit.

The error codes are listed in alphabetical order in Table 1-1.
# Table 1-1. OSF/1 Erros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[E2BIG]</td>
<td>Indicates that the specified argument list exceeds the system limit of ARG_MAX bytes, or the number of bytes in the message exceeds the predefined limit.</td>
<td>7</td>
</tr>
<tr>
<td>[EACCESS]</td>
<td>Indicates that the requested operation did not have the proper access permissions. This error may also indicate one or more of the following: that the named file is not an ordinary file (acct()); the operation would cause the parent directory or process' information level to float such that it would no longer be dominated by the directory or process' sensitivity level; the requested file is not available for read or write access; the process is attempting to mount on a multilevel child directory; the value of the process ID argument matches the process ID of a child process of the calling process and the child process has successfully executed one of the exec functions (setpgid()); the function is trying to manipulate two files on two different file systems (setquota()).</td>
<td>13</td>
</tr>
<tr>
<td>[EADDRINUSE]</td>
<td>Indicates that the specified address is already in use.</td>
<td>48</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>[EADDRNOTAVAIL]</td>
<td>Indicates that the specified address is not available from the local machine.</td>
<td>49</td>
</tr>
<tr>
<td>[EAFNOSUPPORT]</td>
<td>Indicates that the addresses in the specified address family are not supported by the protocol family.</td>
<td>47</td>
</tr>
<tr>
<td>[EAGAIN]</td>
<td>Indicates that the requested resource, such as a lock or a process, is temporarily unavailable. This error may also indicate one or both of the following: if the O_NONBLOCK flag is set for the requested function, the process would be delayed in a read or write operation, or that the specified time has elapsed (pthread_cond_timedwait()).</td>
<td>35</td>
</tr>
<tr>
<td>[EBADF]</td>
<td>Indicates that a socket or file descriptor parameter is invalid.</td>
<td>9</td>
</tr>
<tr>
<td>[EBUSY]</td>
<td>Indicates that the requested element is currently unavailable, or the associated system limit was exceeded.</td>
<td>16</td>
</tr>
<tr>
<td>[ECHILD]</td>
<td>Indicates either that the child process does not exist, or that the requested child process information is unavailable.</td>
<td>10</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>[ECONNABORTED]</td>
<td>Indicates that the software caused a connection abort because there is no space on the socket's queue and the socket cannot receive further connections.</td>
<td>53</td>
</tr>
<tr>
<td>[ECONNREFUSED]</td>
<td>Indicates that the connection request was refused.</td>
<td>61</td>
</tr>
<tr>
<td>[EDEADLK]</td>
<td>Indicates either a probable deadlock condition, or that the requested lock is owned by someone else.</td>
<td>11</td>
</tr>
<tr>
<td>[EDOM]</td>
<td>Indicates that x and/or y are either Not a Number (NaN), or that they are in some other way unacceptable (for example, exceed system limits).</td>
<td>33</td>
</tr>
<tr>
<td>[EDQUOT]</td>
<td>Indicates that the file system of the requested directory has exceeded the user's quota of disk blocks.</td>
<td>69</td>
</tr>
<tr>
<td>[EEXIST]</td>
<td>Indicates that the request element (for example, file, semaphore, etc.) already exists.</td>
<td>17</td>
</tr>
<tr>
<td>[EFAULT]</td>
<td>Indicates that the requested address is in some way invalid, for example, out of bounds.</td>
<td>14</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>[EFBIG]</td>
<td>Indicates either that the file size exceeds the process’ file size limit, or that the requested semaphore number is invalid (0 or greater than or equal to the specified number of semaphores).</td>
<td>27</td>
</tr>
<tr>
<td>[EIDRM]</td>
<td>Indicates that the requested semaphore or message queue ID has been removed from the system.</td>
<td>81</td>
</tr>
<tr>
<td>[EINTR]</td>
<td>Indicates that an interruptible function's process was interrupted by a signal, which it caught.</td>
<td>4</td>
</tr>
<tr>
<td>[EINVAL]</td>
<td>Indicates that an invalid argument was passed to the function (such as, the requested argument does not exist or is out of bounds or is not a regular file, or that the result would be invalid). This error may also indicate one or more of the following: the requested socket is not accepting connections (accept()) or is already bound (bind()); the specified super block had a bad magic number or out of range block size (mount()); the requested parameter is a lock/unlock parameter, but the element to be locked is already locked/unlocked (plock()); the kernel has not been compiled with the QUOTA option (quota()); an attempt was made to to ignore or supply a handler for the SIGKILL, SIGSTOP, and SIGCONT signals</td>
<td>22</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>[EIO]</td>
<td>Indicates a read or write physical I/O error. These errors do not always occur with the associated function, but can occur with the subsequent function. This error may also indicate that the requested parameter does not have an appropriate value, or is invalid.</td>
<td>5</td>
</tr>
<tr>
<td>[EISCONN]</td>
<td>Indicates that the socket is already connected.</td>
<td>56</td>
</tr>
<tr>
<td>[EISDIR]</td>
<td>Indicates either that the request was for a write access to a file but the specified filename was actually a directory, or that the function was trying to rename a directory as a file.</td>
<td>21</td>
</tr>
<tr>
<td>[ELOOP]</td>
<td>Indicates that too many links were encountered in translating a pathname.</td>
<td>62</td>
</tr>
<tr>
<td>[EMFILE]</td>
<td>Indicates one or more of the following errors: too many file descriptors are open (exceeding OPEN_MAX); no space remains in the mount table; the attempt to attach a shared memory region.</td>
<td>24</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>[EMLINK]</td>
<td>exceeded the maximum number of attached regions allowed for any one process.</td>
<td>31</td>
</tr>
<tr>
<td>[EMSGSIZE]</td>
<td>Indicates that the number of links would exceed LINK_MAX.</td>
<td>40</td>
</tr>
<tr>
<td>[ENAMETOOLONG]</td>
<td>Indicates that the message is too large to be sent all at once, as the socket requires.</td>
<td>63</td>
</tr>
<tr>
<td>[ENETUNREACH]</td>
<td>Indicates that no route to the network or host exists.</td>
<td>51</td>
</tr>
<tr>
<td>[ENFILE]</td>
<td>Indicates either that the system file table is full, or that there are too many files currently open in the system.</td>
<td>23</td>
</tr>
<tr>
<td>[ENOBUF]</td>
<td>Indicates insufficient resources, such as buffers, to complete the call.</td>
<td>55</td>
</tr>
<tr>
<td>[ENODEV]</td>
<td>Indicates one or more of the following errors: the file descriptor refers to an object that cannot be mapped; the requested block special device file does not exist; a file system is unmounted.</td>
<td>19</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>ENOENT</td>
<td>Indicates one or more of the following errors: the specified file pathname or directory pathname does not exist or points to an empty string; the O_CREAT flag is set and the named file or path prefix does not exist (<code>open()</code>); a message queue identifier does not exist for a message key identifier and the IPC_CREAT flag is not set for the function (<code>msgget()</code>); a semaphore ID does not exist for a semaphore key identifier and the IPC_CREAT flag is not set for the function (<code>semget()</code>); a shared memory region ID does not exist for a shared memory region key identifier and the IPC_CREAT flag is set for the function (<code>shmeget()</code>).</td>
<td>2</td>
</tr>
<tr>
<td>ENOEXEC</td>
<td>Indicates that the specified file has appropriate access permissions but has an improper format, such as an unrecognizable object file format.</td>
<td>8</td>
</tr>
<tr>
<td>ENOLCK</td>
<td>Indicates that lock table is full because too many regions are already locked, or satisfying a lock/unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.</td>
<td>77</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>[ENOMEM]</td>
<td>Indicates that insufficient memory is available for the requested function. This error may indicate one or more of the following errors: mapped region attribute was set and part of the specified address range is already allocated (mmap()); the specified range is invalid for a process' address space or the range specifies one or more unmapped pages (msync()); a new semaphore could not be created (msem_init()).</td>
<td>12</td>
</tr>
<tr>
<td>[ENOMSG]</td>
<td>Indicates that a message of the requested type does not exist and the IPC_NOWAIT flag is set.</td>
<td>80</td>
</tr>
<tr>
<td>[ENOPKG]</td>
<td>Indicates that the specified package was not found.</td>
<td>92</td>
</tr>
<tr>
<td>[ENOPROTOOPT]</td>
<td>Indicates that the requested socket option is unknown and the protocol is unavailable.</td>
<td>42</td>
</tr>
<tr>
<td>[ENOSPC]</td>
<td>Indicates one or more of the following errors: not enough space to extend the file system or device for file and/or directory writes; the madvise() function tried to reserve resources that were not available to be reserved; the system-imposed limit of the maximum number of allowed message queue identifiers has been exceeded (msgget()); an attempt to create a semaphore ID exceeded the system-wide limit on</td>
<td>28</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>[ENOSYM]</td>
<td>Indicates that the specified package does not contain the named symbol.</td>
<td>93</td>
</tr>
<tr>
<td>[ENOTBLK]</td>
<td>Indicates that the specified parameter is not or does not point to a block device.</td>
<td>15</td>
</tr>
<tr>
<td>[ENOTCONN]</td>
<td>Indicates that the socket is not connected.</td>
<td>57</td>
</tr>
<tr>
<td>[ENOTDIR]</td>
<td>Indicates that a component of the path parameter is not a directory, or an operation is being performed from a directory to a nonexistent directory.</td>
<td>20</td>
</tr>
<tr>
<td>[ENOTSOCK]</td>
<td>Indicates that the specified socket parameter refers to a file, not a socket.</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>the semaphore table (<strong>semget</strong>()); an attempt to create a new shared memory region ID exceeded the system-wide limit of maximum IDs (<strong>shmget</strong>()); the system-defined limit on the number of processes using SEM_UNDO was exceeded (<strong>semop</strong>()).</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>[ENOTTY]</td>
<td>Indicates one or more of the following errors: the file descriptor’s file is not a terminal; the calling process does not have a controlling terminal; the controlling terminal is no longer associated with the calling process session (tcsetpgrp()); the specified open descriptor is not associated with a character special device or the specified request does not apply to the kind of object that the specified open descriptor references (ioctl()).</td>
<td>25</td>
</tr>
<tr>
<td>[ENXIO]</td>
<td>Indicates one or more of the following errors: the specified address, major device number, or channel is out of valid range; no more channels are available (open()); the named file is a character or block special file and the associated device does not exist (open()); the O_NONBLOCK flag is set, the named file is FIFO, O_WRONLY is set, and no process has the file open for reading (open()).</td>
<td>6</td>
</tr>
<tr>
<td>[EOPNOTSUPP]</td>
<td>Indicates either that the socket does not support the requested operation, or that the socket cannot accept the connection.</td>
<td>45</td>
</tr>
<tr>
<td>[EPERM]</td>
<td>Indicates that the function attempted to perform an operation for which it did not have appropriate privileges (such as the privileges</td>
<td>1</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>EPIPE</td>
<td>allowed by the security options), or the caller was not the owner of the requested element or superuser. This error may also indicate one or both of the following: the calling process was not in the same session as the target process (<code>setpgid()</code>) or the calling process is already the process group leader or the process group ID of a process other than the calling process matches the process ID of the calling process (<code>setsid()</code>).</td>
<td></td>
</tr>
<tr>
<td>[EPIPE]</td>
<td>Indicates that an attempt was made to write to a pipe or FIFO that was not open for reading by any process.</td>
<td>32</td>
</tr>
<tr>
<td>EPROTONOSUPPORT</td>
<td>Indicates that either the socket or the protocol is not supported.</td>
<td>43</td>
</tr>
<tr>
<td>ERANGE</td>
<td>Indicates one or more of the following errors: the result would exceed the system-defined limits or cause an overflow (value too large) or an underflow (value too small); a specified parameter is greater than 0 (zero) but smaller than the length of the pathname + 1 (<code>getcwd()</code>); the symbol value cannot be represented as an absolute value; the magnitude of x is such that total or partial loss of significance resulted.</td>
<td>34</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>[EROFSl</td>
<td>Indicates one or more of the following errors: the operation requested was to be performed on a read-only file system; an attempt was made to activate a paging file on a read-only file system; the named file resides on a read-only file system and the file type requires write access.</td>
<td>30</td>
</tr>
<tr>
<td>[ESPIPE]</td>
<td>Indicates that an invalid seek operation was requested for a pipe (FIFO), socket, or multiplexed special file.</td>
<td>29</td>
</tr>
<tr>
<td>[ESRCH]</td>
<td>Indicates one or more of the following errors: the requested process or child process ID is invalid or not in use; no disk quota is found for the specified user; the specified thread ID does not refer to an existing thread.</td>
<td>3</td>
</tr>
<tr>
<td>[ESTALE]</td>
<td>Indicates that the specified process' root or current directory is located in a virtual file system that has been unmounted (stale NFS file handle).</td>
<td>70</td>
</tr>
<tr>
<td>[ETIMEDOUT]</td>
<td>Indicates that the requested attempt at a connection timed out before a connection was established.</td>
<td>60</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>[ETXTBSY]</td>
<td>Indicates either that the requested file is currently opened for writing by another process, or that a write access is requested by a pure procedure (shared text) file that is being executed.</td>
<td>26</td>
</tr>
<tr>
<td>[EUSERS]</td>
<td>Indicates that there are too many users, as evidenced by a full quota table.</td>
<td>68</td>
</tr>
<tr>
<td>[EWOULDBLOCK]</td>
<td>Indicates one or more of the following errors: the socket is marked nonblocking and no connections are waiting to be accepted; the socket is marked nonblocking and connection cannot be immediately completed; the file is locked and the function is instructed not to block when locking; the socket is marked as nonblocking and no space is available for the specified function.</td>
<td>35</td>
</tr>
<tr>
<td>[EXDEV]</td>
<td>Indicates either that a hard link was attempted between two file systems, or that a filename to be renamed by rename() is on a different file system from the link to which it is to be renamed.</td>
<td>18</td>
</tr>
</tbody>
</table>
abort

Purpose  Generates a software signal to end the current process

Library  Standard C Library (libc.a)

Synopsis  
#include <stdlib.h>
int abort ( void );

Description

The abort() function sends a SIGABRT signal to the current process. This terminates the process and produces a memory dump, unless the signal is caught and the signal handler does not return.

If the SIGABRT signal is neither caught nor ignored, and if the current directory is writable, the system produces a memory dump in the core file in the current directory and prints an error message.

If the call to the abort() function terminates the process, the abort() will have the effect of the fclose() function on every open stream. The abort() function then terminates the process with the same result as the _exit() function, except that the status made available to the wait() or waitpid() function by abort() will be that of a process terminated by the SIGABRT signal. If the call to abort() terminates the process, all open message catalog descriptors will also be closed.

Notes

The abort() function is supported for multi-threaded applications.

AES Support Level:  Full use

Related Information

Functions:  exit(2), kill(2), sigaction(2)
abs, div, labs, ldiv

**Purpose**
Computes absolute value and division of integers

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <stdlib.h>

int abs (int i);
long labs (long i);
struct div_t div (int numerator, int denominator);
struct ldiv_t ldiv (long numerator, long denominator);
```

**Parameters**
- **i**
  - For `abs()`, specifies some integer.
  - For `labs()`, specifies some long integer.
- **numerator**
  - For `div()`, specifies some integer.
  - For `ldiv()`, specifies some long integer.
- **denominator**
  - For `div()`, specifies some integer.
  - For `ldiv()`, specifies some long integer.

**Description**
The `abs()` function returns the absolute value of its integer operand.

The `div()` function computes the quotient and remainder of the division of the numerator `numerator` by the denominator `denominator`. If the division is inexact, the sign of the resulting quotient is that of the algebraic quotient, and the magnitude of the resulting quotient is the largest integer less than the magnitude of
the algebraic quotient. If the result cannot be represented (for example, if the denominator is 0), the behavior is undefined. The `div()` function returns a structure of type `div_t`, comprising both the quotient and the remainder.

The `labs()` and `ldiv()` functions perform the same functions as `abs()` and `div()` respectively, but accept long integers rather than integers as parameters. The `ldiv()` function returns a structure of type `ldiv_t`, comprising both the quotient and the remainder.

Notes

The `abs()`, `labs()`, `div()`, and `ldiv()` functions are supported for multi-threaded applications.

A two's-complement integer can hold a negative number whose absolute value is too large for the integer to hold. When given this largest negative value, the `abs()` function returns the same value.

AES Support Level: Full use

Return Values

The `abs()` function and `labs()` function return the absolute value of their arguments.

The `div()` function returns a structure of type `div_t` and the `ldiv()` function returns a structure of type `ldiv_t`.

Related Information

Functions: `floor(3)`
accept

Purpose  Accepts a new connection on a socket

Synopsis  
```
#include <sys/types.h>
#include <sys/socket.h>

int accept (  
    int socket,
    struct sockaddr *address,
    int *address_len );
```

Parameters

- **socket**
  Specifies a socket that was created with the `socket()` function, has been bound to an address with the `bind()` function, and has issued a successful call to the `listen()` function.

- **address**
  Points to a `sockaddr` structure, the format of which is determined by the domain and by the behavior requested for the socket. The `sockaddr` structure is an overlay for a `sockaddr_in`, `sockaddr_un`, or `sockaddr_ns` structure, depending on which of the supported address families is active. If the compile-time option `_SOCKADDR_LEN` is defined before the `sys/socket.h` header file is included, the `sockaddr` structure takes 4.4BSD behavior, with a field for specifying the length of the socket address. Otherwise, the default 4.3BSD `sockaddr` structure is used, with the length of the socket address assumed to be 14 bytes or less.

  If `_SOCKADDR_LEN` is defined, the 4.3BSD `sockaddr` structure is defined with the name `osockaddr`.

- **address_len**
  Specifies the length of the `sockaddr` structure pointed to by the `address` parameter.

Description

The `accept()` function extracts the first connection on the queue of pending connections, creates a new socket with the same properties as the specified socket, and allocates a new file descriptor for that socket.

If the `listen()` queue is empty of connection requests, the `accept()` function blocks a calling socket of the blocking type until a connection is present, or returns an [EWOULDBLOCK] for sockets marked nonblocking.
accept(2)

The accepted socket cannot itself accept more connections. The original socket remains open and can accept more connections.

**Return Values**

Upon successful completion, the `accept()` function returns the nonnegative socket descriptor of the accepted socket, places the address of the peer in the `sockaddr` structure pointed to by the `address` parameter, and sets the `address_len` parameter to the length of `address`. If the `accept()` function fails, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `accept()` function fails, `errno` may be set to one of the following values:

- `[EINVAL]` The socket is not accepting connections.
- `[EBADF]` The `socket` parameter is not valid.
- `[ENOTSOCK]` The `socket` parameter refers to a file, not a socket.
- `[EOPNOTSUPP]` The referenced socket can not accept connections.
- `[EFAULT]` The `address` parameter is not in a writable part of the user address space.
- `[EWOULDBLOCK]` The socket is marked nonblocking, and no connections are present to be accepted.
- `[EMFILE]` There are too many open file descriptors.

**Related Information**

Functions: `bind(2), connect(2), listen(2), socket(2)`
access

Purpose  Determines the accessibility of a file

Synopsis  #include <unistd.h>

int access (const char *path,
            int access_mode);  

Parameters
        path        Points to the file pathname. When the path parameter refers to a symbolic link, the access() function returns information about the file pointed to by the symbolic link.

Permission to access all components of the path parameter is determined by using a real user ID instead of an effective user ID, a group access list (including a real group ID) instead of an effective group ID.

access_mode  Specifies the type of access. The bit pattern contained in the access_mode parameter is constructed by a logical OR of the following values:

R_OK  Checks read permission.
W_OK  Checks write permission.
X_OK  Checks execute (search) permission.
F_OK  Checks to see if the file exists.

Description  The access() function checks for accessibility of the file specified by a pathname.

Only access bits are checked. A directory may be indicated as writable by access(), but an attempt to open it for writing will fail (although files may be created there); a file's access bits may indicate that it is executable, but the execve() function can fail if the file does not contain the proper format.

Notes

AES Support Level: Full use
Return Values

Upon successful completion, the `access()` function returns value of 0 (zero). Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `access()` function fails, access to the file specified by the `path` parameter is denied and `errno` may be set to one of the following values:

- **ENOTDIR** A component of the path prefix is not a directory.
- **EINVAL** The pathname contains a character with the high-order bit set.
- **ENAMETOOLONG** A component of a pathname exceeded PATH_MAX characters, or an entire pathname exceeded NAME_MAX characters.
- **ENOENT** The named file does not exist or is an empty string.
- **EACCES** Permission bits of the file mode do not permit the requested access, or search permission is denied on a component of the path prefix. The owner of a file has permission checked with respect to the “owner” read, write, and execute mode bits, members of the file’s group other than the owner have permission checked with respect to the “group” mode bits, and all others have permissions checked with respect to the “other” mode bits.
- **ELOOP** Too many symbolic links were encountered in translating the pathname.
- **EROFS** Write access is requested for a file on a read-only file system.
- **ETXTBSY** Write access is requested for a pure procedure (shared text) file that is being executed.
- **EFAULT** The `path` parameter points outside the process’ allocated address space.
- **EIO** An I/O error occurred while reading from or writing to the file system.

Related Information

Functions: `chmod(2), stat(2)`
acct

Purpose
Enables and disables process accounting

Synopsis
int acct (char *path);

Parameters
path
Specifies a pointer to the pathname of the file, or specifies a null pointer.

Description
The acct() function enables and disables UNIX process accounting. When enabled, process accounting produces an accounting record on behalf of each terminating process. The path parameter specifies the pathname of the file to which an accounting record is written. When the path parameter is 0 (zero) or a null value, the acct() function disables the accounting routine.

If the path parameter refers to a symbolic link, the acct() function writes records to the file pointed to by the symbolic link.

If Network File System is installed on your system, the accounting file can reside on another node. To ensure accurate accounting, each node must have its own accounting file, which can be located on any node in the network.

The calling process must have superuser privilege to enable or disable process accounting.

Return Values
Upon successful completion, the acct() function returns a value of 0 (zero). Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors
If the acct() function fails, errno may be set to one of the following values:

[EPERM] The calling process does not have appropriate system privilege.
[ENOENT] The file named by the path parameter does not exist.
The file named by the *path* parameter is not an ordinary file.
Write permission is denied for the named accounting file.
The named file resides on a read-only file system.

**Related Information**

Functions: `exit(2)`, `sigaction(2)`, `sigvec(2)`, `expacct(3)`, `raise(3)`
adjtime

Purpose
Corrects the time to allow synchronization of the system clock

Synopsis
#include <sys/time.h>

int adjtime (  
    struct timeval *delta,
    struct timeval *old_delta);

Parameters

delta Points to the amount of time to be altered.

old_delta Points to the number of nanoseconds still to be corrected from an earlier call.

Description

The adjtime() function makes small adjustments to the system time (as returned by the gettimer() function), advancing or decreasing it by the time specified by the delta parameter of the timeval structure. If delta is negative, the clock is slowed down by incrementing it more slowly than normal until the correction is complete. If delta is positive, a larger increment than normal is used until the correction is complete.

The skew used to perform the correction is generally a fraction of one percent. Thus, the time is always a monotonically increasing function.

A time correction from an earlier call to adjtime() may not be finished when adjtime() is called again. In this case, the delta remaining from the original call is replaced by the delta of the current call. If the old_delta parameter is nonzero, then when the adjtime() function returns, the structure pointed to will contain the time remaining from the earlier call.

This call may be used by time servers that synchronize the clocks of computers in a local area network. Such time servers would slow down the clocks of some machines and speed up the clocks of others to bring them to the average network time.

The adjtime() function is restricted to users with superuser privilege.
Notes

In BSD, system time is defined in units of seconds and microseconds, while in POSIX real time extensions, the units are seconds and nanoseconds. However, the adjtime() function is not specified by POSIX. Therefore, the existing BSD interface is preserved.

Return Values

Upon successful completion, the adjtime() function returns a 0 (zero). If the adjtime() function fails, a value of -1 is returned, and errno is set to indicate the error.

Errors

If the adjtime() function fails, errno may be set to one of the following values:

- [EFAULT] An argument address referenced invalid memory.
- [EPERM] The process’s effective user ID does not have appropriate system privilege.

Related Information

Functions: gettimeofday(2), gettimer(3)
alarm, ualarm

Purpose
Sets or changes the timeout of interval timers

Library
Standard C Library (libc.a)

Synopsis
#include <sys/unistd.h>

unsigned int alarm(
    unsigned int seconds);

unsigned int ualarm(
    unsigned int milliseconds,
    unsigned int interval);

Parameters
seconds      Specifies a number of real-time seconds.
milliseconds Specifies a number of real-time microseconds.
interval     Specifies the interval for repeating the timer.

Description
The **alarm**() function is used to obtain notification of a timeout after the number of real-time seconds specified by the `seconds` parameter has elapsed. At some time after `seconds` seconds have elapsed, a signal is delivered to the process. Each call resets the timer until the `seconds` parameter is set to 0 (zero). When the notification signal is caught or ignored, no action takes place; otherwise the calling process is terminated. The **alarm**() function uses the ITIMER_REAL interval timer.

The **ualarm**() function is used to obtain notification of a timeout after the number of real-time microseconds specified by the `milliseconds` parameter has elapsed. When the `interval` parameter is nonzero, timeout notification occurs after the number of microseconds specified by the `interval` parameter has been added to the `milliseconds` parameter. When the notification signal is caught or ignored, no action takes place; otherwise the calling process is terminated. The **ualarm**() function is the simplified interface to the **setitimer**() function, and uses the ITIMER_REAL interval timer.

Notes
The **alarm**() function is supported for multi-threaded applications. The **ualarm**() function is not supported for multiple threads.
Although the `alarm()` function itself is reentrant, it should be noted that just as the second of two calls from a single thread to `alarm()` resets the timer, this is also true if two calls are made from different threads.

**AES Support Level:** Full use

**Return Values**

Upon successful completion, the value 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `alarm()` function fails, `errno` may be set to the following value:

- `[EINVAL]` The `seconds` parameter specifies a negative value or a value greater than 100,000,000.

**Related Information**

Functions: `gettimer(3)`
asinh, acosh, atanh

Purpose Computes inverse hyperbolic functions

Library Math Library (libm.a)

Synopsis #include <math.h>

double asinh (double x);
double atanh (double x);
double acosh (double x);

Parameters

x Specifies some double value.

Description
The asinh() function returns the hyperbolic arc sine of x, in the range -infinity to +infinity. The acosh() function returns the hyperbolic arc cosine of x, in the range 1 to +infinity. The atanh() function returns the hyperbolic arc tangent of x, in the range -infinity to +infinity.

Return Values
Upon successful completion, the asinh(), acosh(), and atanh() functions return the hyperbolic arc sine, hyperbolic arc cosine, and hyperbolic arc tangent of x. Otherwise, acosh() function returns a NaNQ if x < 1, and the atanh() function returns a NaNQ if x > 1.

Related Information
Functions: exp(3), sinh(3)
assert

Purpose
Inserts program diagnostics

Library
Standard C Library (libc.a)

Synopsis
#include <assert.h>

void assert(
    int expression);

Parameters
expression  Specifies an expression that is evaluated as TRUE or FALSE. This expression is evaluated in the same manner as a C language if control statement.

Description
The assert() macro inserts diagnostics into programs. On execution, when the expression parameter is false (returns FALSE), this macro writes information about the particular call that failed, including the text of the argument, the name of the source file, and the source-file line number (the latter two are respectively the values of preprocessing macros __FILE__ and __LINE__) on stderr. The error message is taken from the standard C library message catalog. Also, the abort() function produces a software abort fault.

When you compile a program with the -DNDEBUG preprocessor option, or with the #define NDEBUG preprocessor control statement before the #include <assert.h> statement, calls to the assert() macro have no effect.

Notes

AES Support Level:  Full use
Return Values
The assert() function returns no value.

Related Information
Functions: abort(3)
async_daemon

Purpose
Creates a local NFS asynchronous I/O server

Synopsis
async_daemon( void );

Description
The async_daemon() function starts an NFS compatible asynchronous I/O server. Normally this function does not return unless the server is terminated by a signal.

Return Values
Upon successful completion, 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.

Errors
If the async_daemon() function fails, errno may be set to the following value:
[EBUSY] The system limit on asynchronous daemons has been exceeded.

Related Information
Functions: nfssvc(2)
atof, strtod

Purpose  Converts a character string to a double-precision floating-point value

Library

Standard C Library (libc.a)

Synopsis

#include <stdlib.h>

double atof(
    const char *nptr);

double strtod(
    const char *nptr,
    char **endptr);

Parameters

nptr  Points to the character string to convert.

endptr  Specifies either a null value, or a pointer to the character that ended the scan or to a null value.

Description

The _atof_() function converts the string pointed to by the _nptr_ parameter up to the first character that is inconsistent with the format of a floating-point number to a _double_ floating-point value. Leading white-space characters are ignored. A call to this function is equivalent to a call to _strtod(nptr, (char **) NULL)_ except for error handling. When the value cannot be represented, the result is undefined.

The _strtod_() function converts the initial portion of the string pointed to by the _nptr_ parameter to _double_ representation. First the input string is decomposed into the following three parts:

- An initial, possibly empty, sequence of white-space characters (as specified by the _isspace_() function).
- A subject sequence interpreted as a floating-point constant.
- A final string of one or more unrecognized characters, including the terminating null character of the input string.

After decomposition of the string, the subject sequence is converted to a floating-point number, and the resulting value is returned. A subject sequence is defined as
the longest initial subsequence of the input string, starting with the first nonwhite-space character, that is of the expected form. The expected form and order of the subject sequence is:

- An optional plus (+) or minus (-) sign.
- A sequence of digits optionally containing a radix character.
- An optional exponent part. An exponent part consists of e or E, followed by an optional sign, which is followed by one or more decimal digits.

When the input string is empty or consists entirely of white space, or when the first nonwhite-space character is other than a sign, a digit, or a radix character, the subject sequence contains no characters.

For the `strtod()` function, when the value of the `endptr` parameter is not (char**) NULL, a pointer to the character that terminated the scan is stored at *endptr.

When a floating-point value cannot be formed, *endptr is set to nptr.

Notes

The `setlocale()` function may affect the radix character used in the conversion result.

**AES Support Level:** Full use

Return Values

When the string is empty or begins with an unrecognized character, +0.0 is returned as the floating-point value.

When a correct return value overflows, a properly signed HUGE_VAL (INF) is returned. On underflow, a properly signed 0 (zero) is returned.

Upon successful completion, either function returns the converted floating-point value.

Errors

If the `atof()` or `strtod()` function fails, `errno` may be set to the following value:

[ERANGE] The input string is out of range (that is, the subject sequence can not be converted to a floating-point value without causing underflow or overflow).

Related Information

Functions: `atoi(3), scanf(3)`
atol, strtol, strtoul

**Purpose**

Converts a character string to the specified integer data type

**Library**

Standard C Library (*libc.a*)

**Synopsis**

```c
#include <stdlib.h>

int atoi(
    const char *nptr);

long atol(
    const char *nptr)

long strtol(
    const char *nptr,
    char **endptr,
    int base);

unsigned long int strtoul(
    const char *nptr,
    char **endptr,
    int base);
```

**Parameters**

- `nptr` Points to the character string to convert.
- `endptr` Points to a character string that ends the scan or to a null pointer.
- `base` Specifies the radix to use for the conversion.

**Description**

The `atoi()`, `atol()`, `strtol()`, and `strtoul()` functions are used to convert a character string pointed to by the `nptr` parameter to an integer having a specified data type.

The `atoi()` function converts the character string pointed to by the `nptr` parameter up to the first character inconsistent with the format of a decimal integer to an integer data type. Leading white-space characters are ignored. A call to this function is equivalent to a call to `strtol(nptr, (char**) NULL, 10)`. The `int` value of the input string is returned.

The `atol()` function converts the character string pointed to by the `nptr` parameter up to the first character inconsistent with the format of a decimal integer to a long
integer data type. Leading white-space characters are ignored. A call to this function is equivalent to a call to `strtol(nptr, (char**) NULL, 10)`. The `long int` value of the input string is returned.

The `strtol()` function converts the character string pointed to by the `nptr` parameter up to the first character inconsistent with the format of a decimal integer to a long integer data type. Leading white-space characters are ignored. First the input string is decomposed into the following three parts:

- An initial, possibly empty, sequence of white-space characters (as specified by the `isspace()` function).
- A subject sequence interpreted as an integer represented in some radix determined by the value of the `base` parameter.
- A final string of one or more unrecognized characters, including the terminating null character of the input string.

After decomposition of the string, the subject sequence is converted to a long integer and the resulting value is returned. A subject sequence is defined as the longest initial subsequence of the input string, starting with the first nonwhite-space character that is of the expected form. The expected form and order of the subject sequence depends on the value of the `base` parameter:

- When the value of the `base` parameter is 0 (zero), the expected form of the subject sequence is that of an integer-constant optionally preceded by a + (plus sign) or - (minus sign), but not including an integer suffix. The sequence of characters starting with the first digit is interpreted as an integer constant.
- When the value of the `base` parameter is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with its radix specified by the value of the `base` parameter, optionally preceded by a + (plus sign) or - (minus sign) and not including an integer suffix.

Alphabetic characters from "a" or "A" through "z" or "Z" are assigned decimal values 10 through 35, respectively. Only alphabetic characters with assigned values less than that of the `base` parameter are converted. For example, when the value of the `base` parameter is 20, only the following value assignments are converted:

<table>
<thead>
<tr>
<th>Char</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Val</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>...</td>
</tr>
</tbody>
</table>
• When the value of the base is 16 (hexadecimal), the expected form of the subject sequence is a string of alphanumeric characters optionally preceded by characters "0x" or "0X", which must follow an optional initial sign character, when present.

• When the expected form of the subject sequence is preceded by a - (minus sign), the converted integer value has a negative value.

When the input string is empty or does not have the expected form, conversion does not take place. The value of the nptr parameter is stored in the object pointed to by the endptr parameter, whenever this parameter is not a null pointer.

The strtol() function is the same as the strtol() function, except that it does not accept a leading sign character and it returns an unsigned long integer.

Return Values

For the strtol() and strtoul() functions, when the value of the endptr parameter is not (char **) NULL, a pointer to the character that terminated the scan is stored in the location pointed to by the nptr parameter.

For the strtol() and strtoul() functions, when an integer result cannot be formed, the *nptr parameter is set to the value of endptr, and 0 (zero) is returned.

For the strtol() and strtoul() functions, when the base parameter is positive but not greater than 36, the value of base is used as the conversion radix. After an optional leading sign, leading zeros are ignored. Whenever the base parameter is 16, "0x" or "0X" is ignored.

For the strtol() and strtoul() functions, when the base parameter is 0 (zero), the string pointed to by the nptr parameter determines the radix. Thus, after an optional leading sign, a leading 0 (zero) indicates octal conversion, and a leading "0x" or "0X" indicates hexadecimal conversion. The default conversion is to decimal values.

Upon successful completion, these functions return the proper data type and value of the converted integer.
Errors

If any of these functions fail, `errno` may be set to one of the following values:

- **[ERANGE]** The input string is out of range (that is, the subject sequence can not be converted to the proper data type and value without causing overflow).
- **[EINVAL]** The radix value specified for the `base` parameter is not supported.

Related Information

Functions: `atof(3)`, `scanf(3)`
bcopy, bcmp, bzero, ffs

Purpose
Performs bit and byte string operations

Library
Standard C Library (libc.a)

Synopsis
void bcopy (char *source, char *destination, int length);
int bcmp (char *string1, char *string2, int length);
void bzero (char *string, int length);
int ffs (int index);

Parameters
source Points to the original string for the bcopy() function.
destination Points to the destination string for the bcopy() function.
string1 Specifies the byte string to be compared to the string2 parameter by the bcmp() function.
string2 Specifies the byte string to be compared to the string1 parameter by the bcmp() function.
length Specifies the length (in bytes) of the string.
index Specifies the bit whose index should be returned.

Description
The bcopy(), bcmp(), and bzero() functions operate on variable length strings of bytes. Unlike the string functions, they do not check for null bytes.
The `bcopy()` function copies the value of the `length` parameter in bytes from the string in the `source` parameter to the string in the `destination` parameter.

The `bcmp()` function compares the byte string in the `string1` parameter against the byte string of the `string2` parameter, returning a 0 (zero) value if the two strings are identical and a nonzero value otherwise.

The `bzero()` function nulls the string in the `string` parameter, for the value of the `length` parameter in bytes.

The `ffs()` function finds the first bit set passed to it in the `index` parameter and returns the index of that bit. Bits are numbered starting at 1. A return value of 0 (zero) indicates that the value passed is 0.

**Notes**

The `bcopy()` function takes parameters backwards from the `strcpy()` function.

**Related Information**

Functions: `memccpy(3), string(3), swab(3)`
j0, j1, jn, y0, y1, yn

Purpose
Computes Bessel functions

Library
Math Library (libm.a)

Synopsis
#include <math.h>

double j0 (double x);
double y0 (double x);
double j1 (double x);
double y1 (double x);
double jn (int n, double x);
double yn (int n, double x);

Parameters
x Specifies a double value. The value of x must be positive for the y0(), y1(), and yn() functions.
n Specifies some integer value.

Description
The j0(), j1(), jn(), y0(), y1(), and yn() functions are Bessel functions that are used to compute wave variables, primarily in the field of communications.
Notes

AES Support Level: Trial use

Return Values

The j0() and j1() functions return Bessel functions of x of the first kind, of orders 0 (zero) and 1, respectively. The jn() function returns the Bessel function of x of the first kind of order n.

If the x argument is too large in magnitude, the value 0 (zero) is returned. If x is NaN, NaN is returned. Otherwise, either errno is set to indicate the error or NaN is returned.

The y0() and y1() functions return the Bessel functions of x of the second kind, of orders 0 (zero) and 1, respectively. The yn() function returns the Bessel function of x of the second kind of order n.

If the x argument to the functions y0(), y1() or yn() is nonpositive, - HUGE_ VAL or NaN is returned. Otherwise, NaN is returned and errno is set to indicate the error.

Errors

If the j0(), j1(), or jn() function fails, errno may be set to one of the following values:

[EDOM] The value of x is NaN.

If the y0(), y1() or yn() function fails, errno may be set to one of the following values:

[EDOM] The value of x is nonpositive or NaN.

[ERANGE] The value of x was too large in magnitude.
bind

**Purpose**
Binds a name to a socket

**Synopsis**
```c
#include <sys/types.h>
#include <sys/socket.h>
int bind (int socket, struct sockaddr *address, int address_len);
```

**Parameters**
- *socket*
  Specifies the socket descriptor of the socket to be bound.
- *address*
  Points to a `sockaddr` structure, the format of which is determined by the domain and by the behavior requested for the socket. The `sockaddr` structure is an overlay for a `sockaddr_in`, `sockaddr_un`, or `sockaddr_ns` structure, depending on which of the supported address families is active. If the compile-time option `_SOCKADDR_LEN` is defined before the `sys/socket.h` header file is included, the `sockaddr` structure takes 4.4BSD behavior, with a field for specifying the length of the socket address. Otherwise, the default 4.3BSD `sockaddr` structure is used, with the length of the socket address assumed to be 14 bytes or less.

  If `_SOCKADDR_LEN` is defined, the 4.3BSD `sockaddr` structure is defined with the name `osockaddr`.

- *address_len*
  Specifies the length of the `sockaddr` structure pointed to by the `address` parameter.

**Description**
The `bind()` function assigns an `address` to an unnamed socket. Sockets created with the `socket()` function are unnamed; they are identified only by their address family.
An application program can retrieve the assigned socket name with the `getsockname()` function.

**Return Values**

Upon successful completion, the `bind()` function returns a value of 0 (zero). If the `bind()` function fails, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `bind()` function fails, `errno` may be set to one of the following values:

- **[EBADF]** The `socket` parameter is not valid.
- **[ENOTSOCK]** The `socket` parameter refers to a file, not a socket.
- **[EADDRNOTAVAIL]** The specified address is not available from the local machine.
- **[EADDRINUSE]** The specified address is already in use.
- **[EINVAL]** The socket is already bound to an address.
- **[EACCES]** The requested address is protected and the current user does not have permission to access it.
- **[EFAULT]** The `address` parameter is not in a readable part of the user address space.

**Related Information**

Functions: `connect(2), listen(2), socket(2), getsockname(2)`
brk, sbrk

Purpose
Changes data segment size

Library
Standard C Library (libc.a)

Synopsis
int brk(
    char *addr);

int sbrk(
    int incr);

Parameters

addr Points to the effective address of the maximum available data.

incr Specifies the number of bytes to be added to the current break. The value of incr may be positive or negative.

Description
The brk() function sets the lowest data segment location not used by the program (called the break) to addr, rounded up to the next multiple of the system's page size.

In the alternate function sbrk(), incr more bytes are added to the program's data space, and a pointer to the start of the new area is returned.

When a program begins execution with the execve() function, the break is set at the highest location defined by the program and data storage areas. Therefore, only programs with growing data areas should need to use sbrk().

The current value of the program break is reliably returned by "sbrk(0)". The getrlimit() function may be used to determine the maximum permissible size of the data segment. It is not possible to set the break beyond the value returned from a call to the getrlimit() function.

If the data segment was locked at the time of the brk() function, additional memory allocated to the data segment by brk() will also be locked.

Notes
Programmers should be aware that the concept of a current break is a historical remnant of earlier UNIX systems. Many existing UNIX programs were designed using this memory model, and these programs typically use the brk() or sbrk()
functions to increase or decrease their available memory. OSF/1 provides a more flexible memory model and allows the use of discontiguous memory areas (see, for example, the `mmap()` function). Therefore, references to areas above the break may be legitimate memory references which will not produce memory violations.

**Return Values**
Upon successful completion, the `brk()` function returns a value of 0 (zero), and the `sbrk` function returns the old break value. If either call fails, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**
If the `brk()` or `sbrk()` function fails, no additional memory is allocated and `errno` may be set to the following value:

[ENOMEM] The requested change would allocate more space than allowed by the limit as returned by the `getrlimit()` function.

If the `brk()` function cannot allocate the requested memory, the following message is printed:

```plaintext
cmd: could not sbrk, return = n
```

Where `cmd` is the name of the command currently executing, and `n` is the internal kernel error code returned from the memory allocation routine, `vm_allocate()`. Note that this may occur if the requested break value would cause the data segment to collide with previously allocated memory (for example, memory obtained via the `mmap()` or `vm_allocate()` call). See the *OSF/1 System Programmer's Reference Volume 1* for more information on `vm_allocate()`.

**Related Information**
Functions: `exec(2)`, `getrlimit(2)`, `malloc(3)`, `plock(2)`, `mmap(2)`
bsearch

**Purpose**
Performs a binary search

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <stdlib.h>
void *bsearch(
    const void *key,
    const void *base,
    size_t nmemb,
    size_t size,
    int (*compar)()
) (const void *, const void *);
```

**Parameters**
- `key` Points to an object that compares equal to the desired element.
- `base` Points to the initial object in the array.
- `nmemb` Specifies the number of elements in the array.
- `size` Specifies the byte size of each element of the array.
- `compar` Points to the comparison function, which is called with two parameters that point to the `key` object and to an array member, in that order.

**Description**
The `bsearch()` function does a binary search and returns a pointer in an array that indicates where an object is found. The array must have been previously sorted in increasing order according to a provided comparison function, `compar`.

The `compar` comparison function is called with two parameters that point to objects that are compared during the sort. This function returns an integer less than, equal to, or greater than 0 (zero) depending whether the object pointed to by the first `const void *` parameter is to be considered less than, equal to, or greater than the second `const void *` parameter.
Notes

**AES Support Level:** Full use

**Return Values**

Upon successful completion, the `bsearch()` function returns a pointer to a matching object in the array. A null pointer is returned when no match is found. When two or more objects compare equally, the returned object is unspecified.

**Related Information**

Functions: `hsearch(3)`, `lsearch(3)`, `qsort(3)`
catclose

**Purpose**  Closes a specified message catalog

**Library**  Standard C Library (libc.a)

**Synopsis**  
```c
#include <nl_types.h>

int catclose(
    nl_catd cat_descriptor);
```

**Parameters**

*cat_descriptor*  Specifies an index into the message catalog that is returned from a call to the `catopen()` function.

**Description**

The `catclose()` function closes a message catalog specified by the `cat_descriptor` parameter. If a file descriptor is used to implement the type `nl_catd`, that file descriptor will be closed.

If a program accesses several message catalogs, the NL_MAXOPEN number of open catalogs can be reached. In this event, some message catalogs must be closed before more can be opened.

Before exiting, programs should close any catalogs they have opened.

**Notes**

AES Support Level:  Trial use

**Return Values**

Upon successful completion, 0 (zero) is returned. Otherwise, -1 is returned. The `catclose()` function fails if the `cat_descriptor` parameter value is not valid.
Related Information

Functions: catopen(3), catgets(3)
Commands: dspcat(1), dspmsg(1), gencat(1), mkcatdefs(1)
Functions

catgets(3)

catgets

Purpose
Retrieves a message from a catalog

Library
Standard C Library (libc.a)

Synopsis
#include <nl_types>

char *catgets()

   nl_catd cat_descriptor,
   int set_number,
   int message_number,
   char *string);

Parameters

cat_descriptor
   Specifies a catalog description that is returned by the catopen() function.

set_number
   Specifies the set ID.

message_number
   Specifies the message ID. The set_number and message_number parameters specify a particular message in the catalog to retrieve.

string
   Specifies the character string buffer.

Description
The catgets() function retrieves a message from a catalog after a successful call to the catopen() function. If the catgets() function finds the specified message, it loads that message into a character string buffer, terminates the message string with a null character, and returns a pointer to the buffer. The message in the buffer is overwritten by the next call to the catgets() function.

Notes
AES Support Level: Trial use
Return Values
Upon successful completion, the catgets() function returns a pointer to an internal buffer area containing the null terminated message string. Otherwise, string is returned.

Errors
If the cat_descriptor parameter is not a valid catalog descriptor, the catgets() function returns a pointer to the user-supplied default message string specified by the string parameter. If the catgets() function cannot find the specified message in the catalog, it returns a pointer to a null string.

Related Information
Functions: catopen(3), catclose(3)
Commands: dspcat(1), dspmsg(1), gencat(1), mkcatdefs(1)
catopen

**Purpose**  
Opens a specified message catalog

**Library**  
Standard C Library (libc.a)

**Synopsis**  
```c
#include <limits.h>
#include <nl_types.h>

nl_catd catopen (  
    const char *name,  
    int oflag );
```

**Parameters**  
- `name`  
  Specifies the catalog file to open.
- `oflag`  
  Included for compatibility with X/Open. The `oflag` parameter is reserved for future use and should be set to zero.

**Description**  

The `catopen()` function opens a specified message catalog and returns a catalog descriptor that is used to retrieve messages from the catalog.

The special `nl_catd` data type is used for catalog descriptors. Since this data type is defined in the `nl_types.h` header file, include this file in your application program.

The `name` parameter specifies the name of the message catalog to be opened. If `name` contains a `/` (slash), then `name` specifies a full pathname for the message catalog. Otherwise, the environment variable `NLSPATH` is used with `name` substituted for `%N`. If `NLSPATH` does not exist in the environment, or if a message catalog cannot be opened in any of the components specified by `NLSPATH`, then an default message catalog is used. The variable `%L` will be replaced by the value of the `LANG` environment variable.

If there is an open file descriptor associated with the catalog file, the `FD_CLOEXEC` flag will be set.
The LANG environment variable is used to refer to message catalogs that are separated into directories based on natural languages. For example, if the catopen() function specifies a catalog with the name mycmd, and the environment variables are set as follows:

NLSPATH=.:/%N:/system/nls/%L/%N:/system/nls/%N
LANG=Fr_FR

then the application searches for the catalog in the following order:

. ./mycmd.
/mycmd
/system/nls/Fr_FR/mycmd
/system/nls/mycmd

If you omit the variable %N in a directory specification within the environment variable NLSPATH, the application assumes that the path defines a directory and searches for the catalog in that directory before searching the next specified path. The value /usr/lib/nls/msg/%L/%N:/etc/nls/%L/%N is the default path for NLSPATH.

Notes

AES Support Level: Trial use

Return Values

The catopen() function returns a value of -1 if the number of catalogs already open is equal to the NL_MAXOPEN limit defined in the msg.h header file.

The catopen() function also returns a value of -1 if it cannot find the file.

Related Information

Functions: catgets(3), catclose(3)

Commands: dspcat(1), dspmsg(1), gencat(1)
cfgetispeed

Purpose  Gets input baud rate for a terminal

Library  Standard C Library (libc.a)

Synopsis  

```
#include <termios.h>
speed_t cfgetispeed (  
    struct termios *termios_p);
```

Parameters  

termios_p  Points to a termios structure containing the input baud rate.

Description  
The cfgetispeed() function extracts the input baud rate from the termios structure to which the termios_p parameter points.

If the value in the termios structure was not obtained from a successful call to the tcgetattr() function, the behavior is undefined.

Notes  
AES Support Level: Full use

Return Values  
Upon successful completion, the cfgetispeed() function returns a value of type speed_t representing the input baud rate.

Related Information  
Functions: cfgetospeed(3), cfsetispeed(3), cfsetospeed(3), tcgetattr(3)  
Files: termios(4)
cfgetospeed

Purpose

Gets output baud rate for a terminal.

Library

Standard C Library (libc.a)

Synopsis

```
#include <termios.h>

speed_t cfgetospeed (
    struct termios *termios_p);
```

Parameters

- `termios_p` Points to a `termios` structure containing the output baud rate.

Description

The `cfgetospeed()` function extracts the output baud rate from the `termios` structure to which the `termios_p` parameter points.

If the value in the `termios` structure was not obtained from a successful call to the `tcgetattr()` function, the behavior is undefined.

Notes

AES Support Level: Full use

Return Values

Upon successful completion, the `cfgetospeed()` function returns a value of type `speed_t` representing the output baud rate.

Related Information

Functions: `cfgetispeed(3), cfsetispeed(3), cfsetospeed(3), tcgetattr(3)`

Files: `termios(4)`
Functions

**cfsetispeed(3)**

---

**cfsetispeed**

**Purpose**
Sets input baud rate for a terminal

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <termios.h>

int cfsetispeed (  
    struct termios *termios_p,  
    speed_t speed );
```

**Parameters**
- `termios_p` Points to a `termios` structure containing the input baud rate.
- `speed` Specifies the new input baud rate.

**Description**
The `cfsetispeed() ` function sets the input baud rate stored in the structure pointed to by the `termios_p` parameter to the value specified by the `speed` parameter.

If the input baud rate is set to 0 (zero), the input baud rate will be specified by the value of the output baud rate.

There is no effect on the baud rates set in the hardware until a subsequent successful call to the `tcsetattr()` function on the same `termios` structure.

**Notes**

AES Support Level: Full use

**Return Values**
The `cfsetispeed()` function returns a value of 0 (zero).

**Related Information**
Functions: `cfgetspeed(3), cfgetospeed(3), cfsetospeed(3), tcsetattr(3)`
Files: `termios(4)`
### cfsetospeed

#### Purpose
Sets output baud rate for a terminal

#### Library
Standard C Library (libc.a)

#### Synopsis

```c
#include <termios.h>

int cfsetospeed (
    struct termios *termios_p,
    speed_t speed);
```

#### Parameters
- `termios_p`: Points to a `termios` structure containing the output baud rate.
- `speed`: Specifies the new output baud rate.

#### Description
The `cfsetospeed()` function sets the output baud rate stored in the structure pointed to by the `termios_p` parameter to the speed specified by the `speed` parameter.

The zero baud rate, B0, is used to terminate the connection. If B0 is specified, the modem control lines are no longer asserted. Normally, this disconnects the line.

There is no effect on the baud rates set in the hardware or on modem control lines until a subsequent successful call to the `tcsetattr()` function on the same `termios` structure.

#### Notes

AES Support Level: Full use

#### Return Values
The `cfsetospeed()` function returns 0 (zero).

#### Related Information
Functions: `cfgetispeed(3), cfgetospeed(3), cfsetispeed(3), tcsetattr(3)`
Files: `termios(4)`
chdir, fchdir

Purpose
Changes the current directory

Synopsis
int chdir (
    const char *path);

int fchdir (
    int filedes);

Parameters
path Points to the pathname of the directory.
filedes Specifies the file descriptor of the directory.

Description
The chdir() function changes the current directory to the directory indicated by
the path parameter.
The fchdir() function changes the current directory to the directory indicated by
the filedes parameter.
If the path parameter refers to a symbolic link, the chdir() function sets the current
directory to the directory pointed to by the symbolic link.
The current directory, also called the current working directory, is the starting
point of searches for pathnames that do not begin with a / (slash). In order for a
directory to become the current directory, the calling process must have search
access to the directory.

Notes
The current working directory is shared between all threads within the same
process. Therefore, one thread using the chdir() or fchdir() functions will affect
every other thread in that process.

AES Support Level: Full use chdir() only
Return Values

Upon successful completion, the `chdir()` function returns a value of 0 (zero). Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `chdir()` function fails, the current directory remains unchanged and `errno` may be set to one of the following values:

- `[EACCES]` Search access is denied for any component of the pathname.
- `[ELOOP]` Too many symbolic links were encountered in translating the pathname.
- `[EFAULT]` The `path` parameter points outside the process’s allocated address space.
- `[EIO]` An I/O error occurred while reading from or writing to the file system.
- `[ENOENT]` The named directory does not exist, or is an empty string.
- `[ENOTDIR]` A component of the path prefix is not a directory.
- `[ENAMETOOLONG]` The length of the `path` argument exceeds PATH_MAX or a pathname component is longer than NAME_MAX.

If the `fchdir()` function fails, the current directory remains unchanged and `errno` may be set to one of the following values:

- `[ENOTDIR]` The file descriptor does not reference a directory.
- `[EBADF]` The `filedes` parameter is not a valid open file descriptor.

Related Information

Functions: `chroot(2)`

Commands: `cd(1)`
chmod, fchmod

**Purpose** Changes file access permissions

**Synopsis**
```c
#include <sys/mode.h>
#include <sys/types.h>
#include <sys/stat.h>

int chmod (const char *path, mode_t mode);

int fchmod (int filedes, mode_t mode);
```

**Parameters**
- `path` Specifies the full pathname of the file. If the `path` parameter refers to a symbolic link, the `chmod()` function changes access permissions on the file specified by the symbolic link.
- `filedes` Specifies the file descriptor of an open file.
- `mode` Specifies the bit pattern which determines the access permissions.

**Description**

The `chmod()` function sets the access permissions of the file specified by the `path` parameter according to the bit pattern specified by the `mode` parameter.

The `fchmod()` function sets the access permissions of an open file pointed to by the `filedes` parameter according to the bit pattern specified by the `mode` parameter.

To change file access permissions, the process must have the same effective user ID as the owner of the file or the process must have superuser privilege.

Upon successful completion, the `chmod()` and `fchmod()` functions mark the `st_ctime` field of the file for update.

The `mode` parameter is constructed by logically ORing one or more of the following values, which are defined in the `sys/mode.h` header file:
- `S_ISUID` Sets the process' effective user ID to the file's owner on execution.
- `S_ISGID` Sets the process' effective group ID to the file's group on execution.
chmod(2)

S_ISVTX  Saves text image after execution.
S_IRWXU  Permits the file's owner to read, write, and execute it (or to search the directory).
S_IRUSR  Permits the file's owner to read it.
S_IWUSR  Permits the file's owner to write to it.
S_IXUSR  Permits the file's owner to execute it (or to search the directory).
S_IRWXG  Permits the file's group to read, write, and execute it (or to search the directory).
S_IRGRP  Permits the file's group to read it.
S_IWGRP  Permits the file's group to write to it.
S_IXGRP  Permits the file's group to execute it (or to search the directory).
S_IRWXO  Permits others to read, write, and execute it (or to search the directory).
S_IROTH  Permits others to read the file.
S_IWOTH  Permits others to write to the file.
S_IXOTH  Permits others to execute the file (or to search the directory).

Other mode values exist that can be set with the mknod() function, but not with the chmod() function.

If the mode bit S_ISGID is set and the mode bit S_IXGRP is not set, mandatory file record locking will exist on a regular file. This may affect subsequent calls to other calls on the file, including open(), creat(), read(), write(), and truncate().

The S_ISGID bit of the file is cleared if:

- The file is a regular file.
- The effective user ID of the process does not have appropriate system privilege.
- The effective group ID or one of the IDs in the group access list of the process does not match the file's existing group ID.

AES Support Level: Full use (chmod())
Trial use (fchmod())
Functions

chmod(2)

Return Values

Upon successful completion, the \texttt{chmod}() and \texttt{fchmod}() functions return a value of 0 (zero). If the \texttt{chmod}() or \texttt{fchmod}() function fails, a value of -1 is returned, and \texttt{errno} is set to indicate the error.

Errors

If the \texttt{chmod}() function fails, the file permissions remain unchanged and \texttt{errno} may be set to one of the following values:

- [ENOTDIR] A component of the \texttt{path} parameter is not a directory.
- [ENOENT] The named file does not exist or is an empty string.
- [ENOENT] A symbolic link was named, but the file to which it refers does not exist.
- [EACCES] A component of the \texttt{path} parameter has search permission denied.
- [EPERM] The effective user ID does not match the ID of the owner of the file or the owner does not have appropriate system privilege.
- [EROFS] The named file resides on a read-only file system.
- [EFAULT] The \texttt{path} parameter points to a location outside of the allocated address space of the process.
- [ESTALE] The process' root or current directory is located in a virtual file system that has been unmounted.
- [ELOOP] Too many symbolic links were encountered in translating the \texttt{path} parameter.
- [ENAMETOOLONG] The length of the \texttt{path} argument exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- [EINTR] A signal was caught during execution of the system call.
If the `fchmod()` function fails, the file permissions remain unchanged and `errno` may be set to one of the following values:

- **[EBADF]** The file descriptor `filedes` is not valid.
- **[EROFS]** The file referred to by `filedes` resides on a read-only file system.
- **[EPERM]** The effective user ID does not match the ID of the owner of the file, and the calling process does not have superuser privilege.
- **[ESTALE]** The process' root or current directory is located in a virtual file system that has been unmounted.
- **[EINTR]** A signal was caught during execution of the system call.

**Related Information**

Functions: `chown(2)`, `fcntl(2)`, `getgroups(2)`, `mknod(2)`, `open(2)`, `read(2)`

Commands: `chmod(1)`
chown, fchown

Purpose
Changes the owner and group IDs of a file

Synopsis

```c
int chown(
    const char *path,
    uid_t owner,
    gid_t group);
```

```c
int fchown(
    int filedes,
    uid_t owner,
    gid_t group);
```

Parameters

- **path**: Specifies the name of the file whose owner ID, group ID, or both are to be changed. If the `path` parameter refers to a symbolic link, the `chown()` function changes the ownership of the file pointed to by the symbolic link.
- **filedes**: Specifies a valid open file descriptor.
- **owner**: Specifies a numeric value representing the owner ID.
- **group**: Specifies a numeric value representing the group ID.

Description

The `chown()` and `fchown()` functions change the owner and group of a file.

A process can change the value of the owner ID of a file only if the process has superuser privilege. A process can change the value of the file group ID if the effective user ID of the process matches the owner ID of the file, or if the process has superuser privilege. A process without superuser privilege can change the group ID of a file only to the value of its effective group ID or to a value in its supplementary group list.

If the value of the owner ID is changed and the process does not have superuser privilege, the set-user ID attribute (the S_ISUID bit) of a regular file is cleared.
The set-user ID attribute (S_ISUID bit) of a file is cleared upon successful return if:

- The file is a regular file.
- The process does not have superuser privilege.

The set-group ID attribute (S_ISGID bit) of a file is cleared upon successful return if:

- The file is a regular file.
- The process does not have superuser privilege.

If the `owner` or `group` parameter is specified as `(uid_t)-1` or `(gid_t)-t` respectively, the corresponding ID of the file is unchanged.

Upon successful completion, the `chown()` and `fchown()` functions mark the `st_ctime` field of the file for update.

**AES Support Level:** Full use (`chown()`)

Trial use (`fchown()`)

### Return Values

Upon successful completion, the `chown()` and `fchown()` functions return a value of 0 (zero). Otherwise, a value of -1 is returned, the owner and group of the file remain unchanged, and `errno` is set to indicate the error.

### Errors

If the `chown()` function fails, `errno` may be set to one of the following values:

- **[EACCES]** Search permission is denied on a component of `path`.
- **[EFAULT]** The `path` parameter is an invalid address.
- **[ELOOP]** Too many links were encountered in translating `path`.
- **[ENAMETOOLONG]** The length of the `path` argument exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- **[ENOTDIR]** A component of `path` is not a directory.
- **[ENOENT]** The `path` parameter does not exist or is an empty string.
- **[EPERM]** The effective user ID does not match the ID of the owner of the file, and the calling process does not have appropriate privilege.
- **[EROFS]** The named file resides on a read-only file system.
If the `fchown()` function fails, `errno` may be set to one of the following values:

- `[EBADF]` The file descriptor `filedes` is not valid.
- `[EROFS]` The file referred to by `filedes` resides on a read-only file system.
- `[EPERM]` The effective user ID does not match the ID of the owner of the file, and the calling process does not have appropriate privilege.

**Related Information**

Functions: `chmod(2)`, `chmod(2)`
Commands: `chown(1)`
chroot

Purpose
Changes the effective root directory

Synopsis
int chroot (const char *path);

Parameters
Path
Points to the new effective root directory. If the path parameter refers to a symbolic link, the chroot() function sets the effective root directory to the directory pointed to by the symbolic link.

Description
The chroot() function causes the directory named by the path parameter to become the effective root directory.

The effective root directory is the starting point when searching for a file's pathname that begins with a / (slash). The current working directory is not affected by the chroot() function.

The calling process must have superuser privilege in order to change the effective root directory. The calling process must also have search access to the new effective root directory.

The .. (dot-dot) entry in the effective root directory is interpreted to mean the effective root directory itself. Thus, .. (dot-dot) cannot be used to access files outside the subtree rooted at the effective root directory.

Notes
AES Support Level: Trial use

Return Values
Upon successful completion, a value of 0 (zero) is returned. If the chroot() function fails, a value of -1 is returned and errno is set to indicate the error.
Errors

If the `chroot()` function fails, the effective root directory remains unchanged and `errno` may be set to one of the following values:

- **[EACCESS]** Search permission is denied for any component of the pathname.
- **[EPERM]** The process does not have appropriate privilege.
- **[EFAULT]** The `path` parameter points outside the process' allocated address space.
- **[EIO]** An I/O error occurred while reading from or writing to the file system.
- **[ENOENT]** The `path` parameter does not exist or points to an empty string.
- **[ENAMETOOLONG]** The length of the `path` argument exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- **[ENOTDIR]** A component of `path` is not a directory.
- **[ELOOP]** More than MAXSYMLINKS symbolic links are encountered while resolving `path`.

Related Information

Functions: `chdir(2)`

Commands: `chdir(1)`
clearenv

**Purpose**
Clears the process environment

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <stdlib.h>
int clearenv ( void );
```

**Description**
The `clearenv()` function clears the process environment. No environment variables are defined immediately after a call to `clearenv()`. The `clearenv()` function sets the value of the external variable `environ` to NULL.

**Notes**
AES Support Level: Trial use

**Return Values**
Upon successful completion, the `clearenv()` function returns 0 (zero). Otherwise, it returns -1.

If `environ` has been modified by anything other than the `putenv()`, `getenv()`, or `clearenv()` functions, then `clearenv()` will return an error and the process environment will remain unchanged.

**Related Information**
Functions: `exec(2)`, `getenv(3)`, `putenv(3)`
clearerr

Purpose       Clears indicators on a stream

Library       

Standard I/O Package (libc.a)

Synopsis      

#include <stdio.h>

void clearerr (  
    FILE *stream );

Parameters     

stream       Specifies the input or output stream to be cleared.

Description    

The clearerr( ) function resets the error indicator and the EOF indicator for the stream specified by the stream parameter.

Notes          

The clearerr( ) function is supported for multi-threaded applications.

AES Support Level:  Full use

Return Values  

The clearerr( ) function returns no value.

Related Information

Functions:  open(2), fopen(3), feof(3), fileno(3), ferror(3)
clock

**Purpose**
Reports CPU time used

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <time.h>
clock_t clock (void);
```

**Description**
The `clock()` function reports the amount of processor time used by the calling process.

**Notes**
The `clock()` function is made obsolete by the `getrusage()` function; however, it is included for compatibility with older BSD programs.

**AES Support Level:** Full use

**Return Values**
The `clock()` function returns the amount of processor time (in microseconds) used since the first call to `clock()`. To determine the time in seconds, divide the value returned by `clock()` by the value `CLOCKS_PER_SEC`. If the processor time used is not available or its value cannot be represented, the `clock()` function returns `(clock_t)-1`.

**Related Information**
Functions: `getrusage(2), times(3), wait(2)`
close

Purpose
Closes the file associated with a file descriptor.

Synopsis
int close ( int filedes );

Parameters
filedes Specifies a valid open file descriptor.

Description
The close() function closes the file associated with the filedes parameter.

All regions of a file specified by the filedes parameter that this process has previously locked with the lockf() function are unlocked. This occurs even if the process still has the file open by another file descriptor.

When all file descriptors associated with a pipe or FIFO special file have been closed, any data remaining in the pipe or FIFO is discarded. When all file descriptors associated with an open file descriptor are closed, the open file descriptor is freed. If the link count of the file is 0 (zero) when all file descriptors associated with the file have been closed, the space occupied by the file is freed and the file is no longer accessible.

When the close() function needs to block, only the calling thread is suspended rather than all threads in the calling process.

Notes

AES Support Level: Full use

Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
Errors

If the close() function fails, errno may be set to one of the following values:

- [EBADF] The filedes parameter is not a valid open file descriptor.
- [EINTR] The close() function was interrupted by a signal which was caught.

Related Information

Functions: exec(2), fcntl(2), lockf(3), open(2), open(2), pipe(2), socket(2)
connect

Purpose Connects two sockets

Synopsis

```c
#include <sys/types.h>
#include <sys/socket.h>
int connect (
    int socket,
    struct sockaddr *address,
    int address_len);
```

Parameters

- `socket` Specifies the unique name of the socket.
- `address` Points to a `sockaddr` structure, the format of which is determined by the domain and by the behavior requested for the socket. The `sockaddr` structure is an overlay for a `sockaddr_in`, `sockaddr_un`, or `sockaddr_ns` structure, depending on which of the supported address families is active. If the compile-time option `_SOCKADDR_LEN` is defined before the `sys/socket.h` header file is included, the `sockaddr` structure takes 4.4BSD behavior, with a field for specifying the length of the socket address. Otherwise, the default 4.3BSD `sockaddr` structure is used, with the length of the socket address assumed to be 14 bytes or less.

  If `_SOCKADDR_LEN` is defined, the 4.3BSD `sockaddr` structure is defined with the name `osockaddr`.

- `address_len` Specifies the length of the `sockaddr` structure pointed to by the `address` parameter.
Description

The `connect()` function requests a connection between two sockets. The kernel sets up the communications links between the sockets; both sockets must use the same address format and protocol.

The `connect()` function performs a different action for each of the following types of initiating sockets:

- If the initiating socket is `SOCK_DGRAM`, then the `connect()` function establishes the peer address. The peer address identifies the socket where all datagrams are sent on subsequent `send()` functions. No connections are made by this `connect` function.

- If the initiating socket is `SOCK_STREAM`, then the `connect()` function attempts to make a connection to the socket specified by the `address` parameter. Each communication space interprets the `address` parameter differently.

Return Values

Upon successful completion, the `connect()` function returns a value of 0 (zero). Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `connect()` function fails, `errno` may be set to one of the following values:

- [EBADF] The `socket` parameter is not valid.
- [ENOTSOCK] The `socket` parameter refers to a file, not a socket.
- [EADDRNOTAVAIL] The specified address is not available from the local machine.
- [EAFNOSUPPORT] The addresses in the specified address family cannot be used with this socket.
- [EISCONN] The socket is already connected.
- [ETIMEDOUT] The establishment of a connection timed out before a connection was made.
- [ECONNREFUSED] The attempt to connect was rejected.
Functions

**connect(2)**

[ENETUNREACH]  
No route to the network or host is present.

[EADDRINUSE]  
The specified address is already in use.

[EFAULT]  
The `address` parameter is not in a readable part of the user address space.

[EWOULDBLOCK]  
The socket is marked nonblocking, so the connection cannot be immediately completed. The application program can select the socket for writing during the connection process.

**Related Information**

Functions: **accept(2), bind(2), socket(2), getsockname(2), select(2), send(2)**
toascii, tolower, _tolower, toupper, _toupper

Purpose  Translates characters

Library  Standard C Library (libc.a)

Synopsis  #include <ctype.h>

int toascii(
    int c);
int tolower(
    int c);
int _tolower(
    int c);
int toupper(
    int c);
int _toupper(
    int c);

Parameters

   c  Specifies the character to be converted.

Description

The toascii(), tolower(), _tolower(), toupper(), and _toupper() functions translate all characters, including extended characters, to their specified character values.

The toascii() function converts its input to a 7-bit ASCII character.

The tolower() function takes an int value that can be represented as an unsigned char or the value of EOF (defined in the stdio.h header file) as its input.

When the input of the tolower() function expresses an uppercase letter, as defined by character type information in the program locale (category LC_CTYPE), the corresponding lowercase letter is returned. All other input values in the domain are returned unchanged. The tolower() function has as its domain the range -1 through 255.
In the C locale, or in a locale where case-conversion information is not defined, the `tolower()` function determines the case of characters according to the rules of the ASCII-coded character set. Characters outside the ASCII range of characters are returned unchanged.

The `_tolower()` macro is equivalent to the `tolower()` function, but executes faster.

The `toupper()` function takes an `int` value that can be represented as an `unsigned char` or the value of `EOF` (defined in the `stdio.h` header file) as its input.

When the input of the `toupper()` function expresses a lowercase letter, as defined by character type information in the program locale (category LC_CTYPE), the corresponding uppercase letter is returned. All other input values in the domain are returned unchanged. The `toupper()` function has as its domain the range -1 through 255.

In the C locale, or in a locale where case-conversion information is not defined, the `toupper()` function determines the case of characters according to the rules of the ASCII-coded character set. Characters outside the ASCII range of characters are returned unchanged.

The `_toupper()` macro is equivalent to the `toupper()` function, but executes faster.

Notes

The `setlocale()` function affects all conversions. See the `setlocale()` function for more information.

**AES Support Level:** Full use (`tolower()`, `toupper()`)

Trial use (`_tolower()`, `_toupper()`, `toascii()`)

Return Values

The `toascii()` function returns the logical AND of parameter `c` and the value `0X7F`.

When the `c` parameter is a character for which the `isupper()` function is TRUE, there is a corresponding character for which the `islower()` function is also TRUE. That lowercase character is returned by the `tolower()` function or by the `_tolower()` macro. Otherwise the `c` parameter is returned.
When the c parameter is a character for which the islower() function is TRUE, there is a corresponding character for which the isupper() function is also TRUE. That uppercase character is returned by the toupper() function or by the _toupper() macro. Otherwise, the c parameter is returned.

Related Information

Functions: ctype(3), setlocale(3)
ctermid

Purpose
Generates the pathname for the controlling terminal

Library
Standard I/O Package (libc.a)

Synopsis
#include <stdio.h>
char *ctermid (char *s);

Parameters
s
If the s parameter is a null pointer, the string is stored in an internal static area and the address is returned. The next call to the ctermid() function overwrites the contents of the internal static area.

If the s parameter is not a null pointer, it points to a character array of at least L_ctermid bytes. L_ctermid is defined in the stdio.h header file, and has a value greater than 0 (zero). The pathname is placed in this array and the value of the s parameter is returned.

Description
The ctermid() function generates the pathname of the controlling terminal for the current process and stores it in a string.

The ctermid() function differs from the ttyname() function in that the ttyname() function is supplied a file descriptor and returns the actual name of the terminal associated with that file descriptor, while the ctermid() function returns a string (/dev/tty) that refers to the terminal if used as a filename. Thus, the ttyname() function is useful only if the process already has at least one file open to a terminal.

Notes
AES Support Level: Full use
Return Values

Upon successful completion, the `ctermid()` function returns the address of the generated pathname. Otherwise, an empty string is returned. Access to a pathname returned by the `ctermid()` function is not guaranteed.

The `ctermid_r()` function, the reentrant version of the `ctermid()` function, always returns null if the argument passed is null.

Related Information

Functions: `ttyname(3)`
Functions

ctime(3)

asctime, asctime_r, ctime, ctime_r, difftime, gmtime,
gmtime_r, localtime, localtime_r, mktime, tzset

Purpose
Converting time units

Library
Standard C Library (libc.a)

Synopsis
#include <time.h>

char *asctime(
    const struct tm *timeptr);

int asctime_r(
    const struct tm *timeptr,
    char *buffer,
    int len);

char *ctime(
    const time_t *timer);

int ctime_r(
    const time_t *timer,
    char *buffer,
    int len);

double difftime(
    time_t time1,
    time_t time2);

struct tm *gmtime(
    const time_t *timer);

int gmtime_r(
    struct tm *result,
    const time_t timer);

struct tm *localtime(
    const time_t *timer);

int localtime_r(
    struct tm *result,
    const time_t timer);
Parameters

timeptr
    Points to a type tm structure that defines space for broken-down time.

time1
    Specifies a time value expressed in seconds.

time2
    Specifies a time value expressed in seconds.

timer
    Points to a variable that specifies a time value in seconds.

buffer
    Points to a character array used to store the generated date and time string.

len
    Specifies an integer that defines the length of the character array.

Description

The asctime(), asctime_r(), ctime(), difftime(), gmtime(), gmtime_r(), localtime(), localtime_r(), mktime(), and tzset() functions are used to convert time units to strings, to store converted time units for subsequent processing, and to convert stored time information to other time units. Time information used in these functions is stored in a type tm structure, which is defined in the time.h include file.

The asctime() function converts type tm structure broken-down time information pointed to by the timeptr parameter to a date and time string with the following 5-field format:

    Sun Sep 16 01:03:52 1973

The asctime_r() function is the reentrant version of asctime() for use with multiple threads.

The ctime() function converts the time in seconds since the Epoch, pointed to by the timer parameter, to a character string. The Epoch is taken as 00:00:00 GMT 1 Jan 1970. The character string specifies local time in the same format as does the asctime() function. Local time-zone information is set as though the tzset() function were called. This function is equivalent to asctime (localtime (timer)).

The reentrant version of this function is identical, except that it stores the string in the buffer parameter up to len characters.
The `difftime()` function returns a signed time value in seconds that is the difference between the values of the `time1` and `time2` parameters, also expressed in seconds.

The `gmtime()` function converts the time in seconds since the Epoch, pointed to by the `timer` parameter, into broken-down time, expressed as CUT (Coordinated Universal Time). Broken-down time is stored in the type `tm` structure pointed to by the return value of the `gmtime()` function.

The `gmtime_r()` function is the reentrant version of `gmtime()`. This information is stored in the `tm` structure passed in the `result` parameter.

The `localtime()` function converts the time in seconds since the Epoch, pointed to by the `timer` parameter, into broken-down time, expressed as local time. This function corrects for the time-zone and any seasonal time adjustments. Broken-down time is stored in the type `tm` structure pointed to by the return value of this function. Local time-zone information is set as though the `tzset()` function were called.

The `localtime_r()` function is the reentrant version of `localtime()` for use with multiple threads.

The `mktime()` function converts the broken-down time, expressed as local time, in the type `tm` structure pointed to by the `timeptr` parameter, into a time since the Epoch in the same format as that of values returned by the `time()` function. The original values of parameters `timeptr->tm_wday` and `timeptr->tm_yday` of the structure are ignored, and the original values of other members of the structure are not restricted to the ranges defined in the `time.h` header file. The range [0, 61] for structure member `tm_sec` allows for an occasional leap second or double leap second.

A positive or 0 (zero) value for member `tm_isdst` tells the `mktime()` function whether daylight saving time is in effect. A negative value for `tm_isdst` tells the `mktime()` function to find out whether daylight saving time is in effect for the specified time. Local time-zone information is set as though the `tzset()` function were called.

On successful completion, values for the `timeptr->tm_wday` and `timeptr->tm_yday` members of the structure are set, and the other members are set to specified times since the Epoch, but with their values forced to the ranges indicated above; the final value of `timeptr->tm_mday` is not set until the values of members `timeptr->tm_mon` and `timeptr->tm_year` are determined.

The `tzset()` function uses the value of the environment variable `TZ` to set time conversion information used by the `localtime()`, `localtime_r()`, `ctime()`, `ctime_r()`, `strftime()`, and `mktime()` functions. When environment variable `TZ` is absent, implementation-defined default time-zone information is used.

When the `TZ` environment variable is defined, the defined value overrides the default time-zone value. The `environment` facility contains formatted time zone
information specified by `TZ`. Environment variable `TZ` is usually set when a system is started with the value that is defined in either the `/etc/environment` or `/etc/profile` files. However, `TZ` may also be set by a user as a regular environment variable for converting to alternate time zones.

The `tzset()` function sets the external variable `tzname` as follows:

```c
    tzname[0] = std;
    tzname[1] = dst;
```

where `std` and `dst` are the strings designating standard and daylight saving time zones, respectively, as described for the `TZ` environment variable.

The `tzset()` function also sets the external variable `daylight` to 0 (zero) when daylight saving time conversions should never be applied for the time zone in use; otherwise `daylight` is set to a nonzero value. The external variable `timezone` is set to the difference, in seconds, between Coordinated Universal Time (CUT) and local standard time. In the following table, entries in the `TZ` column are time-zone environmental variables, and entries in the `Timezone` column are time units expressed as UTC time.

<table>
<thead>
<tr>
<th>TZ</th>
<th>Timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>EST</td>
<td>5<em>60</em>60</td>
</tr>
<tr>
<td>GMT</td>
<td>0<em>60</em>60</td>
</tr>
<tr>
<td>JST</td>
<td>-9<em>60</em>60</td>
</tr>
<tr>
<td>MET</td>
<td>-1<em>60</em>60</td>
</tr>
<tr>
<td>MST</td>
<td>7<em>60</em>60</td>
</tr>
<tr>
<td>PST</td>
<td>8<em>60</em>60</td>
</tr>
</tbody>
</table>

External variable `tzname` specifies the name of the standard time zone (`tzname[0]`) and of the time zone when daylight saving time is in effect (`tzname[1]`). For example:

```c
    extern char *tzname[2] = {"EST","EDT"};
```

External variable `timezone` specifies the difference, in seconds, between GMT and local standard time. For example, the value of `timezone` is `5 * 60 * 60` for U.S. Eastern Standard Time.

External variable `daylight` is set nonzero when a daylight saving time conversion should be applied. By default, this conversion follows standard U.S. time conventions; other time conventions may be specified. The default conversion algorithm adjusts for peculiarities of U.S. daylight saving time in 1974 and 1975.
Notes

The `asctime()`, `ctime()`, `gmtime()`, and `localtime()` functions are not supported for multi-threaded applications. Instead, their reentrant equivalents, `asctime_r()`, `ctime_r()`, `gmtime_r()`, and `localtime_r()`, should be used with multiple threads.

The `difftime()`, `mktime()`, and `tzset()` functions are supported for multi-threaded applications.

AES Support Level: Full use (asctime(), ctime(), difftime(), gmtime(), localtime(), mktime(), tzset())

Return Values

When any of the `asctime()`, `ctime()`, `gmtime()`, or `localtime()` functions complete successfully, the return value may point to static storage, which may be overwritten by subsequent calls to these functions. On error, these functions return a null pointer and `errno` is set to a value indicating the error.

Upon successful completion, the `asctime()` and `ctime()` functions return a pointer to a character string that expresses the time in a fixed format.

Upon successful completion the `difftime()` function returns a value, expressed in seconds, that is the difference between the values of parameters `time1` and `time2`.

Upon successful completion, the `gmtime()` and `gmtime_r()` functions return a pointer to a type `tm` broken-down time structure, which contains converted GMT time information. When UTC is not available, this function returns a null pointer.

Upon successful completion, the `localtime()` functions return a pointer to a type `tm` broken-down time structure, which contains converted local time.

Upon successful completion, the `mktime()` function returns the specified time since the Epoch written as a value of type `time_t`. On error, or whenever the time since the Epoch cannot be represented, this function returns the value `(time_t)-1`, and sets `errno` to indicate the error. This function does not return a value.

Upon successful completion, the `asctime_r()`, `ctime_r()`, `gmtime_r()`, and `localtime_r()`, functions return a value of 0 (zero). Otherwise, -1 is returned and `errno` is set to indicate the error.
Errors

If any of these functions fails, \texttt{errno} may be set to the following value:

[EINVAL] The \textit{buffer} or \textit{timer} parameter is null, the \textit{len} parameter is 0 (zero), or the specified broken-down time can not be represented as time since the Epoch.

Related Information

Functions: \texttt{getenv(3), strftime(3), time(3)}
isalpha, isupper, islower, isdigit, isxdigit, isalnum, ispace, ispunct, isprint, isgraph, iscntrl, isascii

Purpose
Classifies characters

Library
Standard C Library (libc.a)

Synopsis
#include <ctype.h>

int isalpha(
    int c);

int isupper(
    int c);

int islower(
    int c);

int isdigit(
    int c);

int isxdigit(
    int c);

int isalnum(
    int c);

int isspace
    int c);

int ispunct(
    int c);

int isprint(
    int c);

int isgraph(
    int c);

int iscntrl(
    int c);

int isascii(
    int c);
ctype(3)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Specifies the character to be tested. In all cases, this parameter is an int data type, whose value must be representable as an unsigned char or must equal the value of the macro EOF (defined in the stdio.h include file). When this parameter has a value that can not be represented as an unsigned char or EOF, the result is undefined.</td>
</tr>
</tbody>
</table>

Description

The ctype functions classify character-coded integer values specified in a table. Each of these functions returns a nonzero value for TRUE and 0 (zero) for FALSE.

The ctype functions, which are defined in the ctype.h include file, are defined as subroutines in the sys/locale.h include file. These functions classify character-coded integer values specified in a table. Each function returns a nonzero value for TRUE and 0 (zero) for FALSE.

For international character support, these operations are implemented as functions. To increase performance in a U.S. English-only environment, the ctype functions are used. However, when the sys/locale.h include file is referenced, the assumption is that international character support is desired, so subroutines are used in place of macros.

The isascii() function is defined for all integer values. All other functions return a meaningful value only when isascii() returns TRUE for the same c parameter value or when c is EOF. (See Standard Input/Output Library for information about the value EOF.)

Function Values

The following lists the set of values for which each function listed in the ctype.h include file returns a nonzero (TRUE) value:

- isalnum() When c is a letter or a digit.
- isalpha When c is a letter.
- isupper When c is an uppercase letter.
- islower When c is a lowercase letter.
- isdigit When c is a digit in the range [0-9].
- isxdigit() When c is a hexadecimal digit in the range [0-9], [A-F], or [a-f].
- isspace() When c is a space, tab, carriage return, newline, vertical tab, or form feed character.
- ispunct() When c is a punctuation character (neither a control character nor an alphanumeric character).

1–90
isprint() When \( c \) is a printing character, ASCII space (040 or 0x20) through ~ (0176 or 0x7E).

isgraph() When \( c \) is a printing character, like isprint(). Unlike isprint(), isgraph() returns FALSE for the space character.

iscntrl() When \( c \) is an ASCII delete character (0177 or 0x7F), or an ordinary control character (less than 040 or 0x20).

isascii() When \( c \) is an ASCII character whose value is in the range 0-0177 (0-0x7F), inclusive.

Notes

The setlocale() function affects all conversions. See the setlocale() function for more information.

In the C locale, or in a locale where character-type information is not defined, characters are classified according to the rules of the US-ASCII 7-bit coded character set. For any character value greater than octal 177 (0177 in C-language context) the value 0 (zero) is returned.

AES Support Level: Full use

Return Values

Upon successful completion of any function, a nonzero (TRUE) value is returned. Otherwise, the value 0 (FALSE) is returned.

Related Information

Functions: ctype(3), setlocale(3)
curses Library

Purpose Controls cursor movement and windowing

Library Curses Library (libcurses.a)

Synopsis

```
#include <curses.h>
#include <term.h>
```

Description

The `curses` library is a screen manipulation package.

The full `curses` interface allows you to manipulate structures called `windows`, which can be thought of as two-dimensional arrays of characters representing all or part of the screen. A default window (called `stdscr`) is supplied, and you can create others using the `newwin()` function. Windows are referred to by variables declared as type `WINDOW *`, defined in the `curses.h` header file. (The `term.h` header file should be used only for using the `terminfo` level functions.)

Routine names beginning with "w" allow you to specify a window. Routine names not beginning with a "w" affect only `stdscr`.

The `minicurses` package is a subset of `curses` that does not allow you to manipulate more than one window. This subset is invoked with the `-DMINICURSES` option to `cc`. This subset is smaller and faster than the full `curses` interface.

If your program needs only one terminal, you can specify the `-DSINGLE` flag to the C compiler. This results in static references instead of dynamic references to capabilities. The result is more concise code, but only one terminal can be used at a time for the program.

To initialize the functions which are described in the `curses` library, you must call the `initscr()` function before using any other functions which affect windows and screens, and the `endwin()` function before exiting.

Screen Dimensions

The screen is a matrix of character positions that can contain any character from the terminal's character set. The actual dimensions of the matrix are different for each type of terminal. These dimensions are defined when the `initscr()` function
calls the terminfo initialization function, setupterm(). The functions enforce the following limits on the terminal:

- If the terminal specification defines less than 5 lines, the functions use a value of 24 lines.
- If the terminal specification defines less than 5 columns, the functions use a value of 80 columns.

Note that line values (y coordinates) are specified first to the library functions which request line and column values.

To update the screen, the functions must know what the screen currently looks like and what it should be changed to. The functions define the WINDOW data type to hold this information. This data type is a structure that describes a window image to the functions, including the starting position on the screen (the (line, col) coordinates of the upper left corner) and size.

You can think of a window as an array of characters on which to make changes. Using the window, a program builds and stores an image of a portion of the terminal that it later transfers to the actual screen. When the window is complete, use one of the following functions to transfer the window to the terminal:

- refresh: Transfers the contents of stdscr to the terminal.
- wrefresh: Transfers the contents of a named window (not stdscr) to the terminal.

This two-step process maintains several different copies of a window in memory and selects the proper one to display at any time. In addition, the program can change the contents of the screen in any order. When it has made all of the changes, the library functions update the terminal in an efficient manner.

The Curses Routines

The curses functions are summarized below:

`int addch( ctype ch );`
Add a character to stdscr, wrapping to the next line at the end of a line (like putchar()). May be called with minicurses.

`int waddch( WINDOW *win, ctype ch );`
Add character ch to window win.

`int mvwaddch( WINDOW *win, int y, int x, ctype ch );`
Move to position (y, x), then add character ch to window win.

`int addstr( char *str );`
Call addch() with each character in string str. May be used with minicurses.
int mvaddstr( int y, int x, char *str );
    Move to position (y, x), then add string str.

int waddstr( WINDOW *win, char *str );
    Add string str to window win.

int mvwaddstr( WINDOW *win, int y, int x, char *str );
    Move to position (y, x), then add string str to window win.

int attroff( chtype attrs );
    Turn off attributes named in list attrs. May be used with minicurses.

int attron( chtype attrs );
    Turn on attributes named in list attrs. May be used with minicurses.

int attrset( chtype attrs );
    Set current attributes to those specified in list attrs. May be used with minicurses.

int baudrate ( void );
    Query current terminal speed. May be used with minicurses.

int beep ( void );
    Sound beep on terminal. May be used with minicurses.

int box( WINDOW *win, chtype vert, chtype hor );
    Draw a box around edges of window win. The vert and hor parameters are the characters to use for vertical and horizontal edges of the box.

int cbreak ( void );
    Set cbreak() mode. May be used with minicurses.

int nocbreak ( void );
    Unset cbreak() mode. May be used with minicurses.

int clear ( void );
    Clear stdscr.

int clearok( WINDOW *win, bool bool_flag );
    Clear screen before next redraw of window win if bool_flag is true.

int clrtoht ( void );
    Clear to bottom of stdscr.
int clrtoeol ( void );
  Clear to end of line on stdscr.

int delay_output( int ms );
  Insert pause of ms milliseconds in output. May be used with minicurses.

int nodelay( WINDOW *win, bool bool_flag );
  Enable nodelay() input mode through getch() on window win if bool_flag is true.

int delch ( void );
  Delete a character.

int deleteln ( void );
  Delete a line.

int delwin( WINDOW *win );
  Delete window win.

int douupdate ( void );
  Update screen from all wnoutrefresh().

int echo ( void );
  Set echo mode. May be used with minicurses.

int noecho ( void );
  Unset echo mode. May be used with minicurses.

int endwin ( void );
  End window mode. May be used with minicurses.

int erase ( void );
  Erase stdscr.

char erasechar ( void );
  Return user's erase character.

int fixterm ( void );
  Restore terminal to "in curses" state.

int flash ( void );
  Flash screen or beep.

int flushinp ( void );
  Throw away any data in type-ahead. May be used with minicurses.
int flushok (WINDOW *win, bool bool_flag);

Set the flush-on-refresh flag for window win to be bool_flag.

int getch (void);

Get a character from stdscr. May be used with minicurses. The following list contains the function keys that might be returned by the getch() function if keypad() has been enabled. Due to lack of definitions in terminfo, or due to the terminal not transmitting a unique code when the key is pressed, not all of these keys are supported.

KEY_BREAK
    Break key (unreliable)

KEY_DOWN
    Down arrow key

KEY_UP
    Up arrow key

KEY_LEFT
    Left arrow key

KEY_RIGHT
    Right arrow key

KEY_HOME
    Home key

KEY_BACKSPACE
    Backspace (unreliable)

KEY_F(n)
    Function key Fn, where n is an integer from 0 to 63

KEY_DL
    Delete line

KEY_IL
    Insert line

KEY_DC
    Delete character

KEY_IC
    Insert character or enter insert mode

KEY_EIC
    Exit insert character mode

KEY_CLEAR
    Clear screen

KEY_EOS
    Clear to end of screen

KEY_EOL
    Clear to end of line

KEY_SF
    Scroll one line forward

KEY_SR
    Scroll one line backwards (reverse)

KEY_NPAGE
    Next page

KEY_PPAGE
    Previous page
KEY_STAB Set tab
KEY_CTAB Clear tab

KEY_CATAB
Clear all tabs

KEY_ENTER Enter or send (unreliable)
KEY_SRESET Soft (partial) reset (unreliable)
KEY_RESET Reset or hard reset (unreliable)
KEY_PRINT Print or copy
KEY_LL Home down or bottom (lower left)
KEY_A1 Upper left key of keypad
KEY_A3 Upper right key of keypad
KEY_B2 Center key of keypad
KEY_C1 Lower left key of keypad
KEY_C3 Lower right key of keypad

char *getcap (char *cap_name);
Get terminal capability cap_name.

int getstr( char *str);
Get the string through stdscr.

int gettmode (void);
Get current tty modes.

int getyx( WINDOW *win, int y, int x);
Get (y, x) coordinates from window win.

bool has_ic ( void );
Has value of TRUE if terminal can insert character.

bool has_il ( void );
Has value of TRUE if terminal can insert line.

int idlok( WINDOW *win, bool bool_flag);
Use terminal's insert/delete line on window win if bool_flag is true.
May be used with minicurses.
chtype inch (void);
    Get character at current \((y, x)\) coordinates.

WINDOW *initscr (void);
    Initialize screens. May be used with minicurses.

int insch (chttype \(ch\));
    Insert character \(ch\).

int insertln (void);
    Insert a line.

int intrflush (WINDOW *win, bool bool_flag);
    Interrupt flush output on window \(win\) if \(bool\_flag\) is true.

int keypad (WINDOW *win, bool bool_flag);
    Enable keypad input on window \(win\) if \(bool\_flag\) is true.

char killchar (void);
    Return current user’s \texttt{kill()} character.

int leaveok (WINDOW *win, bool bool_flag);
    Permit cursor to be left anywhere after refresh for window \(win\) if \(bool\_flag\) is true; otherwise cursor must be left at current position.

char *longname (void);
    Return verbose name of terminal.

char *longname (char *termbuf, char *name);
    Set \textit{name} to the full name of the terminal described by \textit{termbuf}.
    Used in programs that are compiled with the \texttt{-DBSD} option to provide BSD compatibility.

char meta (WINDOW *win, bool bool_flag);
    Allow metacharacters on input from window \(win\) if \(bool\_flag\) is true.
    May be used with minicurses.

int move (int \(y\), int \(x\));
    Move to position \((y, x)\) on \texttt{stdscr}. May be used with minicurses.

int mvaddch (int \(y\), int \(x\), chtype \(ch\));
    Move to position \((y, x)\), then add character \(ch\).

char mvcur (int \(yl\), int \(xl\), int \(y2\), int \(x2\));
    Move cursor from current position \((yl, xl)\) to new position \((y2, x2)\).
int mvdelch( int y, int x);
Move to position (y, x), then delete a character.

int mvgetch( int y, int x);
Move to position (y, x), then get a character from the terminal.

int mvgetstr( int y, int x, char *str);
Move to position (y, x), then get the str string from the terminal.

chtype mvinch( int y, int x);
Move to position (y, x) then get the character at current (y, x) coordinates.

int mvinsch( int y, int x, chtype ch);
Move to position (y, x) then insert the character ch.

int mvprintw( int y, int x, char *fmt[, args]);
Move to position (y, x), then print on stdscr.

int mvscanw( int y, int x, char *fmt[, args]);
Move to position (y, x), then scan through stdscr.

int mvwdelch( WINDOW *win, int y, int x);
Move to position (y, x), then delete a character from win.

int mvwgetch( WINDOW *win, int y, int x);
Move to position (y, x), then get a character through win.

int mvwgetstr( WINDOW *win, int y, int x, char *str);
Move to position (y, x), then get a string through win.

int mvwin( WINDOW *win, int y, int x);
Move win so that the upper left corner is located at position (y, x).

chtype mvwinch( WINDOW *win, int y, int x);
Move to position (y, x) in win, then get the character at the new position.

int mvwinsch( WINDOW *win, int y, int x, chtype ch);
Move to position (y, x), then insert the character ch into win.

int mvwprintw( WINDOW *win, int y, int x, char *fmt[, args]);
Move to position (y, x) then printf() on stdscr.
int mvwscanw( WINDOW *win, int y, int x, char *fmt [, args ]);
    Move (y, x) then scanf() through stdscr.

WINDOW *newpad( int nlines, int ncols );
    Create a new pad with given dimensions.

SCREEN *newterm( char *type, FILE outfd, FILE infd );
    Set up new terminal of given type to output on outfd and input from infd.

WINDOW *newwin( int lines, int cols, int begin_y, int begin_x );
    Create a new window.

int nl ( void );
    Set new line mapping. May be used with minicurses.

int nonl ( void );
    Unset new line mapping. May be used with minicurses.

int overlay( WINDOW *win1, WINDOW *win2 );
    Overlay win1 on win2. The overlaying window (win1) takes as its origin the window being overlayed (win2).

int overwrite( WINDOW *win1, WINDOW *win2 );
    Overwrite win1 on win2.

int printw( char *fmt [, arg1, arg2, ... ] );
    Print on stdscr.

int raw ( void );
    Set raw mode. May be used with minicurses.

int refresh ( void );
    Make current screen look like stdscr. May be used with minicurses.

int prefresh( WINDOW *pad, int pminrow, int pmincol, int sminrow, int smincol,
    int smaxrow, int smaxcol );
    Refresh from pad starting with given upper left corner of pad with output to given portion of screen.
int pnotrefresh(WINDOW *pad, int pminrow, int pmincol,
        int sminrow,
        int smincol, int smaxrow, int smaxcol);

        Refresh like prefresh(), but with no output until doupdate() is called.

int noraw ( void );

        Unset raw mode. May be used with minicurses.

int resetterm ( void );

        Set tty modes to "out of curses" state. May be used with minicurses.

int resetty ( void );

        Reset tty flags to stored value. May be used with minicurses.

int saveterm ( void );

        Save current modes as "in curses" state. May be used with minicurses.

int savetty ( void );

        Store current tty flags. May be used with minicurses.

int scanw( char *fmt [, arg1, arg2, ... ]); 

        Scanf through stdscr.

int scroll( WINDOW *win );

        Scroll win one line.

int scrolllok( WINDOW *win, bool bool_flag );

        Allow terminal to scroll if bool_flag is true.

SCREEN *set_term( SCREEN *new );

        Enable talk to terminal new.

int setscrreg( int top, int bottom );

        Set user scrolling region to lines top through bottom.

void setterm( char *type );

        Establish terminal with a given type.

int standend ( void );

        Clear standout mode attribute. May be used with minicurses.
int standout ( void );
   Set standout mode attribute. May be used with minicurses.

WINDOW *subwin( WINDOW *win, int lines, int cols, int begin_y, int begin_x );
   Create a subwindow.

int touchline( WINDOW *win, int y, int firstcol, int numcol );
   Mark numcol columns, starting at column firstcol, of line y as changed.

int touchoverlap( WINDOW *win1, WINDOW *win2 );
   Mark overlap of win1 on win2 as changed.

int touchwin( WINDOW *win );
   Change all of win.

int traceoff ( void );
   Turn off debugging trace output.

int traceon ( void );
   Turn on debugging trace output.

int typeahead( FILE fd );
   Check file descriptor fd to check type-ahead.

char *unctrl( chtype ch );
   Use printable version of ch. May be used with minicurses.

int wattroff( WINDOW *win, int attrs );
   Turn off attrs in win.

int wattron( WINDOW *win, int attrs );
   Turn on attrs in win.

int wattrset( WINDOW *win, int attrs );
   Set attributes in win to attrs.

int wclear( WINDOW *win );
   Clear win.

int wclrtobot( WINDOW *win );
   Clear to bottom of win.

int wclrtoeol( WINDOW *win );
   Clear to end of line on win.
int wdelch( WINDOW *win );
    Delete the current character from win.
int wdeleteln( WINDOW *win );
    Delete line from win.
int werase( WINDOW *win );
    Erase win.
int wgetch( WINDOW *win );
    Get a character through win.
int wgetstr( WINDOW *win, char *str );
    Get the string str through win.
chttype winch( WINDOW *win );
    Get the character at current (y, x) in win.
int winsch( WINDOW *win, chtype ch );
    Insert the character ch into win.
int winsertln( WINDOW *win );
    Insert line into win.
int wmove( WINDOW *win, int y, int x );
    Set current (y, x) coordinates on win.
int wnoutrefresh( WINDOW *win );
    Refresh but no screen output.
int wprintw( WINDOW *win, char *fmt [, arg1, arg2, ... ] );
    printf() on win.
int wrefresh( WINDOW *win );
    Make screen look like win.
int wscanf( WINDOW *win, char *fmt [, arg1, arg2,... ] );
    scanf() through win.
int wsetscrreg( WINDOW *win, int top, int bottom );
    Set scrolling region of win.
int wstandend( WINDOW *win );

Clear standout attribute in win.

int wstandout( WINDOW *win );

Set standout attribute in win.

Terminfo Level Functions

These functions should be called by programs that have to deal directly with the terminfo database. Due to the low level of this interface, its use is discouraged.

To use the terminfo level functions of curses, include the curses.h and term.h files, in that order, to get the definitions for these strings, numbers, and flags. Programs should call the setupterm() function before using any of the other terminfo functions. The setupterm() function defines the set of terminal-dependent variables defined in the terminfo file.

All terminfo strings (including the output of the tparm() parameter) should be printed using the tputs() or putp() function. Before exiting, your program should call the reset_shell_mode() function to restore the tty modes. Programs desiring shell escapes can call the reset_shell_mode() function before the shell is called, and the reset_prog_mode() function after returning from the shell.

int delay_output ( int ms );

Sets the output delay, in milliseconds.

int def_prog_mode( void );

Saves the current terminal mode as program mode, in cur_term->Nttyb.

int def_shell_mode( void );

Saves the shell mode as normal mode, in cur_term->Ottyb. The def_shell_mode() function is called automatically by setupterm() function.

int putp( char *str );

Calls tputs( char *str, 1, putchar() ).

int reset_prog_mode ( void );

Puts the terminal into program mode.

int reset_shell_mode ( void );

Puts the terminal into shell mode. All programs must call the reset_shell_mode() function before they exit. The higher-level function endwin() automatically does this.
int setupterm( char *term, int fd, int rc );

Reads in the database. The term parameter is a character string that specifies the terminal name. If term is 0 (zero), then the value of the TERM environment variable is used. One of the following status values is stored into the integer pointed to by the rc parameter:

1   Successful completion.
0   No such terminal.
-1   An error occurred while locating the terminfo database.

If the rc parameter is 0 (zero), then no status value is returned, and an error causes the setupterm() function to print an error message and exit, rather than return. The fd parameter is the file descriptor of the terminal being used for output. The setupterm() function calls the TIOCGWINSZ ioctl function to determine the number of lines and columns on the display. If termdef cannot supply this information, then the setupterm() function uses the values in the terminfo database. The simplest call is setupterm(0,1,0), which uses all the defaults.

After the call to the setupterm() function, the global variable cur_term is set to point to the current structure of terminal capabilities. It is possible for a program to use more than one terminal at a time by calling the setupterm() function for each terminal and saving and restoring cur_term.

The setupterm() function also initializes the global variable ttytype as an array of characters to the value of the list of names for the terminal. The list comes from the beginning of the terminfo description.

char *tparm( char *format [, arg, . . . ]);

Instantiates the format string format, and one or more arguments of varying type. The character string returned has the given parameters applied.

void tputs( char *str, int affcnt, int (*putc) () );

Applies padding information to string str. The affcnt parameter is the number of lines affected, or 1 if not applicable. The putc parameter function is similar to putchar() to which the characters are passed one at a time.
Some strings are of a form similar to $<20>$, which is an instruction to pad for 20 milliseconds.

```c
void vidputs( int *attrs, int (*putc) () );
```

Outputs the string to put terminal in video attribute mode attrs. Characters are passed to the putc function. The attrs are defined in curses.h. The previous mode is retained by this function.

```c
void vidattr( int attrs );
```

Like vidputs(), but outputs through putchar().

**Termcap Compatibility Functions**

These functions are included for compatibility with programs that require termcap. Their parameters are the same as for termcap, and they are emulated using the terminfo database.

```c
int tgetent( char *bp, char *name );
```

Looks up the termcap entry for name. Both bp and name are strings. The name parameter is a terminal name; bp is ignored. Calls the setupterm() function.

```c
int tgetflag( char *id );
```

Returns the Boolean entry for id, which is a 2-character string that contains a termcap identifier.

```c
int tgetnum( char *id );
```

Returns the numeric entry for id, which is a 2-character string that contains a termcap identifier.

```c
char *tgetstr( char *id, char *area );
```

Returns the string entry for id, which is a 2-character string that contains a termcap identifier. The area parameter is ignored.

```c
char *tgoto( char *cap, int col, int row );
```

Applies parameters to the given cap. Calls the tparm() function.

```c
void tputs( char *cap, int affcnt, int (*fn) () );
```

Applies padding to cap calling fn as putchar().

**Related Information**

Files: terminfo(4)
cuserid

Purpose  Gets the alphanumeric username associated with the current process

Library   Standard I/O Package (libc.a)

Synopsis  
#include <stdio.h>
char *cuserid ( char *s );

Parameters

s            If the s parameter is a null pointer, the character string is stored into an internal static area, the address of which is returned. This internal static area is overwritten with the next call to cuserid().

            If the s parameter is not a null pointer, the character string is stored into the array pointed to by the s parameter. This array must contain at least L_cuserid bytes. L_cuserid is a constant defined in the stdio.h header file, and has a value greater than 0 (zero).

Description

The cuserid() function generates a character string representing the username of the owner of the current process.

Notes

AES Support Level: Full use

Return Values

If the s parameter is not a null pointer, the cuserid() function returns s. If the s parameter is not a null pointer and the username cannot be found, an empty string is returned.

If the s parameter is a null pointer and the username cannot be found, the cuserid() function returns a null pointer.
The reentrant version of `cuserid()` always returns null if the argument passed is null.

**Related Information**

Functions: `getlogin(2), getpwent(3)`
dbminit, fetch, store, delete, firstkey, nextkey, forder

**Purpose**
Database subroutines

**Library**
DBM Library (libdbm.a)

**Synopsis**
```c
#include <dbm.h>

typedef struct {
    char *dptr;
    int dsize;
} datum;

int dbminit(
    char *file);

datum fetch(
    datum key);

int store(
    datum key,
    datum content);

int delete(
    datum key);

datum firstkey( void );

datum nextkey(
    datum key);

long forder(
    datum key);
```

**Parameters**
- `file` Specifies the database file.
- `key` Specifies the key.
- `content` Specifies a value associated with the `key` parameter.

**Description**
The `dbminit()`, `fetch()`, `store()`, `delete()`, `firstkey()`, `nextkey()`, and `forder()` functions maintain key/content pairs in a database. They are obtained with the
-ldbms loader option. The dbm library is provided only for backwards compatibility, having been obsoleted by the ndbm functions in libc. See the manual page for ndbm for more information.

The dbminit(), fetch(), store(), delete(), firstkey(), nextkey(), and forder() functions handle very large databases (up to a billion blocks) and access a keyed item in one or two file system accesses. Arbitrary binary data, as well as normal ASCII strings, are allowed.

The database is stored in two files. One file is a directory containing a bit map and has .dir as its suffix. The second file contains all data and has .pag as its suffix.

Before a database can be accessed, it must be opened by the dbminit() function. At the time that dbminit() is called, the file .dir and file .pag files must exist. (An empty database is created by creating zero-length .dir and .pag files.)

Once open, the data stored under a key is accessed by the fetch() function and data is placed under a key by the store() function. A key (and its associated contents) is deleted by the delete() function. A linear pass through all keys in a database may be made by use of the firstkey() and nextkey() functions. The firstkey() function returns the first key in the database. With any key, the nextkey() function returns the next key in the database. The following code traverses the database:

for (key = firstkey(); key.dptr != NULL; key = nextkey(key))

Return Values

Upon successful completion, the functions that return an int return 0 (zero). Otherwise, a negative number is returned. The functions that return a datum indicate errors with a null (0) dptr.

Related Information

Functions: ndbm(3)
Functions

decode_mach_o_hdr(3)

decode_mach_o_hdr

Purpose
Converts the canonical header from an OSF/ROSE object file to readable form

Library
libId

Synopsis
#include <mach_o_header.h>
#include <sys/types.h>
int decode_mach_o_hdr(
    void *in_bufp,
    size_t in_bufsize,
    unsigned long hdr_version,
    mo_header_t *headerp);

Parameters

  in_bufp
  Specifies the address of a buffer that contains the object file's header in canonical form.

  in_bufsize
  Specifies the size of the input buffer in bytes. The number of bytes read from the file into the buffer by the caller should not be less than MO_SIZEOF_RAW_HDR, as defined in the mach_o_header.h file.

  hdr_version
  Specifies the version of the header that corresponds to the structure pointed to by headerp.

  headerp
  Specifies the address of the header structure to receive the header translated into native, readable form.

Description

The decode_mach_o_hdr() function converts an OSF/ROSE object file header from its canonical form in the object file to a form that can be read "naturally" in the local environment. "Natural" means with fields aligned to fit the mo_header_t structure defined in the mach_o_header.h header file, as accessed by code generated by the local C compiler. "Canonical" means with fields aligned for 32-bit words and in network byte order, described for the local machine in the mach_o_header_md.h header file.
Since object file headers can change only by growing, any header version that is supported by `decode_mach_o_hdr()` can be given as input or output. The input and output versions can be different.

**Notes**

The caller should make sure that it supports both the header version and object file version returned. In general, callers should not have to check for version numbers greater than those they recognize.

If an error is returned, the contents of the output structure are undefined.

**Return Values**

Upon successful completion, the `decode_mach_o_hdr()` function returns `MO_HDR_CONV_SUCCESS` and stores the converted header in `headerp`. Otherwise, one or more of the following errors is returned:

- **MO_ERROR_BAD_RAW_HDR_VERS**
  - The header version in the input buffer was not recognized.

- **MO_ERROR_BAD_HDR_VERS**
  - The header version specified for `headerp` was not recognized.

- **MO_ERROR_BUF2SML**
  - The size of the input buffer was too small.

- **MO_ERROR_BAD_MAGIC**
  - The input buffer did not contain the OSF/ROSE magic number in the correct location.

- **MO_ERROR_UNSUPPORTED_VERS**
  - Either the version of the header in the object file or the version specified for the output could not be converted, even though both are legal according to the header files.

**Related Information**

Functions: `encode_mach_o_hdr(3)`

Files: `osf_rose(4)`
**dn_comp**

**Purpose**
Compresses a domain name

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

int dn_comp (
    u_char *expanded_name,
    u_char *compressed_name,
    int length,
    u_char **name_ptrs,
    u_char **end_ptr);
```

**Parameters**
- `expanded_name` Points to a domain name.
- `compressed_name` Points to an array containing the compressed domain name.
- `length` Specifies the size of the array pointed to by the `compressed_name` parameter.
- `name_ptrs` Specifies a list of pointers to previously compressed names in the current message.
- `end_ptr` Points to the end of the array pointed to by the `compressed_name` parameter.

**Description**
The **dn_comp**() (domain name compression) function compresses the domain name pointed to by the `expanded_name` parameter and stores it in the area pointed to by the `compressed_name` parameter.
The `dn_comp()` function inserts labels into the message as the name is compressed. The `dn_comp()` function also maintains a list of pointers to the message labels.

If the value of the `name_ptrs` parameter is null, the `dn_comp()` function does not compress any names, but instead translates a domain name from ASCII to internal format without removing suffixes (compressing). Otherwise, the `name_ptrs` parameter is the address of pointers to previously compressed suffixes.

If the `end_ptr` parameter is null, the `dn_comp()` function does not update the list of label pointers.

The `dn_comp()` function is one of a set of subroutines that form the resolver, a set of functions that resolves domain names. Global information that is used by the resolver functions is kept in the `_res` data structure. The `/include/resolv.h` file contains the `_res` data structure definition.

**Return Values**

Upon successful completion, the `dn_comp()` function returns the size of the compressed domain name. Otherwise, a value of -1 is returned.

**Files**

`/etc/resolv.conf`

Defines name server and domain name structures, constants, and values.

**Related Information**


Commands: `named(8)`
Functions

dn_expand(3)

**dn_expand**

**Purpose**
Expands a compressed domain name

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

int dn_expand(
    u_char *message_ptr,
    u_char *end_of_message,
    u_char *compressed_name,
    u_char *expanded_name,
    int length);
```

**Parameters**

- `message_ptr`  Specifies a pointer to the beginning of a message.
- `end_of_message`  Points to the end of the original message that contains the compressed domain name.
- `compressed_name`  Specifies a pointer to a compressed domain name.
- `expanded_name`  Specifies a pointer to a buffer that holds the resulting expanded domain name.
- `length`  Specifies the size of the buffer pointed to by the `expanded_name` parameter.

**Description**
The `dn_expand()` function expands a compressed domain name to a full domain name, converting the expanded names to uppercase.
The `dn_expand()` function is one of a set of subroutines that form the resolver, a set of functions that resolves domain names. Global information that is used by the resolver functions is kept in the `_res` data structure. The `/include/resolv.h` file contains the `_res` structure definition.

**Return Values**

Upon successful completion, the `dn_expand()` function returns the size of the expanded domain name. Otherwise, a value of -1 is returned.

**Files**

`/etc/resolv.conf`

Defines name server and domain name constants, structures, and values.

**Related Information**

Functions: `res_init(3)`, `res_mkquery(3)`, `res_send(3)`, `dn_comp(3)`, `dn_find(3)`, `getshort(3)`, `getlong(3)`, `putshort(3)`, `putlong(3)`, `dn_skipname(3)`
dn_find

**Purpose**
Searches for an expanded domain name

**Library**
Standard C Library (*libc.a*)

**Synopsis**
```c
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
dn_find()(
    char *exp_domain_name,
    char *message,
    char **domain_names,
    char **end_ptr);
```

**Parameters**

`exp_domain_name`
Points to an expanded domain name.

`message`
Points to the address of a domain name message that contains the name sought by the `dn_find()` function.

`domain_names`
Specifies an array of pointers to previously compressed names in the current message.

`end_ptr`
Points to the end of an array of pointers. The array is indicated by the `domain_names` parameter.

**Description**
The `dn_find()` (domain name find) function searches for an expanded domain name from a list of previously compressed names. An application program calls the `dn_find()` function indirectly using the `dn_comp()` function. If an expanded domain name is found, the `dn_comp()` function returns the offset from the `message` parameter.
The `dn_find()` function is one of a set of subroutines that form the resolver, a set of functions that resolves domain names. Global information used by the resolver functions resides in the `_res` data structure. The `include/resolv.h` file contains the `_res` data structure definition.

Return Values

Upon successful completion, the `dn_find()` function returns the offset from the `message` parameter. Otherwise, the `dn_find()` function returns a value of -1.

Files

/etc/resolv.conf

Defines name server and domain name structures and constants.

Related Information

Functions: `res_init(3)`, `res_mkquery(3)`, `res_send(3)`, `dn_comp(3)`, `dn_expand(3)`, `getshort(3)`, `getlong(3)`, `putshort(3)`, `putlong(3)`

Commands: `named(8)`
dn_skipname

Purpose
Skips over a compressed domain name

Library
Standard C Library (libc.a)

Synopsis
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

int dn_skipname(
    u_char *comp_domain_name,
    u_char *end_of_message);

Parameters

comp_domain_name
    Specifies a pointer to a compressed domain name.

dn_skipname
    Specifies the end of the compressed domain name address.

Description

The dn_skipname() function skips over a compressed domain name.

The dn_skipname() function is one of a set of subroutines that form the resolver, a
set of functions that resolve domain names. Global information that is used by the
resolver functions is kept in the _res data structure. The include/resolv.h file
contains the _res structure definition.

Return Values

Upon successful completion, the dn_skipname() function returns the size of the
compressed domain name. If the dn_skipname() function fails, -1 is returned.
Files

/etc/resolv.conf
Defines name server and domain name structures, values, and constants.

Related Information

Functions: res_init(3), res_mkquery(3), res_send(3), dn_comp(3),
dn_expand(3), dn_find(3), getshort(3), getlong(3), putshort(3), putlong(3)
Commands: named(8)
### Functions

**drand48(3)**

**drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48**

**Purpose**
Generates uniformly distributed pseudo-random number sequences

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <stdlib.h>

double drand48 ( void );
double erand48 ( unsigned short xsubi[3] );
long jrand48 ( unsigned short xsubi[3] );
void lcong48 ( unsigned short param[7] );
long lrand48 ( void );
long mrand48 ( void );
long nrand48 ( unsigned short xsubi[3] );
unsigned short *seed48 ( unsigned short seed_l6v[3] );
void srand48 ( long seed_val);
```

**Parameters**

- **xsubi**
  Specifies an array of three shorts, which, when concatenated together, form a 48-bit integer.

- **seed_val**
  Specifies the initialization value to begin randomization. Changing this value changes the randomization pattern.
**seed_16v**

Specifies another seed value; an array of three unsigned shorts that form a 48-bit seed value.

**param**

Specifies an array specifying the initial $X_i$, the multiplier value $a$, and the addend value $c$.

**Description**

This family of functions generates pseudo-random numbers using the linear congruential algorithm and 48-bit integer arithmetic.

The **drand48()** and **erand48()** functions return nonnegative, double-precision, floating-point values uniformly distributed over the range of $y$ values such that $0 \leq y < 1.0$.

The **lrand48()** and **nrand48()** functions return nonnegative long integers uniformly distributed over the range of $y$ values such that $0 \leq y < 2^{31}$.

The **mrand48()** and **jrand48()** functions return signed long integers uniformly distributed over the range of $y$ values such that $-2^{31} \leq y < 2^{31}$.

The **srand48()**, **seed48()**, and **lcong48()** functions initialize the random-number generator. Programs should invoke one of them before calling the **drand48()**, **lrand48()**, or the **mrand48()** functions. (Although it is not recommended practice, constant default initializer values are supplied automatically if the **drand48()**, **lrand48()**, or **mrand48()** functions are called without first calling an initialization function.) The **erand48()**, **nrand48()**, and **jrand48()** functions do not require that an initialization function be called first.

All the functions work by generating a sequence of 48-bit integer values, $X_i$, according to the linear congruential formula:

$$X_{n+1} = (aX_n + c) \mod m \quad n \geq 0$$

The parameter $m$ equals $2^{48}$; hence 48-bit integer arithmetic is performed. Unless **lcong48()** has been invoked, the multiplier value $a$ and the addend value $c$ are given by

$$a = 5DEECE66_{16} = 273673163155_{8}$$
$$c = B_{16} = 13_{8}$$

The values returned by the **drand48()**, **erand48()**, **lrand48()**, **nrand48()**, **mrand48()**, and **jrand48()** functions are computed by first generating the next 48-bit $X_i$ in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (most significant) bits of $X_i$ and transformed into the returned value.
The `drand48()`, `lrand48()`, and `mrand48()` functions store the last 48-bit $X_i$ generated into an internal buffer, which is why they must be initialized prior to being invoked.

The `erand48()`, `nrand48()`, and `jrand48()` functions require that the calling program provide storage for the successive $X_i$ values in the array pointed to by the `xsubi` parameter. This is why these routines do not have to be initialized; the calling program merely has to place the desired initial value of $X_i$ into the array and pass it as a parameter.

By using different parameters, the `erand48()`, `nrand48()`, and `jrand48()` functions allow separate modules of a large program to generate several independent sequences of pseudo-random numbers, that is, the sequence of numbers that one module generates does not depend upon how many times the functions are called by other modules.

The initializer function `srand48()` sets the high-order 32 bits of $X_i$ to the `LONG_BIT` bits contained in its parameter. The low order 16 bits of $X_i$ are set to the arbitrary value $330E_{16}$.

The initializer function `seed48()` sets the value of $X_i$ to the 48-bit value specified in the array pointed to by the `seed_16v` parameter. In addition, `seed48()` returns a pointer to a 48-bit internal buffer that contains the previous value of $X_i$ that is used only by `seed48()`. The returned pointer allows you to restart the pseudo-random sequence at a given point. Use the pointer to copy the previous $X_i$ value into a temporary array. To resume where the original sequence left off, you can call `seed48()` with a pointer to this array.

The `lcong48()` function specifies the initial $X_i$ value, the multiplier value $a$, and the addend value $c$. The `param` array elements `param[0-2]` specify $X_i$, `param[3-5]` specify the multiplier $a$, and `param[6]` specifies the 16-bit addend $c$. After `lcong48()` has been called, a subsequent call to either `srand48()` or `seed48()` restores the standard $a$ and $c$ as specified previously.

Notes

AES Support Level: Trial use
Return Values

The `drand48()` and `erand48()` functions return nonnegative, double-precision, floating-point values. The `lrand48()` and `nrand48()` functions return signed long integers uniformly distributed over the range $0 \leq y < 2^{31}$. The `mrand48()` and `jrand48()` functions return signed long integers uniformly distributed over the range $-2^{31} \leq y < 2^{31}$.

The `seed48()` function returns a pointer to a 48-bit internal buffer.

The `lcong48()` and `srand48()` functions do not return a value.

Related Information

Functions: `rand(3)`, `rand(3)`, `random(3)`
ecvt, fcvt, gcvt

Purpose
Converts a floating-point number to a string.

Library
Standard C Library (libc.a)

Synopsis
#include <stdlib.h>

char *ecvt (double value,
            int num_digits,
            int *decimal_ptr,
            int *sign);

char *fcvt (double value,
            int num_digits,
            int *decimal_ptr,
            int *sign);

char *gcvt (double value,
            int num_digits,
            char *buffer);

Parameters

value  Specifies the double value to be converted.
num_digits  Specifies the number of digits in the resulting string.
decimal_ptr  Holds the position of the decimal point relative to the beginning of the string. A negative number means the decimal point is to the left of the digits given in the string.
sign  Holds 0 (zero) if the value is positive or zero, and a nonzero value if it is negative.
bUFFER  Specifies the character array for the resulting string.
ecvt(3)

Description

The ecvt(), fcvt(), and gcvt() functions convert floating-point numbers to null-terminated strings.

The ecvt() function converts the value specified by the value parameter to a null-terminated string of length num_digits, and returns a pointer to it. The resulting low-order digit is rounded according to the current rounding mode. The decimal_ptr parameter is assigned to the position of the decimal point relative to the position of the string. The sign parameter is assigned 0 (zero) if value is positive or zero, and a nonzero value if value is negative. The decimal point and sign are not included in the string.

The fcvt() function is the same as the ecvt() function, except that it rounds to the correct digit for outputting num_digits digits in C or FORTRAN F-format. In the F-format, num_digits is taken as the number of digits desired after the decimal point.

The gcvt() function converts the value specified by the value parameter to a null-terminated string, stores it in the array pointed to by the buffer parameter, and then returns buffer. The gcvt() function attempts to produce a string of num_digits significant digits in FORTRAN F-format. If this is not possible, then E-format is used. The string is ready for printing, complete with minus sign, decimal point, or exponent, as appropriate. Trailing zeros are suppressed.

Notes

In the F-format, num_digits is the number of digits desired after the decimal point. Very large numbers produce a very long string of digits before the decimal point, and then num_digits digits after the decimal point. For large numbers, it is preferable to use the gcvt() or ecvt() function so that the E-format will be used.

The ecvt(), fcvt(), and gcvt() functions represent the following special values that are specified in ANSI/IEEE Std. 754-1985 for floating-point arithmetic:

<table>
<thead>
<tr>
<th>Quiet NaN</th>
<th>NaNQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>signalling NaN</td>
<td>NaNS</td>
</tr>
<tr>
<td>+</td>
<td>Infinity</td>
</tr>
</tbody>
</table>

The sign associated with each of these values is stored into the sign parameter. Note, also, that in IEEE Floating Point, a value of 0 (zero) can be positive or negative, as set by the sign parameter.
Caution

All three functions store the strings in a static area of memory whose contents are overwritten each time one of the functions is called.

Related Information

Functions: \texttt{atof(3)}, \texttt{printf(3)}, \texttt{scanf(3)}
**Purpose**
Converts an OSF/ROSE object file header from native, readable form to canonical form

**Library**
libld

**Synopsis**
```c
#include <mach_o_header.h>
#include <sys/types.h>
int encode_mach_o_hdr(
    mo_header_t *headerp;
    void *out_bufp;
    size_t out_bufsize;
)
```

**Parameters**
- **headerp**: Specifies the address of a structure containing an OSF/ROSE object file header in native, readable form. The structure should be completely filled in, including the header version number.
- **out_bufp**: Specifies the address of a buffer to receive the header translated into canonical form.
- **out_bufsize**: Specifies the size of the output buffer in bytes. The size should be MO_SIZEOF_RAW_HDR, which is defined in the `mach_o_header_md.h` header file.

**Description**
The `encode_mach_o_hdr()` function converts an OSF/ROSE object file header from a form that can be read "naturally" in the local environment into its corresponding canonical form. "Natural" means with fields aligned to fit the `mo_header_t` structure defined in the `mach_o_header.h` header file, as accessed by code generated by the local C compiler. "Canonical" means with fields aligned for 32-bit words and in network byte order, described for the local machine in the `mach_o_header_md.h` header file.
Functions

encode_mach_o_hdr(3)

Notes

If an error is returned, the contents of the output buffer are undefined.

Return Values

Upon successful completion, the encode_mach_o_hdr() function returns MO_HDR_CONV_SUCCESS and stores the converted header in out_bufp. Otherwise, one or more of the following errors is returned:

- **MO_ERROR_BAD_HDR_VERS**
  - The header version in the input structure was not recognized.

- **MO_ERROR_BUF2SML**
  - The size of the output buffer was too small.

- **MO_ERROR_BAD_MAGIC**
  - The magic number in the input structure was not the OSF/ROSE magic number.

- **MO_ERROR_OLD_RAW_HDR_FILE**
  - The header version in the input structure did not have a corresponding description in canonical form. In other words, the header file for the canonical form does not describe as many fields as the input structure does.

- **MO_ERROR_UNSUPPORTED_VERS**
  - The version of the header in the input structure could not be converted, even though it is legal according to the header files. The reason is that encode_mach_o_hdr() has not been updated to support this version of the header.

Related Information

- Functions: decode_mach_o_hdr(3)
- Files: osf_rose(4)
endhostent

Purpose

Ends retrieval of network host entries

Library

Standard C Library (libc.a)

Synopsis

#include <netdb.h>

void endhostent ( void );

Description

The endhostent() function closes the /etc/hosts file, previously opened with the gethostentbyaddr() or gethostentbyname() function.

Notes

If the most recent sethostent() function has been performed with a nonzero parameter, then the endhostent() function will not close the /etc/hosts file. In this instance, the /etc/hosts file is not closed until a call to the exit() function. A second sethostent() function must be issued with a parameter equal to 0 (zero) in order to ensure that a following endhostent() function will succeed.

Files

/etc/hosts Contains the hostname database.

Related Information

Functions: gethostbyaddr(3), gethostbyname(3), gethostent(3)
endnetent

Purpose  Closes the networks file

Library  Standard C Library (libc.a)

Synopsis  #include <netdb.h>
void endnetent ( void );

Description

The endnetent( ) function closes the /etc/networks file, previously opened with
the getnetent( ), getnetbyaddr( ), setnetent( ) or getnetbyname( ) function.

Notes

If the most recent setnetent() function has been performed with a nonzero
parameter, then the endnetent( ) function will not close the /etc/networks file. In
this instance, the /etc/networks file is not closed until a call to the exit() function.
A second setnetent() function must be issued with a parameter equal to 0 (zero) in
order to ensure that a following endnetent( ) function will succeed.

Files

/etc/networks
Contains official network names.

Related Information

Functions: getnetent(3), getnetbyaddr(3), getnetbyname(3), setnetent(3)
endprotoent

**Purpose**  Closes the /etc/protocols file

**Library**  Standard C Library (libc.a)

**Synopsis**  void endprotoent ( void );

**Description**  The `endprotoent( )` function closes the /etc/protocols file, previously opened with the `getprotoent( )`, `getprotobynumber( )`, or `getprotobyname( )` function.

**Notes**  If the most recent `setprotoent( )` function has been performed with a nonzero parameter, then the `endprotoent( )` function will not close the /etc/protocols file. In this instance, the /etc/protocols file is not closed until a call to the `exit( )` function. A second `setprotoent( )` call must be issued with a parameter equal to 0 (zero) in order to ensure that a following `endprotoent( )` function will succeed.

**Files**  

/etc/protocols  
Contains protocol names.

**Related Information**  
Functions:  `getprotoent(3)`, `getprotobynumber(3)`, `getprotobyname(3)`, `setprotoent(3)`
endservent

Purpose
Closes the /etc/services file entry

Library
Standard C Library (libc.a)

Synopsis
#include <netdb.h>
void endservent ( void );

Description
The endservent( ) function closes the /etc/services file, previously opened with the
getservent( ), getservbyname( ), or getsrvbyport function.

Notes
If the most recent setservent( ) function has been performed with a nonzero parameter, then the endservent( ) function will not close the /etc/services file. In this instance, the /etc/services file is not closed until a call to the exit( ) function. A second setservent( ) function must be issued with a parameter equal to 0 (zero) in order to ensure that a following endservent( ) function will succeed.

Files
/etc/services Contains service names.

Related Information
erf, erfc

**Purpose**
Computes the error and complementary error functions

**Library**
Math Library (libm.a)

**Synopsis**
```c
#include <math.h>

double erf (double x);

double erfc (double x);
```

**Parameters**

- **x**
  Specifies some double value.

**Description**
The **erf**() function computes the error function of **x**, defined as:

\[
\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt
\]

The **erfc**() function computes \(1.0 - \text{erf}(x)\).

The **erfc**() function is provided because of the significant loss of relative accuracy if **erf**(x) is called for large values of x and the result is subtracted from 1.0. For example, 12 decimal places are lost when calculating \((1.0 - \text{erf}(5))\).

**Notes**

- The **erf**() and **erfc**() functions are supported for multi-threaded applications.

  **AES Support Level:** Trial use
Return Values

Upon successful completion, the \texttt{erf()} and \texttt{erfc()} functions return the value of the error function and complementary error function, respectively. If \texttt{x} is NaN, NaN is returned. Otherwise, \texttt{errno} is set to indicate the error or NaN is returned.

Errors

If the \texttt{erf()} or \texttt{erfc()} function fails, \texttt{errno} may be set to the following value:

\texttt{[EDOM]} The value of \texttt{x} is NaN.

Related Information

Functions: \texttt{exp(3)}, \texttt{isnan(3)}
environ, execl, execv, execle, execve, execlp, execvp

**Purpose**  
Executes a file

**Library**  
Standard C Library (libc.a): execlp(), execvp()

**Synopsis**  
extern char **environ;

int execl (  
    const char *path,  
    const char *arg,  
    ...);

int execv (  
    const char *path,  
    char * const argv[]);

int execle (  
    const char *path,  
    const char *arg,  
    ...  
    char * const envp[]);

int execve (  
    const char *path,  
    char * const argv[],  
    char * const envp[]);

int execlp (  
    const char *file,  
    const char *arg,  
    ...);

int execvp (  
    const char *file,  
    char * const argv[]);

**Parameters**

*path*  
Points to a pathname identifying the new process image file.

*arg*  
Specifies a character pointer to null-terminated strings.

*argv*  
Specifies an array of character pointers to null-terminated strings.
Functions

exec(2)

envp
Specifies an array of character pointers to null-terminated strings, constituting the environment for the new process.

file
Identifies the new process image file.

Description

The exec functions replace the current process image with a new process image. The new image is constructed from a regular executable file, called a new process image file. A successful exec does not return, because the calling process image is overlaid by the new process image.

When a program is executed as a result of an exec call, it is entered as a function call as follows:

```c
int main (  
    int argc,  
    char **argv
);
```

Here, argc is the argument count and argv[ ] is an array of character pointers to the arguments themselves. In addition, the following variable is initialized as a pointer to an array of character pointers to the environment strings:

```c
extern char **environ;
```

The argv and environ arrays are each terminated by a null pointer. The null pointer terminating the argv array is not counted in argc.

The arguments specified by a program with one of the exec functions are passed on to the new process image in the corresponding arguments to main().

The path argument points to a pathname that identifies the new process image file. The file argument is used to construct a pathname that identifies the new process image file. If the file argument contains a slash character, the file argument is used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed as the PATH environment variable.

The new process image file is formatted as an executable text or binary file, in one of the formats recognized by the exec functions. An executable text file is identified by a header line with the following syntax:

```text
#!/ interpreter_name [ optional_string ]
```

The #! identifies the file as an executable text file. The new process image is constructed from the process image file named by the interpreter_name string. The arguments are modified as follows:

- argv[0] is set to the name of the interpreter.
- If the optional_string is present, argv[1] is set to the optional_string.

1–137
The next element of `argv[ ]` is set to the original value of `path`.

The remaining elements of `argv[ ]` are set to the original elements of `argv[ ]`, starting at `argv[1]`. The original `argv[0]` is discarded.

An executable binary file can be loaded either directly by the `exec` function, or indirectly by the program loader. The `exec` function chooses to use direct or indirect loading based on the contents of the new process image file. For example, indirect loading might be used if the new process image file has unresolved symbols, requiring use of a shared library.

When indirect loading is used, the new process image is constructed from the default program loader, `/sbin/loader`, in the same manner as described for the `exec_with_loader( )` function. The default program loader is then responsible for completing the new program image by loading the new process image file and any shared libraries on which it depends.

If the process image file is not a valid executable object, the `execlp( )` and `execvp( )` functions use the contents of that file as standard input to a command interpreter conforming to the `system( )` function. In this case, the command interpreter becomes the new process image.

The `argv` argument is an array of character pointers to null-terminated strings. The last member of this array is a null pointer. These strings constitute the argument list available to the new process image. The value in `argv[0]` should point to a filename that is associated with the process being started by one of the `exec` functions.

The `const char *arg` and subsequent ellipses in the `execl( )`, `execlp( )`, and `execle( )` functions can be thought of as `arg0, arg1, ..., argn`. Together they describe a list of one or more pointers to null-terminated character strings that represent the argument list available to the new program. The first argument must point to a filename that is associated with the process being started by one of the `exec` functions, and the last argument must be a null pointer. For the `execle( )` function, the environment is provided by following the null pointer that will terminate the list of arguments in the parameter list to `execle( )` with an additional parameter as if it were declared as:

```
char * const envp []
```

The `envp` argument to `execve( )`, and the final argument to `execle( )`, name an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The environment array is terminated with a null pointer.

For those forms not containing an `envp` pointer (`execl( )`, `execv( )`, `execlp( )` and `execvp( )`) the environment for the new process image is taken from the external variable `environ` in the calling process.
The number of bytes available for the new process’ combined argument and environment lists is ARG_MAX. ARG_MAX includes the null terminators on the strings; it does not include the pointers.

File descriptors open in the calling process image remain open in the new process image, except for those whose close-on-exec flag FD_CLOEXEC is set (see the _f cntl() function). For those file descriptors that remain open, all attributes of the open file description, including file locks, remain unchanged.

The state of directory streams and message catalog descriptors in the new process image is undefined.

Each mapped file and shared memory region created with the _mmap() function is unmapped by a successful call to any of the exec functions, except those regions mapped with the MAP_INHERIT option. Regions mapped with the MAP_INHERIT option remain mapped in the new process image.

Signals set to the default action (SIG_DFL) in the calling process image are set to the default action in the new process image. Signals set to be ignored (SIG_IGN) by the calling process image are set to be ignored by the new process image. Signals set to be caught by the calling process image are set to the default action in the new process image (see the signal.h header file).

If the set user ID mode bit of the new process image file is set (see the _chmod() function), the effective user ID of the new process image is set to the owner ID of the new process image file. Similarly, if the set group ID mode bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID, real group ID, and supplementary group IDs of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved set user ID and the saved set group ID) for use by the _setuid() function.

The following attributes of the calling process image are unchanged after successful completion of any of the exec functions:

- Process ID
- Parent process ID
- Process group ID
- Session membership
- Real user ID
- Real group ID
- Supplementary group IDs
- Time left until an alarm clock signal (see the _alarm() function)
exec(2)

- Current working directory
- Root directory
- File mode creation mask (see the umask() function)
- Process signal mask (see the sigprocmask() function)
- Pending signals (see the sigpending() function)
- The tms_utime, tms_stime, tms_cutime, and tms_cstime fields of the tms structure.
- File size limit (see the ulimit() function)
- Nice value (see the nice() function)

Upon successful completion, the exec functions mark for update the st_atime field of the file.

Notes

AES Support Level: Full use

Return Value

If one of the exec functions returns to the calling process image, an error has occurred; the return value is -1, and errno is set to indicate the error.

Errors

If the exec functions fail, errno may be set to one of the following values:

[E2BIG] The number of bytes used by the new process image's argument list and environment list is greater than the system-imposed limit of ARG_MAX bytes.

[EACCES] Search permission is denied for a directory listed in the new process image file's path prefix, or the new process image file denies execution permission, or the new process image file is not a regular file and the implementation does not support execution of files of its type.

[ENAMETOOLONG] The length of the path or file arguments, or an element of the environment variable PATH prefixed to a file, exceeds PATH_MAX, or a pathname component is longer than NAME_MAX and _POSIX_NO_TRUNC is in effect for that file.

[ENOENT] One or more components of the new process image file's pathname do not exist, or the path or file argument points to an empty string.
[ENOTDIR] A component of the new process image file's path prefix is not a directory.

[EFAULT] The path argument is an invalid address.

[ELOOP] Too many symbolic links were encountered in pathname resolution.

[ENOMEM] Insufficient memory is available.

[ETXTBSY] The new process image file is currently open for writing by some process.

If the exec(), execv(), execl(), or execve() function fails, errno may be set to the following value:

[ENOEXEC] The new process image file has the appropriate access permission but is not in the proper format.

Related Information

Functions: alarm(3), exit(2), fcntl(2), fork(2), getenv(3), nice(3), putenv(3), sigaction(2), system(3), times(3), ulimit(3), umask(2), mmap(2), exec_with_loader(2)
exec_with_loader

Purpose
Executes a file with a loader.

Synopsis
```c
int exec_with_loader (  
    int flags,  
    const char *loader,  
    const char *file,  
    char * const argv[ ],  
    char * const envp[ ]);
```

Parameters
- `flags`: Specifies flags to be passed to the loader.
- `loader`: Points to a pathname that identifies a regular, executable, process image file that contains the loader.
- `file`: Points to a pathname that identifies a regular, executable process image file.
- `argv`: Specifies an array of character pointers to null-terminated strings.
- `envp`: Specifies an array of character pointers to null-terminated strings, constituting the environment for the new process.

Description
The `exec_with_loader()` function replaces the current process image with a new process image, in a manner similar to what the `exec` functions do. Both the `loader` parameter and the `file` parameter point to pathnames that identify regular, executable files called new process image files. Whereas the `exec` functions construct the new process image from the file identified by the `file` parameter, `exec_with_loader()` instead constructs the new process image from the file identified by the `loader` parameter. Throughout this manual page, the regular, executable, process image file specified by the `loader` parameter is referred to as the program loader, and the regular, executable, process image file specified by the `file` parameter is referred to as the file.
Once the `exec_with_loader()` function successfully loads the program loader, it transfers control to the program loader and effectively passes the `file` parameter on to the loader. Under normal usage, the program loader will then load (that is, merge) the file into the newly constructed process image, along with any object files upon which the program (that is, the file) depends. The typical use of `exec_with_loader()` is to load programs that contain unresolved external references, for example, programs that require the use of a shared library.

The `exec_with_loader()` function implements and preserves all of the semantics of the `exec` functions, with respect to the file. These include the handling of the `argv` and `envp` parameters, command interpreters, close-on-exec processing, signals, set user ID and set group ID processing, the process attributes and error returns.

The `loader` parameter may be null, in which case the `exec_with_loader()` function loads the default program loader, found in the `/sbin/loader` file. The `exec_with_loader()` function always loads the default program loader, even if the `loader` parameter points to a valid loader file, if the set user ID mode bit of the file is set (see the `chmod()` function) and the owner ID of the file is not equal to the effective user ID of the process, or if the set group ID mode bit of the file is set and the group ID of the file is not equal to the effective group ID of the process. The setting of the set user ID or set group ID mode bits on the loader have no effect whatsoever.

**Return Values**

If the `exec_with_loader()` function returns to the calling process image, an error has occurred; the return value is -1, and `errno` is set to indicate the error.

**Errors**

If the `exec_with_loader()` function fails, `errno` may be set to one of the following values:

[EACCES] Search permission is denied for a directory listed in either file’s path prefix, or either file denies execution permission, or either file is not a regular file and the implementation does not support execution of files of its type. Note that the `exec_with_loader()` function references two files, one specified by the `loader` parameter and one specified by the `file` parameter.

[ENAMETOOLONG] The length of the `loader` or `file` parameters exceeds PATH_MAX, or a pathname component is longer than NAME_MAX and _POSIX_NO_TRUNC is in effect for that file.
exec_with_loader(2)

[ENOENT] One or more components of either file’s pathname does not exist, or the loader or file parameter points to an empty string. Note that the exec_with_loader( ) function references two files, one specified by the loader parameter and one specified by the file parameter.

[ENOTDIR] A component of either file’s path prefix is not a directory. Note that the exec_with_loader( ) function references two files, one specified by the loader parameter and one specified by the file parameter.

[ENOEXEC] The file specified by the loader parameter has the appropriate access permission but is not in the proper format.

[EFAULT] The loader or file parameter is an invalid address.

[ELOOP] Too many symbolic links were encountered in pathname resolution.

[ENOMEM] Insufficient memory is available.

[ETXTBSY] The file specified by the loader parameter is currently open for writing by some process.

Related Information

Functions: exec(2)
exit, atexit, _exit

**Purpose**
Terminates a process

**Library**
Standard C Library (libc.a): atexit(), _exit()

**Synopsis**
```
#include <stdlib.h>

void exit (int status);
void _exit (int status);
int atexit (void (*function) (void));
```

**Parameters**
- `status` Indicates the status of the process.
- `function` Points to a function that is called at normal process termination for cleanup processing. A push-down stack of functions is kept, such that the last function registered is the first function called. Any function which is registered more than once will be repeated. Up to 32 functions can be specified with atexit().

**Description**
The `atexit()` function registers functions to be called at normal process termination for cleanup processing.

The `exit()` function terminates the calling process after calling the Standard I/O Library `_cleanup()` function to flush any buffered output. Then it calls any functions registered previously for the process by the `atexit()` function, in the reverse order to that in which they were registered. In addition, the `exit()` function flushes all open output streams, closes all open streams, and removes all files created by the `tmpfile()` function. Finally, it calls the `_exit()` function, which completes process termination and does not return.
The _exit() function terminates the calling process and causes the following to occur:

- All of the file descriptors, directory streams, and message catalog descriptors open in the calling process are closed. Since the exit() function terminates the process, any errors encountered during these close operations go unreported.

- Terminating a process by exiting does not terminate its child processes. Instead, the parent process ID of all of the calling process child processes and zombie child processes is set to the process ID of init. The init process thus inherits each of these processes, catches the SIGCHLD signals they generate, and calls the wait() function for each of them.

- If the parent process of the calling process is running a wait() or waitpid() function, it is notified of the termination of the calling process and the low-order 8 bits (that is, bits 0377 or 0xFF) of the status parameter are made available to it.

- If the parent process is not running a wait() or waitpid() function when the child process terminates, it may do so later on, and the child's status will be returned to it at that time. Meanwhile, the child process is transformed into a zombie process, and its parent process is sent a SIGCHLD signal to notify it of the termination of a child process.

A zombie process is a process that occupies a slot in the process table, but has no other space allocated to it either in user or kernel space. The process table slot that it occupies is partially overlaid with time accounting information to be used by the times() function. (See the sys/proc.h header file.)

A process remains a zombie until its parent issues one of the wait functions. At this time, the zombie is laid to rest, and its process table entry is released.

- The parent process is sent a SIGCHLD signal when a child terminates; however, since the default action for this signal is to ignore it, the signal usually is not seen.

If an exiting child's parent is ignoring the SIGCHLD signal, the child's parent process ID is changed to that of the initialization process, init, which will catch the SIGCHLD signal and call the wait() function.
exit(2)

- If the process is a controlling process, a SIGHUP signal is sent to each process in the foreground process group of the controlling terminal belonging to the calling process. The controlling terminal is disassociated from the session, allowing it to be acquired by a new controlling process.
- If the exit of a process causes a process group to become orphaned, and if any member of the newly orphaned process group is stopped, then a SIGHUP signal is sent to each newly orphaned process.
- Each attached shared memory segment is detached and the value of `shm_nattach` in the data structure associated with its shared memory identifier is decremented by 1.
- For each semaphore for which the calling process has set a `semadj` value, that `semadj` value is added to the `sempval` of the specified semaphore. (The `semop()` function provides information about semaphore operations.)
- If the process has a process lock, text lock, or data lock, an `unlock` is performed. (See the `plock()` function.)
- An accounting record is written on the accounting file if the system accounting routine is enabled. (The `acct()` function provides information about enabling accounting routines.)
- Locks set by the `fcntl()`, `flock()`, and `lockf()` functions are removed.

If a thread calls the `_exit()` function, the entire process exits and all threads within the process are terminated.

Notes

The system `init` process is used to assist cleanup of terminating processes. If the code for the `init` process is replaced, the program must be prepared to accept SIGCHLD signals and issue a `wait()` function for each.

AES Support Level: Full use

Return Values

The `exit()` function and `_exit()` function do not return. The `atexit()` function returns 0 (zero) if successful, and a nonzero value if there has been an attempt to register more `exit()` functions than can be held in the `atexit()` array.

Related Information

Functions: `acct(2)`, `sigaction(2)`, `times(3)`, `wait(2)`, `sigvec(2)`
exp, log, log10, pow

**Purpose**
Computes exponential, logarithm, and power functions.

**Library**
Math Library (libm.a)

**Synopsis**
```
#include <math.h>

double exp (double x);
double log10 (double x);
double log (double x);
double pow (double x, double y);
```

**Parameters**
- \( x \)
  - Specifies some double value.
- \( y \)
  - Specifies some double value.

**Description**
The `exp()` function computes the exponential function of \( x \), defined as \( e^x \), where \( e \) is the constant used as a base for natural logarithms.

The `log()` function computes the natural logarithm of \( x \).

The `log10()` function computes the base 10 logarithm of \( x \).

The `pow()` function computes the value of \( x \) raised to the power of \( y \) \((x^y)\). If \( x \) is negative, \( y \) must be an integral value. If \( x \) is 0 (zero), \( y \) must be nonnegative. The `pow(x,0,0)` function call returns 1.0 for all \( x \).
Notes

The exp(), log10(), log(), and pow() functions are supported for multi-threaded applications.

AES Support Level: Full use

Return Values

Upon successful completion, exp() returns the value of the exponential function of \( x \). If the correct value would overflow, the exp() function returns HUGE_VAL and sets errno to [ERANGE]. If the correct value would underflow, the exp() function returns zero. If \( x \) is NaN, NaN is returned.

The log() function returns the natural logarithm of \( x \). The value of \( x \) must be positive. If \( x \) is NaN, NaN is returned. Otherwise, either -HUGE_VAL or NaN is returned and errno is set to indicate the error.

The log10() function returns the base 10 logarithm of \( x \). The value of \( x \) must be positive. If \( x \) is NaN, NaN is returned and errno may be set to [EDOM]. Otherwise, either -HUGE_VAL or NaN is returned and errno is set to indicate the error.

The pow() function returns the value of \( x \) raised to the power of \( y \) \((x^y)\). If \( x \) is negative and \( y \) is not an integer, NaN is returned. If \( x \) is 0 (zero) and \( y \) is negative, -HUGE_VAL is returned. If \( x \) or \( y \) is NaN, NaN is returned and errno is set to [EDOM]. Otherwise, errno is set to indicate the error or [EDOM] is returned.

Errors

If the exp() function fails, errno may be set to one of the following values:

- [ERANGE] The result would overflow.
- [EDOM] The value of \( x \) is NaN.
- [ERANGE] The result would underflow.

If the log() function fails, errno may be set to one of the following values:

- [EDOM] The value of \( x \) is negative or zero.
- [EDOM] The value of \( x \) is NaN.
- [ERANGE] The logarithm of \( x \) cannot be represented, or the result would cause overflow.
If the \texttt{log10()} function fails, \texttt{errno} may be set to one of the following values:

- [EDOM] The value of \( x \) is negative or zero.
- [EDOM] The value of \( x \) is NaN.
- [ERANGE] The logarithm of \( x \) cannot be represented or the result would cause overflow.

If the \texttt{pow()} function fails, \texttt{errno} may be set to one of the following values:

- [EDOM] The value of \( x \) is negative and \( y \) is nonintegral.
- [ERANGE] The value to be returned would have caused overflow.
- [EDOM] The value of \( x \) or \( y \) is NaN, or \( x \) is zero and \( y \) is negative.
- [ERANGE] The value to be returned would have caused underflow.

**Related Information**

Functions: \texttt{hypot(3)}, \texttt{sinh(3)}
expacct

**Purpose**
Expands accounting record

**Synopsis**
```c
#include <sys/acct.h>

double expacct (comp_t record);
```

**Parameters**
- `record` Specifies the compressed data type value obtained from any source containing such information.

**Description**
The `expacct()` function converts `acct` structure members that have been packed into a pseudo floating-point format from the compressed data type `comp_t` to data type `double`.

**Notes**
The algorithm for compressing kernel accounting data is system dependent.

**Related Information**
- Functions: `acct(2)`
fclose, fflush

Purpose   Closes or flushes a stream

Library
Standard I/O package (libc.a)

Synopsis
#include <stdio.h>
int fclose (
   FILE *stream);
int fflush (    
   FILE *stream);

Parameters
stream       Specifies the output or update stream.

Description
The fclose() function writes buffered data to the stream specified by the stream parameter, and then closes the stream. It is automatically called for all open files when the exit() function is invoked. Any unwritten buffered data for the stream is delivered to the host environment to be written to the file; any unread buffered data is discarded. The stream is disassociated from the file. If the associated buffer was automatically allocated, it is deallocated. Any further use of the stream specified by the stream parameter causes undefined behavior.

The fclose() function performs close() on the file descriptor associated with the stream parameter. If the stream was writable and buffered data was not yet written to the file, it marks the st_ctime and st_mtime fields of the underlying file for update. If the file is not already at EOF, and is capable of seeking, the file pointer of the underlying open file description is adjusted so that the next operation on the open file description deals with the byte after the last one read from or written to the stream being closed.

The fflush() function writes any buffered data for the stream specified by the stream parameter and leaves the stream open. If stream is a null pointer, the fflush() function performs this flushing action on all streams for which the behavior is defined above. The st_ctime and st_mtime fields of the underlying file are marked for update. If the stream is open for reading, any unread data buffered in the stream is discarded. If the file is not already at EOF, the stream is open for
reading, and the file is capable of seeking, the file offset of the underlying open file
description is adjusted so that the next operation on the open file description deals
with the byte after the last one read from or written to the stream being flushed.

Notes

AES Support Level: Full use

Return Values

Upon successful completion, both the fclose() and fflush() functions return a value
of 0 (zero). Otherwise, EOF is returned and errno is set to indicate the error.

Errors

If the fclose() function fails, errno may be set to one of the following values:

- [EBADF] The file descriptor underlying the stream parameter is not valid.
- [EINTR] The fclose() function was interrupted by a signal which was caught.
- [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying the stream parameter and the process would be delayed in the write operation.
- [EFBIG] An attempt was made to write a file that exceeds the process’ file size limit or the maximum file size. See the ulimit() function.
- [EIO] The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU and the process group of the process is orphaned.
- [ENOSPC] There was no free space remaining on the device containing the file.
- [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

If the fflush() function fails, errno may be set to one of the following values:

- [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying stream and the process would be delayed in the write operation.
- [EBADF] The file descriptor underlying the stream parameter is not valid.
- [EFBIG] An attempt was made to write a file that exceeds the process’ file size limit or the maximum file size. See the ulimit() function.
- [EINTR] The fflush() function was interrupted by a signal which was caught.
The implementation supports job control, the process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

There was no free space remaining on the device containing the file.

An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

Related Information

Functions: close(2), exit(2), fopen(3), setbuf(3)
Functions
fcntl(2)

fcntl, dup, dup2

Purpose
Controls open file descriptors

Synopsis
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>

int fcntl(
    int filedes,
    int request [,
    int argument | struct flock *argument ]);

int dup(
    int filedes );

int dup2(
    int old,
    int new );

Parameters
filedes Specifies an open file descriptor obtained from a successful open(), fcntl(), or pipe() function.
request Specifies the operation to be performed.
argument Specifies a variable that depends on the value of the request parameter.
old Specifies an open file descriptor.
new Specifies an open file descriptor that is returned by the dup2() function.

Description
The fcntl() function performs controlling operations on the open file specified by the filedes parameter.

When the fcntl(), dup() and dup2() functions need to block, only the calling thread is suspended rather than all threads in the calling process.
The following are values for the request parameter:

F_DUPFD  Returns a new file descriptor as follows:

- Lowest numbered available file descriptor greater than or equal to the argument parameter, taken as type int.
- Same object references as the original file.
- Same file pointer as the original file. (That is, both file descriptors share one file pointer if the object is a file).
- Same access mode (read, write, or read-write).
- Same file status flags. (That is, both file descriptors share the same file status flags).
- The close-on-exec flag (FD_CLOEXEC bit) associated with the new file descriptor is cleared so that the file will remain open across exec functions.

F_GETFD  Gets the value of the close-on-exec flag associated with the file descriptor filedes. File descriptor flags are associated with a single file descriptor and do not affect other file descriptors that refer to the same file. The argument parameter is ignored.

F_SETFD  Sets the close-on-exec flag associated with the filedes parameter to the value of the argument parameter, taken as type int. If the argument parameter is 0 (zero), the file remains open across the exec functions. If the argument parameter is FD_CLOEXEC, the file is closed on successful execution of the next exec function.

F_GETFL  Gets the file status flags and file access modes for the file referred to by the filedes parameter. The file access modes can be extracted by using the mask O_ACCMODE on the return value. File status flags and file access modes are associated with the file description and do not affect other file descriptors that refer to the same file with different open file descriptions. The argument parameter is ignored.

F_SETFL  Sets the file status flags to the argument parameter, taken as type int, for the file to which the filedes parameter refers. The file access mode is not changed.
F_GETOWN  Gets the process ID or process group currently receiving SIGIO and SIGURG signals. Process groups are returned as negative values.

F_SETOWN  Sets the process or process group to receive SIGIO and SIGURG signals. Process groups are specified by supplying the argument parameter as negative; otherwise the argument parameter, taken as type int, is interpreted as a process ID.

The following values for the request parameter are available for record locking:

F_GETLK  Gets the first lock that blocks the lock description pointed to by the argument parameter, taken as a pointer to type struct flock. The information retrieved overwrites the information passed to the fcntl() function in the flock structure. If no lock is found that would prevent this lock from being created, then the structure is left unchanged except for the lock type, which is set to F_UNLCK.

F_SETLK  Sets or clears a file segment lock according to the lock description pointed to by argument, taken as a pointer to type struct flock. F_SETLK is used to establish shared locks (F_RDLCK), or exclusive locks (F_WRLCK), as well as remove either type of lock (F_UNLCK). If a shared (read) or exclusive (write) lock cannot be set, the fcntl() function returns immediately with a value of -1.

F_SETLKW  Same as F_SETLK except that if a shared or exclusive lock is blocked by other locks, the process will wait until it is unblocked. If a signal is received while fcntl() is waiting for a region, the function is interrupted, -1 is returned, and errno is set to [EINTR].

The O_NDELAY and O_NONBLOCK requests affect only operations against file descriptors derived from the same open() function. In BSD, these apply to all file descriptors that refer to the object.

When a shared lock is set on a segment of a file, other processes are able to set shared locks on that segment or a portion of it. A shared lock prevents any other process from setting an exclusive lock on any portion of the protected area. A request for a shared lock fails if the file descriptor was not opened with read access.

An exclusive lock prevents any other process from setting a shared lock or an exclusive lock on any portion of the protected area. A request for an exclusive lock fails if the file descriptor was not opened with write access.

The flock() structure describes the type (l_type), starting offset (l_whence), relative offset (l_start), size (l_len) and process ID (l_pid) of the segment of the file to be affected.
The value of l_whence is set to SEEK_SET, SEEK_CUR or SEEK_END, to indicate that the relative offset l_start bytes is measured from the start of the file, from the current position, or from the end of the file, respectively. The value of l_len is the number of consecutive bytes to be locked. The l_len value may be negative (where the definition of off_t permits negative values of l_len). The l_pid field is only used with F_GETLK to return the process ID of the process holding a blocking lock. After a successful F_GETLK request, the value of l_whence becomes SEEK_SET.

If l_len is positive, the area affected starts at l_start and ends at l_start + l_len - 1. If l_len is negative, the area affected starts at l_start + l_len and ends at l_start - 1. Locks may start and extend beyond the current end of a file, but may not be negative relative to the beginning of the file. If l_len is set to 0 (zero), a lock may be set to always extend to the largest possible value of the file offset for that file. If such a lock also has l_start set to 0 (zero) and l_whence is set to SEEK_SET, the whole file is locked. Changing or unlocking a portion from the middle of a larger locked segment leaves a smaller segment at either end. Locking a segment that is already locked by the calling process causes the old lock type to be removed and the new lock type to take effect. All locks associated with a file for a given process are removed when a file descriptor for that file is closed by that process or the process holding that file descriptor terminates. Locks are not inherited by a child process in a fork() function.

If a regular file has enforced record locking enabled, record locks on the file will affect calls to other calls, including creat(), open(), read(), write(), truncate(), and ftruncate().

A potential for deadlock occurs if a process controlling a locked region is put to sleep by attempting to lock another process’ locked region. If the system detects that sleeping until a locked region is unlocked would cause a deadlock, the fcntl() function fails with an [EDEADLK] error.

Notes

The dup(filedes) function is equivalent to fcntl(filedes, F_DUPFD, 0).

The dup2(oldfiledes, newfiledes) function has similar functionality to:

   close(newfiledes)
   fcntl(oldfiledes, F_DUPFD, newfiledes)

The file locks set by the fcntl() and lockf() functions do not interact in any way with the file locks set by the flock() function. If a process sets an exclusive lock on a file using the fcntl() or lockf() function, the lock will not affect any process that is setting or clearing locks on the same file using the flock() function. It is
therefore possible for an inconsistency to arise if a file is locked by different processes using flock() and fcntl(). (The fcntl() and lockf() functions use the same mechanism for record locking.)

AES Support Level: Full use

Return Values

Upon successful completion, the value returned depends on the value of the request parameter as follows:

- F_DUPFD: Returns a new file descriptor.
- F_GETFD: Returns FD_CLOEXEC or 0 (zero).
- F_SETFD: Returns a value other than -1.
- F_GETFL: Returns the value of file status flags and access modes. (The return value will not be negative.)
- F_SETFL: Returns a value other than -1.
- F_GETOWN: Returns the value of descriptor owner.
- F_GETLK: Returns a value other than -1.
- F_SETLK: Returns a value other than -1.
- F_SETLKW: Returns a value other than -1.

If the fcntl() function fails, a value of -1 is returned and errno is set to indicate the error.

Errors

If the fcntl() function fails, errno may be set to one of the following values:

- [EBADF]: The filedes parameter is not a valid open file descriptor.
- [EBADF]: The request parameter is F_SETLK or F_SETLKW, the type of lock (l_type) is a shared lock (F_RDLCK), and filedes is not a valid file descriptor open for reading.
- [EBADF]: The type of lock (l_type) is an exclusive lock (F_WRLCK), and filedes is not a valid file descriptor open for writing.
- [EMFILE]: The request parameter is F_DUPFD and OPEN_MAX file descriptors are currently open in the calling process, or no file descriptors greater than or equal to argument are available.
- [EINVAL]: The request parameter is F_DUPFD and the argument parameter is negative or greater than or equal to OPEN_MAX.
[EINVAL] An illegal value was provided for the request parameter.

EINVAL The request parameter is F_GETLK, F_SETLK, or F_SETLKW and the data pointed to by argument is invalid, or filedes refers to a file that does not support locking.

EFAULT The argument parameter is an invalid address.

ESRCH The value of the request parameter is F_SETOWN and the process ID given as argument is not in use.

EAGAIN The request parameter is F_SETLK, the type of lock (l_type) is a shared (F_RDLCK) or exclusive (F_WRLCK) lock, and the segment of a file to be locked is already exclusive-locked by another process.

EAGAIN The request parameter is F_SETLK, and the type is an exclusive lock and some portion of the segment of a file to be locked is already shared-locked or exclusive-locked by another process.

EINTR The request parameter is F_SETLKW and the fcntl() function was interrupted by a signal which was caught.

ENOLCK The request parameter is F_SETLK or F_SETLKW and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.

EDEADLK The request parameter is F_SETLKW, the lock is blocked by some lock from another process and putting the calling process to sleep, and waiting for that lock to become free would cause a deadlock.

If the dup() or dup2() function fails, errno may be set to one of the following values:

[EBADF] The filedes or old parameter is not a valid open file descriptor or the new parameter file descriptor is negative or greater than OPEN_MAX.

EMFILE The number of file descriptors exceeds OPEN_MAX or there is no file descriptor above the value of the new parameter.

EINTR The dup2() function was interrupted by a signal which was caught.

Related Information

Functions: close(2), exec(2), lockf(3), open(2), read(2), truncate(2), write(2)
feof

Purpose  Tests EOF on a stream

Library

Standard I/O Package (libc.a)

Synopsis

```
#include <stdio.h>
int feof (
    FILE *stream);
```

Parameters

- `stream` Specifies the input stream.

Description

The `feof()` macro tests the EOF (End Of File) condition on the specified stream.

Notes

**AES Support Level:** Full use

Return Values

If EOF has previously been detected reading the input stream specified by the `stream` parameter, a nonzero value is returned. Otherwise, a value of 0 (zero) is returned.

Related Information

Functions: `ferror(3)`, `fileno(3)`, `clearerr(3)`, `fopen(3)`
ferror

Purpose Tests the error indicator on a stream

Library

Standard I/O package (libc.a)

Synopsis #include <stdio.h>

int ferror (
    FILE *stream );

Parameters

stream Specifies the input or output stream.

Description

The ferror() macro tests whether input/output errors have occurred on the specified stream.

Notes

AES Support Level: Full use

Return Values

If an I/O error occurred when reading from or writing to the stream specified by the stream parameter, a nonzero value is returned. Otherwise, a value of 0 (zero) is returned.

Related Information

Functions: fopen(3), feof(3), fileno(3), clearerr(3)
fileno

**Purpose**
Maps stream pointer to file descriptor

**Library**
Standard I/O Package (libc.a)

**Synopsis**
```c
#include <stdio.h>

int fileno (FILE *stream);
```

**Parameters**
- `stream` Specifies the input stream.

**Description**
The `fileno()` macro returns the file descriptor of a stream.

**Notes**
AES Support Level: Full use

**Return Values**
The `fileno()` macro returns the file descriptor associated with the `stream` parameter.

**Related Information**
Functions: `clearerr(3), feof(3), ferror(3), fopen(3), open(2)`
flock

**Purpose**
Applies or removes an advisory lock on an open file

**Synopsis**
```
#include <sys/file.h>

#define LOCK_SH
#define LOCK_EX
#define LOCK_NB
#define LOCK_UN

int flock(
    int filedes,
    int operation);
```

**Parameters**
- `filedes` Specifies a file descriptor returned by a successful `open()` or `fcntl()` function, identifying the file to which the lock is to be applied or removed.
- `operation` Specifies one of the following constants for `flock()`, defined in the `fcntl.h` file:
  - `LOCK_SH` Apply a shared lock.
  - `LOCK_EX` Apply an exclusive lock.
  - `LOCK_NB` Do not block when locking. This value can be logically ORed with either `LOCK_SH` or `LOCK_EX`.
  - `LOCK_UN` Remove a lock.

**Description**
The `flock()` function applies or removes an advisory lock on the file associated with the `filedes` file descriptor. Advisory locks allow cooperating processes to perform consistent operations on files, but do not guarantee consistency (that is, processes may still access files without using advisory locks, possibly resulting in inconsistencies).

The locking mechanism allows two types of locks: shared locks and exclusive locks. At any time multiple shared locks may be applied to a file, but at no time are multiple exclusive, or both shared and exclusive, locks allowed simultaneously on a file.
A shared lock may be upgraded to an exclusive lock, and vice versa, simply by specifying the appropriate lock type. This results in the previous lock being released and the new lock applied (possibly after other processes have gained and released the lock).

Requesting a lock on an object that is already locked normally causes the caller to be blocked until the lock may be acquired. If LOCK_NB is included in operation, then this will not happen; instead, the call will fail and errno will be set to [EWOULDBLOCK].

Notes

Locked are on files, not file descriptors. That is, file descriptors duplicated using the dup() or fork() functions do not result in multiple instances of a lock, but rather multiple references to a single lock. If a process holding a lock on a file forks and the child explicitly unlocks the file, the parent will lose its lock.

Processes blocked awaiting a lock may be awakened by signals.

The file locks set by the flock() function do not interact in any way with the file locks set by the fcntl() and lockf() functions. If a process sets an exclusive lock on a file using the flock() function, the lock will not affect any process that is setting or clearing locks on the same file using the fcntl() or lockf() functions. It is therefore possible for an inconsistency to arise if a file is locked by different processes using flock() and fcntl(). (The fcntl() and lockf() functions use the same mechanism for record locking.)

Return Values

Upon successful completion, 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.

Errors

If the flock() function fails, errno may be set to one of the following values:

[EWOULDBLOCK] The file is locked and the LOCK_NB option was specified.

[EBADF] The the filedes argument is not a valid open file descriptor.

[EINVAL] The operator is not valid.
flock(2)

[ENOLCK] The lock table is full. Too many regions are already locked.

[EDEADLK] The lock is blocked by some lock from another process. Putting the calling process to sleep while waiting for that lock to become free would cause a deadlock.

Related Information

Functions: open(2), close(2), exec(2), fcntl(2), fork(2), lockf(3)
flockfile

Purpose
Locks a stdio stream

Library
Locks Library (libc_r.a)

Synopsis
#include <stdio.h>
void flockfile(
    FILE * file);

Parameters
file Specifies the stream to be locked.

Description
The flockfile() function locks a stdio stream so that a thread can have exclusive use of that stream for multiple I/O operations. Use the flockfile() function for a thread that wishes to ensure that the output of several printf() functions, for example, is not garbled by another thread also trying to use printf().

Matching flockfile() and funlockfile() calls can be nested.

The behavior of the flockfile() function is unspecified if the file parameter does not point to a valid FILE structure.

Related Information
Functions: funlockfile(3), unlocked_getc(3), unlocked_putc(3)
floor, ceil, rint, fmod, fabs

Purpose  
Rounds floating-point numbers to floating-point integers, or computes the Modulo Remainder and floating-point absolute value functions

Library  
Math Library(libm.a)  
Standard C Library (libc.a)

Synopsis  
#include <math.h>

double floor (double x);
double ceil (double x);
double fmod (double x, double y);
double fabs (double x);
double rint (double x);

Parameters  
\textit{x} \quad \text{Specifies some double value.}
\textit{y} \quad \text{Specifies some double value.}

Description  
The \texttt{floor()} function returns the largest floating-point integer not greater than the \textit{x} parameter.

The \texttt{ceil()} function returns the smallest floating-point integer not less than the \textit{x} parameter.

The \texttt{rint()} function returns one of the two nearest floating point integers to the \textit{x} parameter. Which integer is returned is determined by the current floating-point rounding mode as described in the IEEE Standard for Binary Floating Point
Arithmetic. If the current rounding mode is round toward -infinity, then \texttt{rint}(x) is identical to \texttt{floor}(x). If the current rounding mode is round toward +infinity, then \texttt{rint}(x) is identical to \texttt{ceil}(x).

The \texttt{fmod}() function computes the modulo floating-point remainder of \(x/y\). The \texttt{fmod}() function returns the value \(x-(i*y)\) for some \(i\) such that if \(y\) is nonzero, the result has the same sign as \(x\) and magnitude less than the magnitude of \(y\).

The \texttt{fabs}() function returns the absolute value of \(x\), a floating-point number.

Notes

The default floating-point rounding mode is round to nearest. All C main programs begin with the rounding mode set to round to nearest.

\textbf{AES Support Level:} Full use (\texttt{floor()}, \texttt{ceil()}, \texttt{fmod()}, \texttt{fabs}())

Return Values

Upon successful completion, the \texttt{floor()} function returns the largest integral value not greater than \(x\). If \(x\) is NaN, NaN is returned and \texttt{errno} may be set to [EDOM]. Otherwise, -HUGE_VAL is returned.

Upon successful completion, the \texttt{ceil()} function returns the smallest integral value not less than \(x\). If \(x\) is NaN, NaN is returned. Otherwise, either HUGE_VAL or NaN is returned.

Upon successful completion, the \texttt{fmod()} function returns the remainder of the division of \(x\) by \(y\). If \(x\) or \(y\) is NaN, NaN is returned. If \(y\) is 0 (zero), the \texttt{fmod}() function returns NaN and sets \texttt{errno} to [EDOM].

Upon successful completion, the \texttt{fabs}() function returns the absolute value of \(x\). If \(x\) is NaN, NaN is returned. Otherwise, either \texttt{errno} is set to indicate the error or NaN is returned.

Errors

If the \texttt{floor()} function fails, \texttt{errno} may be set to one of the following values:

- [ERANGE] The result would cause an overflow.
- [EDOM] The value of \(x\) is NaN.

If the \texttt{ceil()} function fails, \texttt{errno} may be set to one of the following values:

- [ERANGE] The result would cause an overflow.
- [EDOM] The value of \(x\) is NaN.
If the `fmod()` function fails, `errno` may be set to the following value:
[EDOM] The `y` argument is zero or one of the arguments is NaN.

If the `fabs()` function fails, `errno` may be set to the following value:
[EDOM] The value of `x` is NaN.

**Related Information**

Functions: `isnan(3)`
fopen, freopen, fdopen

**Purpose**
Opens a stream

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <stdio.h>

FILE *fopen (const char *path, const char *type);

FILE *fdopen (int filedes, const char *type);

FILE *freopen (const char *path, const char *type, FILE *stream);
```

**Parameters**
- *path* Points to a character string that contains the name of the file to be opened. If the final component of the *path* parameter specifies a symbolic link, the link is traversed and pathname resolution continues.
- *type* Points to a character string that has one of the following values:
  - r Open text file for reading.
  - w Create a new text file for writing, or open and truncate to zero length. (The file is not truncated under the fdopen() function.)
  - a Append (open text file for writing at the end of the file, or create for writing).
  - rb Open binary file for reading.
  - wb Create a binary file for writing, or open and truncate to zero length.
  - ab Append (open binary file for update, writing at the end of the file, or create for writing).
Description

The `fopen()` function opens the file named by the `path` parameter and associates a stream with it, returning a pointer to the `FILE` structure of this stream.

When you open a file for update, you can perform both input and output operations on the resulting stream. However, an output operation cannot be directly followed by an input operation without an intervening `flush()` function call or a file positioning operation (`.fseek()`, `.fsetpos()`, or `.rewind` function). Also, an input operation cannot be directly followed by an output operation without an intervening flush or file positioning operation, unless the input operation encounters the end of the file.

When you open a file for append (that is, when the `type` parameter is `a` or `a+`), it is impossible to overwrite information already in the file. You can use the `.fseek()` function to reposition the file pointer to any position in the file, but when output is written to the file, the current file pointer is ignored. All output is written at the end of the file and the file pointer is repositioned to the end of the output.

If two separate processes open the same file for append, each process can write freely to the file without destroying the output being written by the other. The output from the two processes is intermixed in the order in which it is written to the file. Note that if the data is buffered, it is not actually written until it is flushed.
When opened, a stream is fully buffered if and only if it can be determined not to refer to an interactive device. The error and End-of-File indicators for the stream are cleared.

If the type parameter is w, a, w+, or a+ and the file did not previously exist, upon successful completion the fopen() function marks the st_atime, st_ctime and st_mtime fields of the file and the st_ctime and st_mtime fields of the parent directory for update. If the type parameter is w or w+ and the file did previously exist, upon successful completion the fopen() function marks the st_ctime and st_mtime fields of the file for update.

The freopen() function substitutes the named file in place of the open stream. The original stream is closed regardless of whether the open() function succeeds with the named file. The freopen() function returns a pointer to the FILE structure associated with the stream parameter. The freopen() function is typically used to attach the preopened streams associated with stdin, stdout, and stderr to other files.

The fdopen() function associates a stream with a file descriptor obtained from an open(), dup(), creat(), or pipe() function. These functions open files but do not return pointers to FILE structures. Many of the standard I/O package functions require pointers to FILE structures. Note that the type of stream specified must agree with the mode of the open file.

Notes

AES Support Level: Full use

Return Values

If the fopen(), fdopen(), or freopen() function fails, a null pointer is returned and errno may be set to indicate the error.

Errors

If the fopen() function fails, errno may be set to one of the following values:

[EACCES] Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by the type parameter are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.

[EINTR] The fopen() function was interrupted by a signal which was caught.
The named file is a directory and type requires write access.

OPEN_MAX file descriptors are currently open in the calling process.

Too many links were encountered in translating path.

The length of the path string exceeds PATH_MAX or a pathname component is longer than NAME_MAX.

Too many files are currently open in the system.

The named file does not exist or the path parameter points to an empty string.

The directory or file system that would contain the new file cannot be expanded.

A component of the path prefix is not a directory.

The named file is a character special or block special file and the device associated with this special file does not exist.

The named file resides on a read only file system and type requires write access.

The file is being executed and mode requires write access.

If the fdopen() function fails, errno may be set to one of the following values:

The filedes parameter is not a valid file descriptor.

The type parameter is not a valid mode.

Insufficient space to allocate a buffer.

The freopen() function fails if the following is true:

Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by the type parameter are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.

The freopen() function was interrupted by a signal which was caught.

The named file is a directory and type requires write access.

OPEN_MAX file descriptors are currently open in the calling process.

Too many links were encountered in translating path.
[ENAMETOOLONG]  
The length of the *path* string exceeds PATH_MAX or a pathname component is longer than NAME_MAX.

[ENFILE]  
Too many files are currently open in the system.

[ENOENT]  
The named file does not exist or the *path* parameter points to an empty string.

[ENOSPC]  
The directory or file system that would contain the new file cannot be expanded.

[ENOTDIR]  
A component of the path prefix is not a directory.

[ENXIO]  
The named file is a character special or block special file, and the device associated with this special file does not exist.

[ERofs]  
The named file resides on a read only file system and *type* requires write access.

[EINVAL]  
The *type* parameter is not a valid type.

[ETXTBSY]  
The file is being executed and *mode* requires write access.

**Related Information**

Functions: open(2), fclose(3), fseek(3), setbuf(3)
fork, vfork

Purpose

Creates a new process

Synopsis

#include <sys/types.h>

pid_t fork ( void );

pid_t vfork ( void );

Description

The fork() and vfork() functions create a new process (child process) that is identical to the calling process (parent process).

The child process inherits the following attributes from the parent process:

- Environment
- Close-on-exec flags
- Signal handling settings
- Set user ID mode bit
- Set group ID mode bit
- Trusted state
- Profiling on/off status
- Nice value
- All attached shared libraries
- Process group ID
- tty group ID
- Current directory
- Root directory
- File mode creation mask
- File size limit
- Attached shared memory segments
- Attached mapped file segments
- All mapped regions with the same protection and sharing mode as in the parent process
The child process differs from the parent process in the following ways:

- The child process has a unique process ID and does not match any active process group ID.
- The parent process ID of the child process matches the process ID of the parent.
- The child process has its own copy of the parent process's file descriptors. However, each of the child's file descriptors shares a common file pointer with the corresponding file descriptor of the parent process.
- All semadj values are cleared.
- Process locks, text locks, and data locks are not inherited by the child process.
- The child process's utime(), stime(), cutime(), and cstime() are set to 0 (zero).
- Any pending alarms are cleared in the child process.
- Any interval timers enabled by the parent process are disabled in the child process.
- Any signals pending for the parent process are disabled for the child process.

If a multithreaded process calls the fork() function, the new process contains a replica of the calling thread and its entire address space, possibly including the states of mutexes and other resources. Consequently, to avoid errors, the child process should only execute operations it knows will not cause deadlock until one of the exec functions is called.

Notes

The fork() function is supported for multi-threaded applications.

The vfork() function is supported as a compatibility interface for older BSD system programs, and can be used by compiling with Berkeley Compatibility Library (libbsd.a). The memory sharing semantics of the vfork() function are synonymous with the fork() function.

AES Support Level: Full use (fork())
Return Values

Upon successful completion, the `fork()` function returns a value of 0 (zero) to the child process and returns the process ID of the child process to the parent process. If the `fork()` function fails, a value of -1 is returned to the parent process, no child process is created, and `errno` is set to indicate the error.

Errors

If the `fork()` function fails, `errno` may be set to one of the following values:

- **[EAGAIN]** The system-imposed limit on the total number of processes executing for a single user would be exceeded. This limit can be exceeded by a process with superuser privilege.
- **[ENOMEM]** There is not enough space left for this process.

Related Information

Functions: `exec(2)`, `exit(2)`, `getpriority(2)`, `getrusage(2)`, `nice(3)`, `plock(2)`, `ptrace(2)`, `raise(3)`, `semop(2)`, `shmat(2)`, `sigaction(2)`, `sigvec(2)`, `times(3)`, `ulimit(3)`, `umask(2)`, `wait(2)`
fread, fwrite

**Purpose**
Performs input/output

**Library**
Standard I/O Package (libc.a)

**Synopsis**

```c
#include <stdio.h>

size_t fread (  
    void *pointer,  
    size_t size,  
    size_t num_items,  
    FILE *stream );

size_t fwrite (  
    const void *pointer,  
    size_t size,  
    size_t num_items,  
    FILE *stream );
```

**Parameters**

- **pointer**
  Points to an array.
- **size**
  Specifies the size of the variable type of the array pointed to by the `pointer` parameter.
- **num_items**
  Specifies the number of items of data.
- **stream**
  Specifies the input or output stream.

**Description**

The `fread()` function copies `num_items` items of data of length `size` from the input stream into an array beginning at the location pointed to by the `pointer` parameter.

The `fread()` function stops copying bytes if an End-of-File or error condition is encountered while reading from the input specified by the `stream` parameter, or when the number of data items specified by the `num_items` parameter have been copied. It leaves the file pointer of the `stream` parameter, if defined, pointing to the byte following the last byte read, if there is one. The `fread()` function does not change the contents of the `stream` parameter.
The `fwrite()` function appends `num_items` items of data of length `size` from the array pointed to by the `pointer` parameter to the output stream.

The `fwrite()` function stops writing bytes if an error condition is encountered on the stream, or when the number of items of data specified by the `num_items` parameter have been written. The `fwrite()` function does not change the contents of the array pointed to by the `pointer` parameter.

**Notes**

AES Support Level: Full use

**Return Values**

Upon successful completion, the `fread()` and `fwrite()` functions return the number of items actually transferred. If the `num_items` parameter is negative or 0 (zero), no characters are transferred, and a value of 0 is returned. If a read or write error occurs, the error indicator for the stream is set and `errno` is set to indicate the error.

**Errors**

Refer to the reference page for the `fputc()` function for error codes returned by `fread()` and `fwrite()`.

**Related Information**

Functions: `fopen(3)`, `getc(3)`, `gets(3)`, `printf(3)`, `putc(3)`, `puts(3)`, `read(2)`, `scanf(3)`, `write(2)`
frexp, ldexp, modf

**Purpose**
Manipulates floating-point numbers

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <math.h>

double frexp (double value, int *exp);

double ldexp (double mantissa, int exp);

double modf (double value, double *int_pointer);
```

**Parameters**
- `value`: Specifies some double value.
- `exp`: Specifies an integer pointer to store the exponent for `frexp()`; for `ldexp()`, specifies some integer value.
- `mantissa`: Specifies some double value.
- `int_pointer`: Specifies a double pointer in which to store the signed integral part.

**Description**
Every nonzero number can be written uniquely as \( x \) times 2 raised to the power \( n \), where the mantissa (fraction), \( x \), is in the range \( 0.5 \leq |x| < 1.0 \), and the exponent, \( n \), is an integer.

The `frexp()` function breaks a floating-point number into a normalized fraction and an integral power of 2. It stores the integer in the `int` object pointed to by the `exp` parameter and returns the fraction part.

The `ldexp()` function multiplies a floating-point number by an integral power of 2.
The `modf()` function breaks the `value` parameter into an integral and fractional part, each of which has the same sign as the `value` parameter. It stores the integral part as a `double` in the location pointed to by the `int_pointer` parameter.

**Notes**

The `frexp()` and `modf()` functions are supported for multi-threaded applications. The `ldexp()` function is not supported for multiple threads.

**AES Support Level:** Full use

**Return Values**

Upon successful completion, the `frexp()` function returns the value `x` such that `x` is a `double` with magnitude in the interval 1/2 to 1, or 0 (zero), and `value` equals `x` times 2 raised to the power of `*exp`. If `value` is 0, both parts of the result are 0. If `value` is NaN, then the result is NaN and `*exp` is set to LONG_MIN. If `value` is `+infinity`, then the result is +0.0 and `*exp` is set to +LONG_MAX.

Upon successful completion, the `ldexp()` function returns a `double` equal to `value` times 2 to the power `exp`. If `value` is NaN, NaN is returned. If `ldexp()` would cause overflow, ±HUGE_VAL is returned (according to the sign of `value`) and `errno` is set to [ERANGE]. If `ldexp()` would cause underflow, 0 (zero) is returned. Otherwise, either `errno` is set to indicate the error or NaN is returned.

Upon successful completion, the `modf()` function returns the signed fractional part of `value` and stores the signed integral part in the object pointed to by `int_pointer`. If `value` is NaNQ or NaNS, then NaNQ is returned and NaNQ is stored in the object pointed to by `int_pointer`. If `value` is +infinity, then a +0.0 is returned and +infinity is stored in the object pointed to by `int_pointer`.

**Errors**

If the `ldexp()` function fails, `errno` may be set to one of the following values:

- `[ERANGE]` The value to be returned would cause overflow or underflow.
- `[EDOM]` The `value` parameter is NaN.

If the `modf()` function fails, `errno` may be set to one of the following values:

- `[EDOM]` The `value` parameter is NaN.
If the `frexp()` function fails, `errno` may be set to one of the following values:

[EDOM] The *value* parameter is NaN or infinity.

**Related Information**

Functions: `isnan(3)`
fseek, rewind, ftell, fgetpos, fsetpos

**Purpose**
Repositions the file pointer of a stream

**Library**
Standard I/O Package (libc.a)

**Synopsis**
```
#include <stdio.h>

int fseek (FILE *stream, long int offset, int whence);
void rewind (FILE *stream);
long int ftell (FILE *stream);
int fsetpos (FILE *stream, const fpos_t *position);
int fgetpos (FILE *stream, fpos_t *position);
```

**Parameters**
- `stream` Specifies the I/O stream.
- `offset` Determines the position of the next operation.
- `whence` Determines the value for the file pointer associated with the `stream` parameter.
- `position` Specifies the value of the file position indicator.

**Description**
The `fseek()` function sets the position of the next input or output operation on the I/O stream specified by the `stream` parameter. The position of the next operation is determined by the `offset` parameter, which can be either positive or negative.
The `fseek()` function sets the file pointer associated with the specified `stream` as follows:

- If the `whence` parameter is `SEEK_SET(0)`, the pointer is set to the value of the `offset` parameter.
- If the `whence` parameter is `SEEK_SET(1)`, the pointer is set to its current location plus the value of the `offset` parameter.
- If the `whence` parameter is `SEEK_SET(2)`, the pointer is set to the size of the file plus the value of the `offset` parameter.

The `fseek()` function fails if attempted on a file that was not opened with the `fopen()` function. In particular, the `fseek()` function cannot be used on a terminal or on a file opened with the `popen()` function.

A successful call to the `fseek()` function clears the End-of-File indicator for the stream and undoes any effects of the `ungetc()` function on the same stream. After a call to the `fseek()` function, the next operation on an update stream may be either input or output.

If the stream is writable and buffered data was not written to the underlying file, the `fseek()` function causes the unwritten data to be written to the file and marks the `st_ctime` and `st_mtime` fields of the file for update.

The `fseek()` function allows the file-position indicator to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads of data in the gap will return bytes with the value 0 (zero) until data is actually written into the gap. The `fseek()` function does not, by itself, extend the size of a file.

The `rewind()` function is equivalent to `(void) fseek(stream, 0L, SEEK_SET)`, except that it also clears the error indicator.

The `ftell()` function obtains the current value of the file position indicator for the specified stream.

The `fgetpos()` and `fsetpos()` functions are similar to the `ftell()` and `fseek()` functions, respectively. The `fgetpos()` function stores the current value of the file position indicator for the stream pointed to by the `stream` parameter in the object pointed to by the `position` parameter. The `fsetpos` function sets the file position indicator according to the value of the `position` parameter, returned by a prior call to the `fgetpos()` function.

A successful call to the `fsetpos()` function clears the EOF indicator and undoes any effects of the `ungetc()` function.
Notes

AES Support Level: Full use

Return Values

Upon successful completion, the fseek() function returns a value of 0 (zero). If the fseek() function fails, a value of -1 is returned and errno is set to indicate the error.

The rewind() function does not return a value.

Upon successful completion, the ftell() function returns the offset of the current byte relative to the beginning of the file associated with the named stream. Otherwise, -1 is returned and errno is set to indicate the error.

Upon successful completion, the fgetpos() and fsetpos() functions return 0 (zero). If the fgetpos() or the fsetpos() function fails, a value of -1 is returned and errno is set to [EINVAL].

Errors

The fseek() function fails if either the stream is unbuffered, or the stream's buffer needed to be flushed and the call to fseek() caused an underlying lseek() or write() function to be invoked. In addition, if the fseek() function fails, errno may be set to one of the following values:

[EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying the stream parameter and the process would be delayed in the write operation.

[EBADF] The file descriptor underlying the stream parameter is not a valid file descriptor open for writing.

[EFBIG] An attempt was made to write to a file that exceeds the process’ file size limit or the maximum file size. See the ulimit() function.

[EINTR] The read operation was interrupted by a signal which was caught, and no data was transferred.

[EIO] The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned.
[ENOSPC] There was no free space remaining on the device containing the file.

[EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

The rewind() and ftell() functions fail under the same conditions as the fseek() function, with the exception of [EINVAL], which does not apply.

If the fgetpos() or fsetpos() function fails, errno may be set to the following value:

[EINVAL] The stream parameter does not point to a valid FILE structure.

Related Information

Functions: lseek(2), fopen(3)
fsync

Purpose  Writes changes in a file to permanent storage

Synopsis  int fsync (  
            int filedes );

Parameters
  
  filedes  Specifies a valid open file descriptor.

Description

The fsync() function saves all modified data in the file open on the filedes parameter to permanent storage. On return from the fsync() function, all updates have been saved on permanent storage.

Notes

The file identified by the filedes parameter must be open for writing when the fsync() function is issued or the call fails. This restriction was not enforced in BSD systems.

AES Support Level: Trial use

Return Values

Upon successful completion, the fsync() function returns a value of 0 (zero). If fsync() fails, a value of -1 is returned and errno is set to indicate the error.

Errors

If the fsync() function fails, errno may be set to one of the following values:

[EIO]  An I/O error occurred while reading from or writing to the file system.

[EBADFD]  The filedes parameter is not a valid file descriptor open for writing.
[EINVAL] The `filedes` parameter does not refer to a file on which this operation is possible.

[EINTR] The `fsync()` function was interrupted by a signal which was caught.

Related Information

Functions: `open(2)`, `sync(2)`, `write(2)`
ftok

Purpose Generates a standard interprocess communication key

Library Standard C Library (libc.a)

Synopsis

```c
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok(
    char *path_name,
    char project_id);
```

Parameters

- `path_name` Specifies the pathname of an existing file that is accessible to the process.
- `project_id` Specifies a character that uniquely identifies a project.

Description

The `ftok()` function returns a key, based on the `path_name` and `project_id` parameters, to be used to obtain interprocess communication identifiers. The `ftok()` function returns the same key for linked files if called with the same `project_id` parameter. Different keys are returned for the same file if different `project_id` parameters are used.

Interprocess communication facilities require you to supply a key to the `msgget()`, `semget()`, and `shmget()` functions in order to obtain interprocess communication identifiers. The `ftok()` function provides one method of creating keys, but many others are possible. For example, you can use the project ID as the most significant byte of the key, and use the remaining portion as a sequence number.
Caution

It is important for each installation to define standards for forming keys. If some standard is not adhered to, it is possible for unrelated processes to interfere with each other’s operation.

Return Values

Upon successful completion, the ftok() function returns a key that can be passed to the msgget(), semget(), or shmget() function. The ftok() function returns the value (key_t)-1 if any of the following are true:

- The file named by the path_name parameter does not exist.
- The file named by the path_name parameter is not accessible to the process.
- The project_id parameter is 0 (zero) or the null string ("").

If the path_name parameter of the ftok() function names a file that has been removed while keys still refer to it, then the ftok() function returns an error. If that file is then recreated, the ftok() function may return a different key than the original one.

Related Information

Functions: msgget(2), semget(2), shmget(2)
ftw

Purpose  Walks a file tree

Library  Standard C Library (libc.a)

Synopsis  
#include <ftw.h>

int ftw (  
    const char *path,  
    int (*)(char*, const struct stat*, int),  
    int depth);

Parameters

path  Specifies the directory hierarchy to be searched.
function  Specifies the file type.
depth  Specifies the maximum number of file descriptors to be used.

Description

The ftw() function recursively searches the directory hierarchy that descends from the directory specified by the path parameter.

For each file in the hierarchy, the ftw() function calls the function specified by the function parameter, passes it a pointer to a null-terminated character string containing the name of the file, a pointer to a stat structure containing information about the file, and an integer. (See the stat() function for more information about this structure.)

The integer passed to the function parameter identifies the file type, and it has one of the following values:

FTW_F  Regular file
FTW_D  Directory
FTW_DNR  Directory that cannot be read
FTW_SL  Symbolic link
FTW_NS  A file for which the lstat() function could not be executed successfully
If the integer is FTW_DNR, then the files and subdirectories contained in that directory are not processed.

If the integer is FTW_NS, then the stat structure contents are meaningless. An example of a file that causes FTW_NS to be passed to the function parameter is a file in a directory for which you have read permission but not execute (search) permission.

The ftw() function finishes processing a directory before processing any of its files or subdirectories.

The ftw() function continues the search until the directory hierarchy specified by the path parameter is completed, an invocation of the function specified by the function parameter returns a nonzero value, or an error is detected within the ftw() function, such as an I/O error.

Because the ftw() function is recursive, it is possible for it to terminate with a memory fault due to stack overflow when applied to very deep file structures.

The ftw() function uses the malloc() function to allocate dynamic storage during its operation. If the ftw() function is terminated prior to its completion, such as by the longjmp() function being executed by the function specified by the function parameter or by an interrupt routine, the ftw() function cannot free that storage. The storage remains allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have the function specified by the function parameter return a nonzero value the next time it is called.

The ftw() function traverses symbolic links encountered in the resolution of path, including the final component. Symbolic links encountered while walking the directory tree rooted at path will not be traversed.

Notes

AES Support Level: Trial use

Return Values

If the directory hierarchy is completed, the ftw() function returns a value of 0 (zero). If the function specified by the function parameter returns a nonzero value, the ftw() function stops its search and returns the value that was returned by the function. If the ftw() function detects an error, a value of -1 is returned and errno is set to indicate the error.
Errors

If the `ftw()` function fails, `errno` may be set to one of the following values:

[EACCES] Search permission is denied for any component of the `path` parameter or read permission is denied for the `path` parameter.

[ENAMETOOLONG] The length of the `path` string exceeds PATH_MAX, or a pathname component is longer than NAME_MAX.

[ENOENT] The `path` parameter points to the name of a file which does not exist or points to an empty string.

[ENOTDIR] A component of the `path` parameter is not a directory.

[ENOMEM] There is insufficient memory for this operation.

In addition, if the function pointed to by the `function` parameter encounters an error, `errno` may be set accordingly.

Related Information

Functions: `malloc(3), setjmp(3), sigaction(2), stat(2)`
funlockfile

Purpose     Unlocks a stdio stream

Library     Reentrant Library (libc_r.a)

Synopsis    #include <stdio.h>
             void funlockfile(
                 FILE * file);

Parameters

  file     Specifies the stream to be unlocked.

Description

The funlockfile() function unlocks a stdio stream, causing the thread that had been
holding the lock to relinquish exclusive use of the stream.

Matching flockfile() and funlockfile() calls can be nested. If the stream has been
locked recursively, then it will remain locked until the last matching funlockfile() is called.

Behavior is unspecified if the file parameter is not pointer to a valid FILE structure.

Related Information

Functions: flockfile(3), unlocked_getc(3), unlocked_putc(3)
lgamma, gamma

Purpose
Computes the logarithm of the gamma function

Library
Math Library (libm.a)

Synopsis
#include <math.h>

double gamma (double x);

double lgamma (double x);

extern int signgam;

Parameters

x Specifies some positive double value.

Description

The gamma() and lgamma() functions return the logarithm of the absolute value of the gamma of x, where the gamma of x is defined as:

\[ \int_0^{\infty} e^{-t} t^{x-1} dt \]

The names lgamma() and gamma() are different names for the same function. The sign of the gamma of x is stored in the external integer variable signgam. The x parameter cannot be a nonpositive integer. The gamma of x is defined over the reals, except the nonpositive integers.
Notes

Do not use the expression
\[ g = \text{signgam} \times \exp (\text{lgamma} (x)) \]
to compute
\[ g = \text{gamma} (x) \]
Instead, use the following sequence:
\[
\begin{align*}
lg &= \text{lgamma} (x); \\
g &= \text{signgam} \times \exp (lg);
\end{align*}
\]
This is because the C language does not specify evaluation order, and \text{signgam} is modified by the call to the \text{lgamma}() function.

AES Support Level: Trial use

Return Values

If the \text{gamma}() or \text{lgamma}() function fails, either INF or NaN is returned.

Related Information

Functions: \text{exp}(3)
getaddressconf

Purpose
Gets information about system address space configuration

Synopsis
```
#include <sys/types.h>
#include <sys/addrconf.h>

int getaddressconf (
    struct addressconf *buffer,
    size_t length);
```

Parameters

- `buffer` Points to an array of `addressconf` structures.
- `length` Specifies the size in bytes of the array pointed to by the `buffer` parameter.

Description
The `getaddressconf()` function fills in the array of structures pointed to by the `buffer` parameter with information describing the configuration of process address spaces on this system. This information is intended to be used by programs such as the program loader, which need to manage the contents of a process’ address space using the memory management primitives such as the `mmap()` function.

The `buffer` parameter points to an array of `addressconf` structures, occupying a total of `length` bytes. Each element of the array describes a single area of the process address space. The `addressconf` structure is defined in the `sys/addrconf.h` header file, and it contains the following members:

- `caddr_t ac_base;`
- `unsigned ac_flags;`

`ac_base` The base virtual address of the area. For an upward-growing area, this is the lowest virtual address in the area; for a downward-growing area, this is the lowest virtual address above the area.

`ac_flags` The flags describe the area. They are also defined in the `sys/addrconf.h` header file, and are described as follows:

- `AC_UPWARD`
  The area grows towards higher addresses. The base address specified is the lowest address in the area.
AC_DOWNWARD
The area grows towards lower addresses. The base address specified is the lowest address above the area.

AC_FIXED
The area always starts at the specified base address. For example, on many machines the text area is a fixed area.

AC_FLOAT
The area floats to the first available virtual address above the specified base address. For example, on many machines, the data area floats above the text area.

Each element in the array of `addressconf` structures describes a separate area of the process’ address space. These areas have been defined in the `sys/addrconf.h` header file; other areas may be defined in the future or on other machine types. The array elements are indexed with the following constants:

AC_TEXT
The area that normally contains the text region of an absolute executable program.

AC_DATA
The area that normally contains the data region of an absolute executable program.

AC_BSS
The area that normally contains the bss region of an absolute executable program.

AC_STACK
The area that normally contains the process’ user-mode stack.

AC_LDR_TEXT
The area reserved for the text region of the default program loader see the `exec_with_loader()` function.

AC_LDR_DATA
The area reserved for the data region of the default program loader.

AC_LDR_BSS
The area reserved for the bss region of the default program loader.

AC_LDR_PRIV
The area that normally contains the default program loader’s private keep-on-exec data. See the `mmap()` function.

AC_LDR_GLB
The area that normally contains the default program loader’s Global Installed Package tables. See the `libadmin` administrative command.

AC_LDR_PRELOAD
The area that normally contains the text, data, and bss regions of the preloaded shared libraries.
getaddressconf(2)

AC_MMAP_TEXT
The area that normally contains text regions of relocatable files
loaded by the program loader, or otherwise mapped using the
\texttt{mmap()} function.

AC_MMAP_DATA
The area that normally contains data regions of relocatable files
loaded by the program loader, or otherwise mapped using the
\texttt{mmap()} function.

AC_MMAP_BSS
The area that normally contains the bss regions of relocatable files
loaded by the program loader, or anonymous regions mapped using
the \texttt{mmap()} function.

The \texttt{sys/addrconf.h} header file also defines the AC_N_AREAS symbol to be the
number of distinct areas currently defined for this system. Normally, the \texttt{buffer}
parameter supplied to the \texttt{getaddressconf()} function should be large enough to
hold information for AC_N_AREAS regions. If \texttt{buffer} is not large enough, the
remaining information is truncated. The \texttt{getaddressconf()} call fills in the first
AC_N_AREAS records in the user-supplied buffer with the address configuration
information for this system, as described above.

Return Values

Upon successful completion, the number of bytes actually written to the user’s
buffer is returned. If an error occurs, -1 is returned, and \texttt{errno} is set to indicate the
error.

Errors

If the \texttt{getaddressconf()} function fails, \texttt{errno} may be set to the following value:
\begin{itemize}
  \item \texttt{EFAULT} The address specified for \texttt{buffer} is not valid.
\end{itemize}

Related Information

Functions: \texttt{mmap(2), exec(2), exec_with_loader(2), brk(2)}
Commands: \texttt{libadmin(8)}
getc, fgetc, getchar, getw

**Purpose**  
Gets a character or word from an input stream

**Library**  
Standard I/O Package (libc.a)

**Synopsis**  
```c
#include <stdio.h>

int getc (  
    FILE *stream );
int fgetc (  
    FILE *stream );
int getchar ( void );
int getw (  
    FILE *stream );
```

**Parameters**  

- `stream`  
  Points to the file structure of an open file.

**Description**  
The `getc()` macro returns the next byte from the input specified by the `stream` parameter and moves the file pointer, if defined, ahead one byte in `stream`. The `getc()` macro cannot be used where a function is necessary; for example, a subroutine pointer cannot point to it.

Because it is implemented as a macro, `getc()` does not work correctly with a `stream` parameter that has side effects. In particular, the following does not work:

```c
getc(*f++)
```

In cases like this, use the `fgetc()` function instead.

The `fgetc()` function performs the same function as the `getc()` macro, but `fgetc()` is a subroutine, not a macro.

The `getchar()` macro returns the next byte from `stdin`, the standard input stream. Note that `getchar()` is also a macro.

The `getw()` function returns the next word (int) from the input specified by the `stream` parameter and increments the associated file pointer, if defined, to point to the next word. The size of a word varies from one machine architecture to another.
The `getw()` function returns the constant EOF at the end of the file or when an error occurs. Since EOF is a valid integer value, the `feof()` and `ferror()` functions can be used to check the success of `getw()`. The `getw()` function assumes no special alignment in the file.

Because of possible differences in word length and byte ordering from one machine architecture to another, files written using the `putw()` subroutine are machine dependent and may not be readable using `getw()` on a different type of processor.

**Notes**

The reentrant versions of these functions are all locked against multiple threads calling them simultaneously. This will incur an overhead to ensure integrity of the stream. The unlocked versions of these calls may be used safely, providing that the stream is locked when the calls are used, using the `flockfile()` and `funlockfile()` functions.

**AES Support Level:** Full use (`getc()`, `fgetc()`, `getchar()`)  
Trial use (`getw()`)  

**Return Values**

These functions and macros return the integer constant EOF at the end of the file or upon an error.

**Related Information**

Functions: `gets(3)`, `getwc(3)`, `putc(3)`, `unlocked_getc(3)`, `unlocked_getchar(3)`
Functions

getclock(3)

getclock

Purpose
Gets current value of system-wide clock

Library
Standard C Library (libc.a)

Synopsis
#include <sys/timers.h>

int getclock(
    int clktyp,
    struct timespec *tp);

Parameters

clktyp Identifies a system-wide clock.

tp Points to a timespec structure space where the current value of the
      system-wide clock is stored.

Description

The getclock() function sets the current value of the clock specified by clktyp into
the location pointed to by the tp parameter.

The clktyp parameter is given as a symbolic constant name, as defined in the
sys/timers.h include file. Only the TIMEOFDAY symbolic constant, which
specifies the normal time-of-day clock to access for system-wide time, is
supported.

For the clock specified by TIMEOFDAY, the value returned by this function is the
elapsed time since the epoch. The epoch is referenced to 00:00:00 CUT
(Coordinated Universal Time) 1 Jan 1970.

The getclock() function returns a timespec structure, which is defined in the
sys/timers.h header file. It has the following members:

unsigned long tv_sec Elapsed time in seconds since the epoch
long tv_nsec Elapsed time as a fraction of a second since
    the epoch (expressed in nanoseconds)
getclock(3)

The time interval expressed by the members of this structure is \(((tv\_sec * 10^9) + (tv\_nsec))\) nanoseconds.

Notes

AES Support Level: Trial use

Return Values

Upon successful completion, the getclock() function returns a value of 0 (zero). Otherwise, getclock() returns a value of -1 and sets errno to indicate the error.

Errors

If the getclock() function fails, errno may be set to one of the following values:

- [EINVAL] The clktyp parameter does not specify a known system-wide clock.
- [EIO] An error occurred when the system-wide clock specified by the clktyp parameter was accessed.

Related Information

Functions: gettimer(3), setclock(3), time(3)
getcwd

Purpose
Gets the pathname of the current directory

Library
Standard C Library (libc.a)

Synopsis
char *getcwd (char *buffer, int size);

Parameters
buffer Points to a string space to hold the pathname. If the buffer parameter is a null pointer, the getcwd() function, using the malloc() function, obtains the number of bytes of free space as specified by the size parameter. In this case, the pointer returned by the getcwd() function can be used as the parameter in a subsequent call to the free() function.

size Specifies the length of the string space in bytes. The value of the size parameter must be at least the length of the pathname to be returned plus one byte for the terminating null.

Description
The getcwd() function returns a pointer to a string containing the pathname of the current directory. The getwd function is called to obtain the pathname.

Notes
The getcwd() function is supported for multi-threaded applications.

AES Support Level: Full use

Return Values
Upon successful completion, the buffer parameter is returned. Otherwise, a null value is returned and errno is set to indicate the error.
Errors

If the `getcwd()` function fails, `errno` may be set to one of the following values:

- `[EINVAL]` The `size` parameter is zero or negative.
- `[ERANGE]` The `size` parameter is greater than zero, but is smaller than the length of the pathname + 1.
- `[ENOMEM]` The requested amount of memory could not be allocated.

Related Information

Functions: `malloc(3)`, `getwd(3)`
getdirentries

Purpose
Gets directory entries in a file-system independent format

Synopsis
#include <dirent.h>
int getdirentries(
    int fd,
    char *buf,
    int nbytes,
    long *basep);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>Specifies the file descriptor of a directory to be read.</td>
</tr>
<tr>
<td>buf</td>
<td>Points to a buffer containing the directory entries as dirent structures.</td>
</tr>
<tr>
<td>nbytes</td>
<td>Specifies the maximum amount of data to be transferred, in bytes.</td>
</tr>
<tr>
<td>basep</td>
<td>Points to the position of the block read.</td>
</tr>
</tbody>
</table>

Description
The getdirentries() function reads directory entries from a directory into a buffer. The entries are returned as dirent structures, a file-system independent format.

The nbytes parameter must be greater than or equal to the block size associated with the file (see the stat() function). Some file systems may not support the getdirentries() function with buffers smaller than this size.

The entries returned by the getdirentries() function into the location pointed to by buf may be separated by extra space.

The getdirentries() function writes the position of the block read into the location pointed to by the basep parameter. Alternatively, the current position pointer may be set and retrieved by lseek(). The current position pointer should only be set to a value returned by lseek(), a value returned in the location pointed to by basep, or 0 (zero).
Upon successful completion, the actual number of bytes transferred is returned and the current position pointer associated with the \textit{fd} parameter is set to point to the next block of entries. The file descriptor pointer may not advance by the same number of bytes returned by the \texttt{getdirentries}() function. A value of 0 (zero) is returned when the end of the directory has been reached.

**Return Values**

Upon successful completion, the actual number of bytes transferred is returned. Otherwise, -1 is returned and \texttt{errno} is set to indicate the error.

**Errors**

If the \texttt{getdirentries}() function fails, \texttt{errno} may be set to one of the following values:

- \texttt{[EBADF]}: The \textit{fd} parameter is not a valid file descriptor open for reading.
- \texttt{[EFAULT]}: Either the \textit{buf} or \textit{basep} parameter point outside the allocated address space.
- \texttt{[EINVAL]}: The \textit{fd} parameter is not a valid file descriptor for a directory.
- \texttt{[EIO]}: An I/O error occurred while reading from or writing to the file system.

**Related Information**

Functions: \texttt{open(2)}, \texttt{lseek(2)}
getdiskbyname

**Purpose**  
Gets disk description using a disk name

**Library**  
Standard C Library (libc.a)

**Synopsis**  
```c  
#include <sys/disklabel.h>  
struct disklabel *getdiskbyname(  
    char *name) ;  
```

**Parameters**  

- **name**  
  Specifies a common name for the disk drive whose geometry and partition characteristics are sought.

**Description**  
The `getdiskbyname()` function uses a disk (diskdrive) name to return a pointer to a structure that describes the geometry and standard partition characteristics of the named disk drive. Information obtained from the `/etc/disktab` database file is written to the type `disklabel` structure space referenced by the returned pointer.

**Return Values**  
Upon successful completion, a pointer to a type `disklabel` structure is returned.

**Related Information**  
Files: `disklabel(4), disktab(4)`  
Commands: `disklabel(8)`
getdtablesize

Purpose  Gets the descriptor table size

Synopsis  int getdtablesize ( void );

Description

The getdtablesize( ) function returns the total number of file descriptors in a
process’ descriptor table. Each process has a fixed size descriptor table that is
guaranteed to have at least 64 slots. The entries in the descriptor table are
numbered with small integers starting at 0 (zero).

Return Values

The getdtablesize( ) function returns the size of the descriptor table, and is always
successful.

Related Information

Functions: close(2), open(2), select(2)
getenv

Purpose
Returns the value of an environment variable

Library
Standard C Library (libc.a)

Synopsis
#include <stdlib.h>

char *getenv (const char *name);

Parameters

name Specifies the name of an environment variable.

Description
The getenv() function searches the environment list for a string of the form name=value, and returns a pointer to a string containing the corresponding value for name.

Notes

AES Support Level: Full use

Return Values
The getenv() function returns a pointer to a string containing the value in the current environment if such a string is present. If such a string is not present, a null pointer is returned.

The returned string should not be modified by the application, and may be overwritten or changed as a result of the putenv(), setenv(), or unsetenv() functions.

Related Information
Functions: putenv(3), clearenv(3)
Commands: sh(1)
getfh

Purpose
Gets a file handle

Synopsis
#include <sys/types.h>
#include <sys/mount.h>
getfh(
    char *path,
    struct fhandle_t *fhp);

Parameters
path Points to the file.
fhp Points to a fhandle_t structure.

Description
The getfh() function returns a file handle for the specified file or directory in the file handle pointed to by the fhp parameter. This function is restricted to the superuser.

Return Values
Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.

Errors
If the getfh() function fails, errno may be set to one of the following values:

[ENOTDIR] A component of the path prefix of the path parameter is not a directory.

[EINVAL] The path parameter contains a character with the high-order bit set.

[ENAMETOOLONG] The length of a component of the pathname parameter exceeds NAME_MAX characters, or the length of the path parameter exceeds PATH_MAX characters.

[ENOENT] The file referred to by the path parameter does not exist.

[EACCES] Search permission is denied for a component of the path prefix of the path parameter.
Functions

getfh(2)

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ELOOP]</td>
<td>Too many symbolic links were encountered in translating the <em>path</em> parameter.</td>
</tr>
<tr>
<td>[EFAULT]</td>
<td>The <em>fhp</em> parameter points to an invalid address.</td>
</tr>
<tr>
<td>[EIO]</td>
<td>An I/O error occurred while reading from or writing to the file system.</td>
</tr>
<tr>
<td>[EPERM]</td>
<td>The calling process does not have appropriate privilege.</td>
</tr>
</tbody>
</table>
getfsent(3)

getfsent, getfsspec, getfsfile, getfstype, setfsent, endfsent

Purpose

Gets information about a file system

Library

Standard C Library (libc.a)

Synopsis

#include <fstab.h>
struct fstab *getfsent( void);
struct fstab *getfsspec ( char *spec_file );
struct fstab *getfsfile( char *fs_file );
struct fstab *getfstype( char *fs_type );
void setfsent( void );
void endfsent( void );

Parameters

spec_file Specifies the special filename.
fs_file Specifies the file system filename.
fs_type Specifies the file system type.

Description

The getfsent( ) function reads the next line of the file, opening the file if necessary.
The setfsent( ) function opens the file and positions to the first record.
The endfsent( ) function closes the file.
The getfsspec( ) function sequentially searches from the beginning of the file until a matching special filename is found, or until the end of the file is encountered.
The getfsfile( ) function sequentially searches from the beginning of the file until a matching file system filename is found, or until the end of the file is encountered.
The getfstype( ) function sequentially searches from the beginning of the file until a matching file system type is found, or until the end of the file is encountered.
Notes

All information is contained in a static area, so it must be copied if it is to be saved.

Return Values

Upon successful completion, the `getfsent()`, `getfspec()`, `getfstype()`, and `getsfile()` functions return a pointer to a structure that contains information about a file system, defined in the `fstab.h` file. A pointer to null is returned on EOF (End-of-File) or error.
getfsstat

Purpose
Gets list of all mounted file systems

Synopsis
#include <sys/types.h>
#include <sys/mount.h>

getfsstat(  
    struct statfs *buf[],
    long bufsize,
    int flags);

Parameters
buf
Points to an array of statfs structures.

bufsize
Specifies the size in bytes of the buf parameter.

flags
Specifies one of the following flags:

MNT_WAIT
Wait for an update from each file system before returning information.

MNT_NOWAIT
Information is returned without requesting an update from each file system. Thus, some of the information will be out of date, but the getfsstat() function will not block waiting for information from a file system that is unable to respond.

Description
The getfsstat() function returns information about all mounted file systems. Upon successful completion, the buffer pointed to by the buf parameter is filled with an array of statfs structures, one for each mounted file system up to the size specified by the bufsize parameter.

If the buf parameter is given as 0 (zero), the getfsstat() function returns just the number of mounted file systems.
Return Value
Upon successful completion, the number of statfs structures is returned. Otherwise, -1 is returned and errno is set to indicate the error.

Errors
If the getfsstat() function fails, errno may be set to one of the following values:
[EFAULT] The buf parameter points to an invalid address.
[EIO] An I/O error occurred while reading from or writing to the file system.

Related Information
Functions: statfs(2)
Commands: mount(8)
getgid(2)

getgid, getegid

Purpose
Gets the process group IDs

Synopsis
#include <sys/types.h>

gid_t getgid ( void );
gid_t getegid ( void );

Description
The getgid() function returns the real group ID of the calling process.
The getegid() function returns the effective group ID of the calling process.
The real group ID is specified at login time. The effective group ID is more transient, and determines additional access permission during execution of a "set-group-ID" process. It is for such processes that the getgid() function is most useful.

Notes
AES Support Level: Full use

Return Values
The getgid() and getegid() functions return the requested group ID. They are always successful.

Related Information
Functions: getgroups(2), initgroups(3), setgroups(2), setregid(2)
Commands: groups(1)
getgrent, getgrgid, getgrnam, setgrent, endgrent

**Purpose**
Accesses the basic group information in the user database

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <grp.h>

struct group *getgrent ( void )

struct group *getgrgid (  
    gid_t gid);

struct group *getgrgid_r (  
    struct group *result,  
    gid_t gid,  
    char *buffer,  
    int len);

struct group *getgrnam (  
    const char *name);

struct group *getgrnam_r (  
    struct group result,  
    const char *name,  
    char *buffer,  
    int len);

void setgrent ( void )

void endgrent ( void )
```

**Parameters**

- **name**
  Specifies the name of the group for which the basic attributes are to be read.

- **gid**
  Specifies the group ID of the group for which the basic attributes are to be read.

- **result**
  Points to a buffer containing the result.
**buffer**

Points to a character array to contain the strings associated with the entry returned by the `getpwnam_r()` or `getpwuid_r()` functions.

**len**

Specifies the length of `buffer`.

---

**Description**

The `getgrent()`, `getgrgid()`, `getgrnam()`, `setgrent()`, and `endgrent()` functions may be used to access the basic group attributes. These attributes can also be accessed with the `getgroupattr()` function, which can access all group attributes and offer better granularity of access.

The `setgrent()` function opens the user database (if not already open) and rewinds the cursor to point to the first group entry in the database.

The `getgrent()`, `getgrnam()`, and `getgrgid()` functions return information about the requested group. The `getgrent()` function returns the next group in the sequential search. The `getgrnam()` function returns the first group in the database with the `gr_name` field that matches the `name` parameter. The `getgrgid()` function returns the first group in the database with a `gr_gid` field that matches the `gid` parameter. The `endgrent()` function closes the user database.

The `group` structure, which is returned by the `getgrent()`, `getgrnam()`, and `getgrgid()` functions, is defined in the `grp.h` header file, and contains the following members:

- **gr_name**
  The name of the group.

- **gr_passwd**
  The password of the group. (Note that this field is no longer used by the system, so its value is meaningless.)

- **gr_gid**
  The ID of the group.

- **gr_mem**
  The members of the group.

The `getgrgid_r()` and `getgrnam_r()` functions are the reentrant versions of `getgrgid()` and `getgrnam()`, respectively. Upon successful completion, the result is stored in the buffer pointed to by the `result` parameter.

---

**Notes**

The data that is returned by the `getgrent()`, `getgrnam()`, and `getgrgid()` functions is stored in a static area and will be overwritten on subsequent calls to these routines. If it is to be saved, it should be copied.

**AES Support Level:** Full use (`getgrgid()`, `getgrnam()`)

---

1–220
Return Values

Upon successful completion, the `getgrent()`, `getgrnam()`, and `getgrgid()` functions return a pointer to a valid group structure containing a matching entry. Otherwise, null is returned.

Related Information

Functions: `getpwent(3)`
getgroups

Purpose
Gets the supplementary group set of the current process

Synopsis
#include <unistd.h>
#include <sys/types.h>

int getgroups (  
    int gidsetsize,  
    gid_t grouplist[]);  

Parameters

  gidsetsize  Indicates the number of entries that can be stored in the array  
              pointed to by the grouplist parameter.

  grouplist   Points to the array in which the process' supplementary group set  
              of the user process is stored. Element grouplist[0] is the effective  
              group ID of the process.

Description

The getgroups() function gets the supplementary group set of the process. The list  
is stored in the array pointed to by the grouplist parameter. The gidsetsize  
parameter indicates the number of entries that can be stored in this array.

The getgroups() function never returns more than NGROUPS_MAX entries.  
(NGROUPS_MAX is a constant defined in the limits.h header file.) If the  
gidsetsize parameter is 0 (zero), the getgroups() function returns the number of  
groups in the supplementary group set.

Notes

AES Support Level: Full use

Return Values

Upon successful completion, the getgroups() function returns the number of  
elements stored in the array pointed to by the grouplist parameter. If getgroups()  
fails, then a value of -1 is returned and errno is set to indicate the error.
Errors

If the `getgroups()` function fails, `errno` may be set to one of the following values:

[EFAULT]  The `gidsetsize` and `grouplist` parameters specify an array that is partially or completely outside of the allocated address space of the process.

[EINVAL]  The `gidsetsize` parameter is nonzero and smaller than the number of groups in the supplementary group set.

Related Information

Functions: `setgroups(2), getgid(2), setsid(2), initgroups(3)`

Commands: `groups(1)`
gethostbyaddr

Purpose  Gets network host entry by address

Library  Sockets Library (libc.a)

Synopsis  
#include <netdb.h>
struct hostent *gethostbyaddr (  
    char *addr,  
    int len,  
    int type);  

Parameters  

addr  Specifies an Internet address in network order.
len  Specifies the number of bytes in an Internet address.
type  Specifies the Internet Domain address format. The value AF_INET must be used.

Description  

The gethostbyaddr( ) function searches the hosts network hostname file sequentially until a match with the addr and type parameters occurs. The len parameter must specify the number of bytes in an Internet address. The address parameter must specify the address in network order. The type parameter must be the constant AF_INET, which specifies the Internet address format. When EOF (End-of-File) is reached without a match, an error value is returned.

The gethostbyaddr( ) function returns a pointer to a structure of type hostent. Its members specify data obtained from a name server specified in the /etc/resolv.conf file or from fields of a record line in the /etc/hosts network hostname database file. When the name server is not running, the gethostbyaddr( ) function searches the hosts name file. The hostent structure is defined in the netdb.h header file.

Use the endhostent( ) function to close the /etc/hosts file.
Notes
A return value points to static data, which is overwritten by any subsequently
called functions using the same structure.

Return Values
Upon successful completion, a pointer to a hostent structure is returned. A null
pointer is returned whenever the end of the hosts network hostname file is reached.

Errors
If the gethostbyaddr() function fails, h_errno may be set to one of the following
values:

[TRYAGAIN]
This is a soft error that indicates that the local server did not receive
a response from an authoritative server. A retry at some later time
may be successful.

[NORECOVERY]
This is a nonrecoverable error.

[NOADDRESS]
The address you used is not valid. This is not a soft error, another
type of name server request may be successful.

Files
/etc/hosts This file is the DARPA Internet network hostname database. Each
record in the file occupies a single line and has three fields
consisting of the host address, official host name, and aliases.

Related Information
Functions: gethostent(3), gethostbyname(3), endhostent(3)
gethostbyname

Purpose Gets network host entry by name

Library Sockets Library (libc.a)

Synopsis #include <netdb.h>
struct hostent *gethostbyname (
    char *name );

Parameters

name Specifies the official network name or alias.

Description

The gethostbyname() function returns a pointer to a structure of type hostent. Its members specify data obtained from a name server specified in the /etc/resolv.conf file or from fields of a record line in the /etc/hosts network hostname database file. When the name server is not running, this function searches the hosts name file. The netdb.h header file defines the hostent structure.

The gethostbyname() function searches the hosts network hostname file sequentially until a match with the name parameter occurs. If the environment variable HOSTALIASES is set, the gethostbyname() function first searches the file named by HOSTALIASES. The name parameter must specify the host official name or an alias. When EOF (End-Of-File) is reached without a match, an error value is returned by this function.

Use the endhostent() function to close the /etc/hosts file.

Notes

A return value points to static data, which is overwritten by any subsequently called functions using the same structure.
Functions

gethostbyname(3)

Return Values

Upon successful completion, a pointer to a hostent structure is returned. A null pointer is returned whenever the end of the hosts network hostname file is reached.

Errors

If the gethostbyname() function fails, h_errno may be set to one of the following values:

[TRY_AGAIN]
This is a soft error that indicates that the local server did not receive a response from an authoritative server. A retry at some later time may be successful.

[NO_RECOVERY]
This is a nonrecoverable error.

[HOST_NOT_FOUND]
The name you have used is not an official hostname or alias; this is not a soft error, another type of name server request may be successful.

[NO_ADDRESS]
The requested name is valid but does not have an Internet address at the name server.

Files

/etc/hosts The DARPA Internet network hostname database. Each record in the file occupies a single line and has three fields consisting of the host address, official hostname, and aliases.

/etc/resolv.conf Contains the name server and domain name.

Related Information

Functions: gethostent(3). gethostbyaddr(3). endhostent(3)
Files: hostname(5)
gethostent, sethostent

Purpose
Opens network host file

Library
Standard C Library (libc.a)

Synopsis
#include <netdb.h>
struct hostent *gethostent ( void );
void sethostent ( int stay_open );

Parameters
stay_open 
Contains a value used to indicate when to close the host file. Specifying a value of 0 (zero) closes the /etc/hosts file after each call to the gethostbyname() or gethostbyaddr() function. Specifying a nonzero value allows the /etc/hosts file to remain open after each call.

Description
The gethostent() (get host entry) function reads the next line of the /etc/hosts file, opening it if necessary.

The sethostent() (set host entry) function opens the /etc/hosts file and resets the file marker to the beginning of the file.

Passing a nonzero value to the stay_open parameter establishes a connection with a name server and allows a client process to retrieve one entry at a time from the /etc/hosts file. The client process can close the connection with the endhostent() function.
Return Values

If an error occurs or if the end of the file is reached, the `sethostent()` function returns a null pointer to the calling program and an error code, indicating the specific error, is moved into the `h_errno` variable. The calling program must examine `h_errno` to determine the error.

Errors

If the `sethostent()` function fails, `h_errno` may be set to the following value:

[NO_RECOVERY]

This error code indicates an unrecoverable error.

Files

/etc/hosts Contains the hostname database.
/etc/resolv.conf
    Contains the name server and domain name.

Related Information

Functions: `endhostent(3), gethostbyaddr(3), gethostbyname(3)`
gethostid

Purpose       Gets the unique identifier of the current host

Synopsis      int gethostid ( void );

Description   The gethostid( ) function allows a process to retrieve the 32-bit identifier for the current host. In most cases, the host ID is stored in network standard byte order and is a DARPA Internet address for the local machine.

Return Values

Upon completion, the gethostid() function returns the identifier for the current host.

Related Information

Functions: gethostname(2), sethostname(2)
gethostname

Purpose

Gets the name of the local host

Synopsis

int gethostname (  
          char *address,  
          int address_len );

Parameters

address     Returns the address of an array of bytes where the hostname is stored.
address_len Specifies the length of the array pointed to by the address parameter.

Description

The gethostname() function retrieves the standard host name of the local host. If sufficient space is provided, the returned address parameter is null-terminated.

System hostnames are limited to MAXHOSTNAMELEN as defined in the /usr/include/sys/param.h file.

The gethostname() function allows a calling process to determine the internal hostname for a machine on a network.

Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the gethostname() function fails, errno may be set to the following value:

[EFAULT] The address parameter or address_len parameter gives an invalid address.

Related Information

Functions: gethostid(2), sethostid(2), sethostname(2)
setitimer, getitimer

Purpose

Sets or returns the value of interval timers

Synopsis

`
#include <sys/time.h>
#define ITIMER_REAL 0
#define ITIMER_VIRTUAL 1
#define ITIMER_PROF 2

int setitimer(
    int which,
    struct itimerval *value,
    struct itimerval *ovalue);

int getitimer(
    int which,
    struct itimerval *value);
`

Parameters

which  Identifies the interval timer. This parameter may be expressed as one of three symbolic constants: ITIMER_REAL, ITIMER_VIRTUAL, and ITIMER_PROF.

value  Points to an itimerval structure whose members specify a timer interval and the time left to the end of the interval.

ovalue  Points to an itimerval structure whose members specify a current timer interval and the time left to the end of the interval.

Description

The getitimer() function returns the current value for the timer specified by the which parameter in the structure pointed to by the value parameter.

The setitimer() function sets a timer to the specified value (returning the previous value of the timer if ovalue is nonzero).

A timer value is defined by the itimerval structure:

```c
struct itimerval {
    struct timeval it_interval;
    struct timeval it_value;
};
```
If the \texttt{it\_value} field is nonzero, it indicates the time to the next timer expiration. If the \texttt{it\_interval} field is nonzero, it specifies a value to be used in reloading \texttt{it\_value} when the timer expires. Setting \texttt{it\_value} to 0 (zero) disables a timer. Setting \texttt{it\_interval} to 0 causes a timer to be disabled after its next expiration (assuming \texttt{it\_value} is nonzero).

Time values smaller than the resolution of the system clock are rounded up to this resolution.

The system provides each process with three interval timers, defined in the \texttt{sys/time.h} header file:

- The \texttt{ITIMER\_REAL} timer decrements in real time. A SIGALRM signal is delivered when this timer expires.
- The \texttt{ITIMER\_VIRTUAL} timer decrements in process virtual time. It runs only when the process is executing. A SIGVTALRM signal is delivered when it expires.
- The \texttt{ITIMER\_PROF} timer decrements both in process virtual time and when the system is running on behalf of the process. It is designed to be used by interpreters in statistically profiling the execution of interpreted programs. Each time the \texttt{ITIMER\_PROF} timer expires, the SIGPROF signal is delivered. Because this signal may interrupt in-progress system calls, programs using this timer must be prepared to restart interrupted system calls.

Notes

Three macros for manipulating time values are defined in the \texttt{sys/time.h} header file. The \texttt{timerclear()} macro sets a time value to zero, the \texttt{timerisset()} macro tests if a time value is nonzero, and the \texttt{timercmp()} macro compares two time values. Beware that the comparisons \texttt{>=} and \texttt{<=} do not work with the \texttt{timercmp()} macro.

Return Values

Upon successful completion, the value 0 (zero) is returned. Otherwise, -1 is returned and \texttt{errno} is set to indicate the error.
getitimer(2)

Errors

If the getitimer() or setitimer() function fails, errno may be set to one of the following values:

[EFAULT] The value parameter specified a bad address.

[EINVAL] The value parameter specified a time that was too large to be handled.

Related Information

Functions: gettimeofday(2)
getlogin, getlogin_r, setlogin

Purpose

Gets and sets login name

Synopsis

char *getlogin( void );

int getlogin_r(
    char *name,
    int len );

setlogin ( 
    char *name );

Parameters

name Points to the login name.

len Specifies the length of the buffer pointed to by name.

Description

The getlogin() function returns the login name of the user associated with the current session, as previously set by the setlogin() function. The name is normally associated with a login shell at the time a session is created, and is inherited by all processes descended from the login shell. (This is true even if some of those processes assume another user ID, for example when the su command is used.)

The setlogin() function sets the login name of the user associated with the current session to name. This call is restricted to the superuser, and is normally used only when a new session is being created on behalf of the named user (for example, at login time, or when a remote shell is invoked).

The getlogin_r() function is the reentrant version of getlogin(). Upon successful completion, the login name is stored in name.

Notes

AES Support Level: Full use (getlogin())

Return Values

Upon successful completion, the getlogin() function returns a pointer to a null-terminated string in a static buffer. If the name has not been set, it returns null.

Upon successful completion, the setlogin() function returns a value of 0 (zero). If setlogin() fails, then a value of -1 is returned and an error code is placed in errno.
Upon successful completion, the `getlogin_r()` function returns a value of 0 (zero). Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors

If the `getlogin()`, `getlogin_r()`, or `setlogin()` function fails, `errno` may be set to one of the following values:

- `[EFAULT]` The `name` parameter gave an invalid address.
- `[EINVAL]` The `name` parameter pointed to a string that was too long. Login names are limited to MAXLOGNAME characters (defined in `sys/param.h`).
- `[EPERM]` The caller tried to set the login name and was not the superuser.

Related Information

Functions: `setsid(2)`
Command: `su(1)`
_getlong

Purpose
Retrieves long quantities from a byte stream

Library
Standard C Library (libc.a)

Synopsis
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

unsigned long _getlong (u_char *message_ptr);

Parameters
message_ptr
Specifies a pointer into the byte stream.

Description
The _getlong() function gets long quantities from the byte stream or arbitrary byte boundaries.

The _getlong() function is one of a set of subroutines that form the resolver, a set of functions that resolves domain names. Global information that is used by the resolver functions is kept in the _res data structure. The include/resolv.h file contains the _res data structure definition.

Return Values
Upon successful completion, the _getlong() function returns an unsigned long (32-bit) value.
_getlong(3)

Files

/etc/resolv.conf
Defines name server and domain names.

Related Information

Functions: res_init(3), res_mkquery(3), res_send(3), dn_comp(3),
dn_expand(3), dn_find(3), getshort(3), putshort(3), putlong(3),
dn_skipname(3)
getnetbyaddr

Purpose
Gets network entry by address

Library
Sockets Library (libc.a)

Synopsis
#include <netdb.h>

struct netent *getnetbyaddr (
    long net,
    int type);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>net</td>
<td>Specifies the number of the network in host-byte order.</td>
</tr>
<tr>
<td>type</td>
<td>Specifies the Internet Domain address format. The value AF_INET must be used.</td>
</tr>
</tbody>
</table>

Description

The getnetbyaddr( ) function returns a pointer to a structure of type netent. Its members specify data in fields from a record line in the /etc/networks network name database file. The netdb.h header file defines the netent structure.

The getnetbyaddr( ) function searches the networks file sequentially until a match with the net and type parameters occurs. The net parameter must specify the network number in host-byte order. The type parameter must be the constant AF_INET. When EOF (End-of-File) is reached without a match, an error value is returned by this parameter.

Use the endnetent( ) function to close the /etc/networks file.

Notes

The return value points to static data, which is overwritten by any subsequently called functions using the same structure.

Return Values

Upon successful completion, a pointer to a netent structure is returned. A null pointer is returned when an error occurs or when the end of the networks name file is reached.
OSF/1 Programmer's Reference

getnetbyaddr(3)

Files

/etc/networks

The DARPA Internet network-name database. Each record in the file occupies a single line and has three fields consisting of the official service name, network number, and aliases.

Related Information

Functions: getnetent(3), getnetbyname(3), setnetent(3), endnetent(3)
getnetbyname

Purpose         Gets network entry by name

Library         Sockets Library (libc.a)

Synopsis        #include <netdb.h>
                struct netent *getnetbyname (char *name);

Parameters      

name             Specifies the official network name or alias.

Description     

The getnetbyname() function returns a pointer to a structure of type netent. Its 
members specify data in fields from a record line in the /etc/networks network-
name database file. The netdb.h header file defines the protoent structure.

The getnetbyname() function searches the networks file sequentially until a 
match with the name parameter occurs. When EOF (End-of-File) is reached 
without a match, an error value is returned by this function.

Use the endnetent() function to close the /etc/networks file.

Notes           

The return value points to static data, which is overwritten by any subsequently 
called functions using the same structure.

Return Values   

Upon successful completion, a pointer to a servant structure is returned. A null 
pointer is returned when an error occurs or when the end of the networks name file 
is reached.
Files

/etc/networks

This file is the DARPA Internet network-name database. Each record in the file occupies a single line and has three fields consisting of the official service name, network number, and alias.

Related Information

Functions: getnetent(3), getnetbyaddr(3), setnetent(3), endnetent(3)
getnetent

Purpose
Gets network entry

Library
Standard C Library (libc.a)

Synopsis
#include <netdb.h>
struct netent *getnetent ( void);

Description
The getnetent( ) function retrieves network information by opening and sequentially reading the /etc/networks file.

The getnetent( ) function returns a pointer to a netent structure, which contains the equivalent fields for a network description line in the /etc/networks file. The netent structure is defined in the netdb.h header file.

Use the endnetent( ) function to close the /etc/networks file.

Notes
The return value points to static data that is overwritten by subsequent calls.

Return Values
Upon successful completion, the getnetent( ) function returns a pointer to a netent structure. If an error occurs or the end of the file is reached, the getnetent( ) function returns a null (0) pointer.

Files
/etc/networks
Contains official network names.

Related Information
Functions: getnetbyaddr(3), getnetbyname(3), setnetent(3), endnetent(3)
getopt

Purpose

Gets flag letters from the argument vector

Library

Standard C Library (libc.a)

Synopsis

#include <stdio.h>
#include <stdlib.h>

int getopt (  
    int argc,  
    char *argv[],  
    char *optstring );  

extern char *optarg;  
extern int optind;  
extern int opterr;  
extern char optopt;

Parameters

argc Specifies the number of parameters passed to the routine.
argv Points to an array of argc pointers to argument strings.
optstring Specifies a string of recognized option characters. If a character is followed by a colon, the flag is expected to take a parameter that may or may not be separated from it by white space.

Description

The getopt() function returns the next flag character in the argv parameter list that matches a character in the optstring parameter. The getopt() function is used to help programs interpret shell command-line flags that are passed to them.

The optarg external variable is set to point to the start of the flag’s parameter on return from the getopt() function.
The `getopt()` function places the `argv` index of the next argument to be processed in `optind`. The `optind` variable is externally initialized to 1 before the first call to `getopt()` so that `argv[0]` is not processed.

**Notes**

The external `int optopt` variable is set to the real option found in the `argv` parameter. This is true whether the flag is in the `optstring` parameter or not.

**AES Support Level:** Trial use

**Return Values**

Upon successful completion, the `getopt()` function returns the flag character that was detected. If it encounters a flag that is not included in the `optstring` parameter, or if the `:` (colon) character is used incorrectly, the `getopt()` function prints an error message on `stderr` and returns a `?` (question mark). The error message can be suppressed by setting the `int` variable `opterr` to 0 (zero).

When all flags have been processed (that is, up to the first nonflag argument), the `getopt()` function returns EOF. The special flag `--` (dash dash) can be used to delimit the end of the flags; EOF is returned, and the `--` string is skipped.

**Related Information**

Commands: `getopt(1)`
getpagesize

**Purpose**  Gets the system page size

**Synopsis**  int getpagesize ( void );

**Description**

The `getpagesize()` function returns the number of bytes in a page. Knowing the system page size is useful for specifying arguments to memory management system calls.

The page size is a system page size and may not be the same as the underlying hardware page size.

**Return Values**

The `getpagesize()` function returns the number of bytes in a page, and is always successful.

**Related Information**

Functions: `brk(2)`, `getrlimit(2)`, `mmap(2)`, `mprotect(2)`, `munmap(2)`, `sysconf(3)`, `madvise(2)`, `msync(2)`
getpass

Purpose
Reads a password

Library
Standard C Library (libc.a)

Synopsis
#include <stdlib.h>
char *getpass (const char *prompt);

Parameters
prompt Points to the prompt string that is written to stderr.

Description
The getpass() function opens the /dev/tty file, flushes output, disables echoing, and reads up to a newline character or an End-of-File character from the /dev/tty file. The terminal state is then restored and /dev/tty is closed.

If the getpass() function is interrupted by the SIGINT signal, the terminal state of /dev/tty will be restored before the signal is delivered to the calling process.

Notes
AES Support Level: Trial use

Return Values
Upon successful completion, the getpass() function returns a pointer to a null-terminated string of no more than PASS_MAX characters. This return value points to data that is overwritten by successive calls. If the /dev/tty file cannot be opened, a NULL pointer is returned.
getpass(3)

Files

/dev/tty Specifies the tty device special file.

Related Information

Files: tty(7) termios(4)
getpeername

**Purpose**
Gets the name of the peer socket

**Synopsis**
```
#include<sys/types.h>
#include <sys/socket.h>
int getpeername (
    int socket,
    struct sockaddr *address,
    int *address_len);
```

**Parameters**
- `socket` Specifies the descriptor number of a connected socket.
- `address` Points to a `sockaddr` structure, the format of which is determined by the domain and by the behavior requested for the socket. The `sockaddr` structure is an overlay for a `sockaddr_in`, `sockaddr_un`, or `sockaddr_ns` structure, depending on which of the supported address families is active. If the compile-time option `_SOCKADDR_LEN` is defined before the `sys/socket.h` header file is included, the `sockaddr` structure takes 4.4BSD behavior, with a field for specifying the length of the socket address. Otherwise, the default 4.3BSD `sockaddr` structure is used, with the length of the socket address assumed to be 14 bytes or less.
  
  If `_SOCKADDR_LEN` is defined, the 4.3BSD `sockaddr` structure is defined with the name `osockaddr`.
- `address_len` Specifies the length of the `sockaddr` structure pointed to by the `address` parameter.

**Description**
The `getpeername()` function retrieves the name of the peer socket connected to the specified socket.
getpeername(2)

A process created by another process can inherit open sockets, but may need to identify the addresses of the sockets it has inherited. The getpeername() function allows a process to retrieve the address of the peer socket at the remote end of the socket connection.

Notes

The getpeername() function operates only on connected sockets.
A process can use the getsockname() function to retrieve the local address of a socket.

Return Values

Upon successful completion, a value of 0 (zero) is returned and the address parameter holds the address of the peer socket. If the getpeername() function fails, a value of -1 is returned and errno is set to indicate the error.

Errors

If the getpeername() function fails, errno may be set to one of the following values:

[EBADF] The socket parameter is not valid.
[ENOTSOCK] The socket parameter refers to a file, not a socket.
[ENOTCONN] The socket is not connected.
[ENOBUFS] Insufficient resources were available in the system to complete the call.
[EFAULT] The address or address_len parameter is not in a writable part of the user address space.

Related Information

Functions: accept(2), bind(2), getsockname(2), socket(2)
### getpid, getpgrp, getppid

<table>
<thead>
<tr>
<th><strong>Purpose</strong></th>
<th>Gets the process ID, process group ID, parent process ID</th>
</tr>
</thead>
</table>
| **Synopsis** | ```
#include <unistd.h>
#include <sys/types.h>

pid_t getpid( void );
pid_t getpgrp( void );
pid_t getppid( void );
``` |

**Description**

The `getpid()` function returns the process ID of the calling process.

The `getpgrp()` function returns the process group ID of the calling process.

The `getppid()` function returns the parent process ID of the calling process. When a process is created, its parent process ID is the process ID of its parent process. If a parent process exits, the parent process IDs of its child processes are changed to the process ID of `init`.

**Notes**

AES Support Level: Full use

**Related Information**

System calls: `fork(2)`, `kill(2)`, `setpgid(2)`, `setsid(2)`, `wait(2)`
getpriority, setpriority

**Purpose**  
Gets or sets process scheduling priority

**Synopsis**  
#include <sys/resource.h>

```c
int getpriority(
    int which,
    int who );

int setpriority(
    int which,
    int who,
    int priority );
```

**Parameters**

- **which**  
  Specifies one of PRIO_PROCESS, PRIO_PGRP, or PRIO_USER.

- **who**  
  Specifies a numeric value interpreted relative to the `which` parameter (a process identifier, process group identifier, and a user ID, respectively). A 0 (zero) value for the `who` parameter denotes the current process, process group, or user.

- **priority**  
  Specifies a value in the range -20 to 20. The default `priority` is 0 (zero); negative priorities cause more favorable scheduling.

**Description**

The `setpriority()` function sets the scheduling priority of a process, process group, or user. The `getpriority()` function obtains the current priority of a process, process group, or user.

The `getpriority()` function returns the highest priority (lowest numerical value) pertaining to any of the specified processes. The `setpriority()` function sets the priorities of all of the specified processes to the specified value. If the specified value is less than -20, a value of -20 is used; if it is greater than 20, a value of 20 is used.
Return Values

Upon successful completion, the `getpriority()` function returns an integer in the range -20 to 20. Otherwise, -1 is returned.

Upon successful completion, the `setpriority()` function returns 0 (zero). Otherwise, -1 is returned.

Errors

If the `getpriority()` or `setpriority()` function fails, `errno` may be set to one of the following values:

- **[ESRCH]** No process was located using the `which` and `who` parameter values specified.
- **[EINVAL]** The `which` parameter was not recognized.

In addition to the errors indicated above, the `setpriority()` function can fail with `errno` set to one of the following values:

Related Information

Functions: `exec(2)`, `nice(3)`
getprotobynname

Purpose
Gets protocol entry by protocol name

Library
Sockets Library (libc.a)

Synopsis
#include <netdb.h>
struct protoent *getprotobynname ( 
    char *name );

Parameters
name Specifies the official protocol name or alias.

Description
The getprotobynname() function returns a pointer to a structure of type protoent.
Its members specify data in fields from a record line in the /etc/protocols network
protocols database file. The netdb.h header file defines the protoent structure.

The getprotobynname() function searches the protocols file sequentially until a
match with the name parameter occurs. The name parameter may specify either the
official protocol name or an alias. When EOF (End-of-File) is reached without a
match, an error value is returned by this function.

Use the endprotoent() function to close the protocols file.

Notes
The return value points to static data, which is overwritten by any subsequently
called functions using the same structure.

Return Values
Upon successful completion, a pointer to a protoent structure is returned. A null
pointer is returned when an error occurs or when the end of the protocols file is
reached.
Files

/etc/protocols

The DARPA Internet network protocols name database. Each record in the file occupies a single line and has three fields consisting of the official protocol name, protocol number, and protocol alias.

Related Information

Functions: getprotobynumber(3), getprotoent(3), setprotoent(3), endprotoent(3)
getprotobynumber

Purpose
Gets a protocol entry by number

Library
Sockets Library (libc.a)

Synopsis
#include <netdb.h>
struct protoent *getprotobynumber (int proto);

Parameters
proto Specifies the protocol number.

Description
The getprotobynumber() function returns a pointer to a structure of type protoent. Its members specify data in fields from a record line in the /etc/protocols network protocols database file. The netdb.h header file defines the protoent structure.

The getprotobynumber() function searches the protocol file sequentially until a match with the proto parameter occurs. The proto parameter must specify the official protocol number. When EOF (End-Of-File) is reached without a match, an error value is returned by this function.

Use the endprotoent() function to close the protocols file.

Notes
The return value points to static data, which is overwritten by any subsequently called functions using the same structure.

Return Values
Upon successful completion, a pointer to a protoent structure is returned. A null pointer is returned when an error occurs or whenever the end of the protocols file is reached.
Functions

getprotobynumber(3)

Files

/etc/protocols
The DARPA Internet network protocols name database. Each record in the file occupies a single line and has three fields consisting of the official protocol name, protocol number, and protocol alias.

Related Information

Functions: getprotobynames(3), getprotoent(3), setprotoent(3), endprotoent(3)
getprotoent

**Purpose**  Gets protocol entry from the `/etc/protocols` file

**Library**  Sockets Library (`libc.a`)

**Synopsis**

```c
#include <netdb.h>

struct protoent *getprotoent ( void );
```

**Description**

The `getprotoent()` function retrieves protocol information from the `/etc/protocols` file. The `getprotoent()` function returns a pointer to a `protoent` structure, which contains the fields for a line of information in the `/etc/protocols` file. The `netdb.h` header file defines the `protoent` structure.

An application program can use the `getprotoent()` function to access a protocol name, its aliases, and protocol number. Use the `endprotoent()` function to close the `/etc/protocols` file.

**Notes**

The return value points to static data that is overwritten by subsequent calls.

**Return Values**

Upon successful completion, the `getprotoent()` function returns a pointer to a `protoent` structure.

If an error occurs or the end of the file is reached, the `getprotoent()` function returns a null pointer.

**Files**

`/etc/protocols`

Contains protocol information.

**Related Information**

Functions: `getprotobynumber(3)`, `getprotobyname(3)`, `setprotoent(3)`, `endprotoent(3)`
getpwent, getpwuid, getpwnam, putpwent, setpwent, endpwent

Purpose  Accesses the basic user information in the user database

Library  Standard C Library (libc.a)

Synopsis  
```c
#include <pwd.h>

struct passwd *getpwent ( void )
struct passwd *getpwuid ( uid_t uid );
int *getpwuid_r (  
    struct passwd *result,  
    uid_t uid,  
    char buffer,  
    int len );

int *getpwnam_r (  
    struct passwd *result,  
    const char *name,  
    char buffer,  
    int len );

struct passwd *getpwnam (  
    const char *name );

int putpwent (  
    struct passwd *passwd  
    FILE *file );

void setpwent ( void )
void endpwent ( void )
```

Parameters

- **uid**  Specifies the ID of the user for which the basic attributes are to be read.
- **name**  Specifies the name of the user for which the basic attributes are to be read.
The **getpwent()** function may be used to access the basic user attributes.

The **getpwent()**, **getpwnam()**, **getpwuid()**, **putpwent()**, **setpwent()**, and **endpwent()** functions return information about the specified user. The **getpwent()** function returns the next user entry in the sequential search. The **getpwnam()** function returns the first user entry in the database with a **pw_name** field that matches the **name** parameter. The **getpwuid()** function returns the first user entry in the database with a **pw_uid** field that matches the **u<id** parameter.

The **putpwent()** function writes a password entry into a file in the colon-separated format of the **/etc/passwd** file. Note that the **pw_passwd** field will be written into the corresponding field in the file. If this user’s password is stored in the shadow password file, this field must be an ! (exclamation mark). The password in the shadow file cannot be updated with this function.

The **setpwent()** function insures that the next call to **getpwent()** returns the first entry.

The **endpwent()** function closes the user database.

The user structure, which is returned by the **getpwent()**, **getpwnam()**, and **getpwuid()** functions and which is written by the **putpwent()** function, is defined in the **pwd.h** file and has the following members:

- **pw_name**: The name of the user.
- **pw_passwd**: The encrypted password of the user. If the password is not stored in the **/etc/passwd** file and the invoker does not have access to the shadow file which contains them, this field will contain an unencryptable string, usually an ! (exclamation mark).
- **pw_uid**: The ID of the user.
The getpwuid_r() and getpwnam_r() functions are the reentrant versions of the getpwuid() and getpwnam() functions, respectively. Upon successful completion, the result is stored in two parts. The struct passwd (which includes only pointers) is stored in result, and the strings themselves are stored in buffer.

Notes

All information generated by the getpwent(), getpwnam(), and getpwuid() functions is stored in a static area and will be overwritten on subsequent calls to these routines. If it is to be saved, it should be copied.

AES Support Level: Full use (getpwnam(), getpwuid())

Return Values

Upon successful completion, the getpwent(), getpwnam() and getpwuid() functions return a pointer to a valid password structure. Otherwise, null is returned.

Upon successful completion, the getpwnam_r() and getpwuid_r() functions return a value of 0 (zero). Otherwise, -1 is returned and errno is set to indicate the error.

Errors

If the getpwnam_r() or getpwuid_r() function fails, errno may be set to one of the following values:

[EINVAL] Either the result or buffer parameter is empty.
[ENOENT] The entry could not be found.

Related Information

Functions: getgrent(3)
getrlimit(2)

getrlimit, setrlimit

Purpose Controls maximum system resource consumption

Synopsis #include <sys/time.h>
#include <sys/resource.h>

int setrlimit(
    int resource1,
    struct rlimit *rlp);

int getrlimit (
    int resource1,
    struct rlimit *rlp);

Parameters

resource1 Specifies one of the following values:

 RLIMIT_CPU
    The maximum amount of CPU time (in seconds) to be used
    by each process.

 RLIMIT_FSIZE
    The largest size, in bytes, of any single file that can be
    created.

 RLIMIT_DATA
    The maximum size, in bytes, of the data segment for a
    process; this defines how far a program can extend its break
    with the sbrk() function.

 RLIMIT_STACK
    The maximum size, in bytes, of the stack segment for a
    process; this defines how far a program stack segment can be
    extended. Stack extension is performed automatically by the
    system.
Functions

getrlimit(2)

RLIMIT_CORE
The largest size, in bytes, of a core file that can be created.

RLIMIT_RSS
The maximum size, in bytes, to which a process's resident set size can grow. This imposes a limit on the amount of physical memory to be given to a process; if memory is tight, the system prefers to take memory from processes that are exceeding their declared resident set size.

_points to the rlimit structure, which contains the current soft and hard limits. For the getrlimit() function, the requested limits are returned in this structure, and for the setrlimit() function, the desired new limits are specified here.

Description

The getrlimit() function obtains the limits on the consumption of system resources by the current process and each process it creates. The setrlimit() function is used to set these resources.

Each resource limit is specified as either a soft limit or a hard limit. When a soft limit is exceeded (for example, if the CPU time is exceeded) a process can receive a signal until it reaches the hard limit, or until it modifies its resource limit. The rlimit structure is used to specify the hard and soft limits on a resource, as defined in the sys/resource.h header file.

The calling process must have superuser privilege in order to raise the maximum limits. An unprivileged process can alter the rlim_cur field of the rlimit structure within the range from 0 (zero) to rlim_max or can (irreversibly) lower rlim_max.

An infinite value for a limit is defined as RLIM_INFINITY.

Because this information is stored in the per-process information, the setrlimit() function must be executed directly by the shell in order to affect all future processes created by the shell; limit is thus a built-in command to the shells.

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way: a brk() function fails if the data space limit is reached. When the stack limit is reached, the process receives a SIGSEGV signal; if this signal is not caught by a handler using the signal stack, this signal kills the process. A file I/O operation that would create a file that is too large causes a signal...
getrlimit(2)

SIGXFSZ to be generated; this normally terminates the process, but can be caught. When the soft CPU time limit is exceeded, a signal SIGXCPU is sent to the offending process.

Notes

The ulimit() function is implemented in terms of setrlimit(). Therefore, the two interfaces should not be used in the same program. The result of doing so is undefined.

Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the getrlimit() or setrlimit() function fails, errno may be set to one of the following values:

[EFAULT] The address specified for the rlp parameter is invalid.
[EINVAL] The resource parameter is not a valid resource.
[EPERM] The limit specified to the setrlimit() function would have raised the maximum limit value, and the caller does not have appropriate privilege.

Related Information

Functions: quota(2), setquota(2), sigaction(2), sigstack(2), sigvec(2), ulimit(3)
getrusage, vtimes

Purpose
Gets information about resource utilization

Library
Berkeley Compatibility Library (libbsd.a)
(vtimes() only)

Synopsis
#include <sys/time.h>
#include <sys/resource.h>
int getrusage (
    int who,
    struct rusage *r_usage);
#include <sys/vtimes.h>
vtimes ( 
    struct vtimes *par_vm,
    struct vtimes ch_vm );

Parameters
who Specifies one of the following:

RUSAGE_SELF
    Retrieve information about resources used by the current process.

RUSAGE_CHILDREN
    Retrieve information about resources used by child processes of the current process.

r_usage Points to a buffer that will be filled in as described in the sys/resource.h header file.

Description
The getrusage() function returns information describing the resources utilized by the current process or its terminated child processes.
Notes

The numbers the ru_inblock and ru_outblock fields of the rusage structure account only for real I/O; data supplied by the caching mechanism is charged only to the first process to read or write the data.

The vtimes() function is supported to provide compatibility with older programs. It is superceded by the getrusage() function.

The vtimes() function returns accounting information for the current process and for the terminated child processes of the current process. Either par_vm or ch_vm or both may be zero, in which case only the information for the pointers which are nonzero are returned.

After the call, each buffer contains information as defined by the contents of the sys/vtimes.h include file.

Return Values

Upon successful completion, the getrusage(), function returns 0 (zero). Otherwise, -1 is returned and errno is set to indicate the error.

Errors

If the getrusage() function fails, errno may be set to one of the following values:

[EINVAL] The who parameter is not a valid value.
[EFAULT] The address specified for r_usage is not valid.

Related Information

Functions: gettimer(3), time(3), times(3), wait(2)
gets, fgets

Purpose

Gets a string from a stream

Library

Standard I/O Library (libc.a)

Synopsis

```c
#include <stdio.h>
char *gets ( const char *string );
char *fgets ( const char *string, int n, FILE *stream );
```

Parameters

- `string` Points to a string to receive characters.
- `stream` Points to the `FILE` structure of an open file.
- `n` Specifies an upper bound on the number of characters to read.

Description

The `gets()` function reads characters from the standard input stream, `stdin`, into the array pointed to by the `string` parameter. Data is read until a newline character is read or an End-of-File condition is encountered. If reading is stopped due to a newline character, the newline character is discarded and the string is terminated with a null character.

The `fgets()` function reads characters from the data pointed to by the `stream` parameter into the array pointed to by the `string` parameter. Data is read until the `n-1` characters have been read, until a newline character is read and transferred to `string`, or until an End-of-File condition is encountered. The string is then terminated with a null character.
Notes

AES Support Level: Full use

Return Values

If the end of the file is encountered and no characters have been read, no characters are transferred to string and a null pointer is returned. If a read error occurs, a null pointer is returned. Otherwise, string is returned.

Related Information

**getservbyname**

**Purpose**  
Get service entry by name

**Library**  
Sockets Library (libc.a)

**Synopsis**  
#include <netdb.h>

struct servent *getservbyname (  
    char *name,  
    char *proto );

**Parameters**

- name: Specifies the official name or alias name of the service.
- proto: Specifies the name of the protocol to use when contacting the service.

**Description**

The `getservbyname()` function returns a pointer to a structure of type `servent`. Its members specify data in fields from a record line in the `/etc/services` database file. The `netdb.h` header file defines the `servent` structure.

The `getservbyname()` function searches the `/etc/services` file sequentially until a match with the `name` parameter or with the `proto` parameter occurs. The `name` parameter may specify either the official name or its alias. When EOF (End-of-File) is reached without a match, an error value is returned by this subroutine. When the protocol name is not specified (`proto` parameter is null), the `proto` parameter need not be matched during the `/etc/services` file record search.

The `getservbyname()` function searches the `/etc/services` file sequentially until one of the following occurs:

- A name and protocol number match.
- A name match when the `proto` parameter is set to null.
- The end of the `/etc/services` file is reached.

Use the `endservent()` function to close the `/etc/services` file.
Notes
The return value points to static data, which is overwritten by any subsequently called functions using the same structure.

Return Values
Upon successful completion, a pointer to a servent structure is returned. A null pointer is returned when an error occurs or whenever the end of the /etc/services file is reached.

Files
/etc/services The DARPA Internet network service-name database. Each record in the file occupies a single line and has four fields consisting of the official service name, port reference, protocol name, and alias.

Related Information
Functions: getprotobyname(3), getprotobynumber(3), getprotoent(3), setprotoent(3), endprotoent(3)
getservbyport

Purpose
Gets service entry by port

Library
Sockets Library (libc.a)

Synopsis
#include <netdb.h>

struct servent *getservbyport (  
    int port,  
    char *proto );

Parameters
port Specifies the port number where the service is located.
proto Specifies the protocol name to use when contacting the service.

Description
The getservbyport( ) function returns a pointer to a structure of type servent. Its members specify data in fields from a record line in the /etc/services network services database file. The netdb.h header file defines the servent structure.

The getservbyport( ) function searches the /etc/services file sequentially until a match with the port parameter or with the proto parameter occurs. When used, the proto parameter must specify the /etc/services file protocol name. When a port number is not used (port parameter is null), the port parameter need not be matched during the /etc/services file record search. When EOF (End-of-File) is reached without a match, an error value is returned by this function.

The getservbyport( ) function searches the /etc/services file sequentially until one of the following occurs:

- A port number and protocol name match.
- A protocol name match when the port parameter is set to null.
- The end of the file is reached.

Use the endservent( ) function to close the /etc/services file.
Notes
The return value points to static data, which is overwritten by any subsequently called functions using the same structure.

Return Values
Upon successful completion, a pointer to a servent structure is returned. A null pointer is returned when an error occurs or whenever the end of the /etc/services file is reached.

Files
/etc/services The DARPA Internet network service-name database. Each record in the file occupies a single line and has four fields consisting of the official service name, port number, protocol name, and aliases.

Related Information
Functions: getprotobyname(3), getprotobynumber(3), getprotoent(3), setprotoent(3), endprotoent(3)
getservent

Purpose       Gets services file entry

Library       Standard C Library (libc.a)

Synopsis      #include <netdb.h>
               struct servent *getservent ( void );

Description   The getservent() (get service entry) function opens and reads the next line of the /etc/services file.
               An application program can use the getservent() function to retrieve information about network services and the protocol ports they use.
               The getservent() function returns a pointer to a servent structure, which contains fields for a line of information from the /etc/services file. The servent structure is defined in the netdb.h header file.
               The /etc/services file remains open after a call by the getservent() function. To close the /etc/services file after each call, use the setservent() function. Otherwise, use the endservent() function to close the /etc/services file.

Notes         The return value points to static data that is overwritten by subsequent calls.

Return Values
               Upon successful completion, the getservent() function returns a pointer to a servent structure.
               If an error occurs or the end of the file is reached, the getservent() function returns a null pointer.
**Files**

/etc/services  The DARPA Internet network service-name database. Each record in the file occupies a single line and has four fields consisting of the official service name, port number, protocol name, and aliases.

**Related Information**

Functions: `getservbyport(3)`, `getservbyname(3)`, `endservent(3)`, `setservent(3)`, `getprotoent(3)`, `getprotobynumber(3)`, `getprotobyname(3)`, `setprotoent(3)`, `endprotoent(3)`
_getshort

**Purpose**
Retrieves short quantities from a byte stream

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
unsigned short getshort (
    u_char *message_ptr);
```

**Parameters**
- `message_ptr` Specifies a pointer into the byte stream.

**Description**
The `_getshort()` function gets quantities from the byte stream or arbitrary byte boundaries.

The `_getshort()` function is one of a set of subroutines that form the resolver, a set of functions that resolve domain names. Global information that is used by the resolver functions is kept in the `_res` data structure. The include/resolv.h file contains the `_res` data structure definition.

**Return Values**
Upon successful completion, the `_getshort()` function returns an unsigned short (16-bit) value.
_getshort(3)

Files

/etc/resolv.conf

Defines name server and domain names.

Related Information

Functions: res_init(3), res_mkquery(3), res_send(3), dn_comp(3),
dn_expand(3), dn_find(3), getlong(3), putshort(3), putlong(3), dn_skipname(3)
getsockname

Purpose
Gets the socket name

Synopsis
```
#include <sys/types.h>
#include <sys/socket.h>

int getsockname(
    int socket,
    struct sockaddr *address,
    int *address_len);
```

Parameters
- `socket` Specifies the socket for which the local address is desired.
- `address` Points to a sockaddr structure, the format of which is determined by the domain and by the behavior requested for the socket. The sockaddr structure is an overlay for a sockaddr_in, sockaddr_un, or sockaddr_ns structure, depending on which of the supported address families is active. If the compile-time option _SOCKADDR_LEN is defined before the sys/socket.h header file is included, the sockaddr structure takes 4.4BSD behavior, with a field for specifying the length of the socket address. Otherwise, the default 4.3BSD sockaddr structure is used, with the length of the socket address assumed to be 14 bytes or less.

  If _SOCKADDR_LEN is defined, the 4.3BSD sockaddr structure is defined with the name osockaddr.

- `address_len` Specifies the length of the sockaddr structure pointed to by the address parameter.

Description
The getsockname() function retrieves the locally bound address of the specified socket.

A process created by another process can inherit open sockets. To use the inherited sockets, the created process may need to identify its address. The getsockname() function allows a process to retrieve the local address bound to the specified socket.
A process can use the getpeername() function to determine the address of a destination socket in a socket connection.

Return Values

Upon successful completion, a value of 0 (zero) is returned, and the address_len parameter points to the size of the socket address. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the getsockname() function fails, errno may be set to one of the following values:

[EBADF] The socket parameter is not valid.

[ENOTSOCK] The socket parameter refers to a file, not a socket.

[ENOBUFS] Insufficient resources are available in the system to complete the call.

[EFAULT] The address or address_len parameter is not in a writable part of the user address space.

Related Information

Functions: accept(2), bind(2), getpeername(2), socket(2)
getsockopt

Purpose
Gets socket options

Synopsis
#include <sys/types.h>
#include <sys/socket.h>

int getsockopt (  
    int socket,  
    int level,  
    int option_nam,  
    char *option_value,  
    int *option_len );

Parameters

socket
Specifies the unique socket name.

level
Specifies the protocol level at which the option resides. To retrieve options at the socket level, specify the level parameter as SOL_SOCKET. To retrieve options at other levels, supply the appropriate protocol number for the protocol controlling the option. For example, to indicate that an option will be interpreted by the TCP protocol, set level to the protocol number of TCP, as defined in the netinet/in.h header file, or as determined by using the getprotobynam() function.

option_nam
Specifies a single option to be retrieved. The socket level options can be enabled or disabled by the setsockopt() function. The getsockopt() function retrieves information about the following options:

SO_DEBUG
Reports whether debugging information is being recorded. This option returns an int value.

SO_ACCEPTCONN
Reports whether socket listening is enabled. This option returns an int value.

SO_BROADCAST
Reports whether transmission of broadcast messages is supported. This option returns an int value.
SO_REUSEADDR
Reports whether the rules used in validating addresses supplied by a `bind()` function should allow reuse of local addresses. This option returns an `int` value.

SO_KEEPALIVE
Reports whether connections are kept active with periodic transmission of messages. If the connected socket fails to respond to these messages, the connection is broken and processes using that socket are notified with a SIGPIPE signal. This option returns an `int` value.

SO_DONTROUTE
Reports whether outgoing messages should bypass the standard routing facilities. (Not recommended, for debugging purposes only.) This option returns an `int` value.

SO_USELOOPBACK
Only valid for routing sockets. Reports whether the sender receives a copy of each message. This option returns an `int` value.

SO_LINGER
Reports whether the socket lingers on a `close()` function if data is present. If SO_LINGER is set, the system blocks the process during the `close()` function until it can transmit the data or until the time expires. If SO_LINGER is not specified, and a `close()` function is issued, the system handles the call in a way that allows the process to continue as quickly as possible. This option returns an `struct linger` value.

SO_OOBINLINE
Reports whether the socket leaves received out-of-band data (data marked urgent) in line. This option returns an `int` value.

SO_SNDBUF
Reports send buffer size information. This option returns an `int` value.

SO_RCVBUF
Reports receive buffer size information. This option returns an `int` value.

SO_SNDOLOWAT
Reports send low-water mark information. This option returns an `int` value.
Functions

getsockopt(2)

SO_RCVLOWAT
Reports receive low-water mark information. This option returns an int value.

SO_SNDTIMEO
Reports send time-out information. This option returns a struct timeval value.

SO_RCVTIMEO
Reports receive time-out information. This option returns a struct timeval value.

SO_ERROR
Reports information about error status and clear. This option returns an int value.

SO_TYPE
Reports the socket type. This option returns an int value.

Options at other protocol levels vary in format and name.

option_value
Points to the address of a buffer.

option_len
Specifies the length of buffer pointed to by option_value. The option_len parameter initially contains the size of the buffer pointed to by the option_value parameter. On return, the option_len parameter is modified to indicate the actual size of the value returned. If no option value is supplied or returned, the option_value parameter can be 0 (zero).

Options at other protocol levels vary in format and name.

Description

The getsockopt() function allows an application program to query socket options. The calling program specifies the name of the socket, the name of the option, and a place to store the requested information. The operating system gets the socket option information from its internal data structures and passes the requested information back to the calling program.

Options may exist at multiple protocol levels. They are always present at the uppermost socket level. When retrieving socket options, specify the level at which the option resides and the name of the option.
Return Values

Upon successful completion, the `getsockopt()` function returns a value of 0 (zero). Otherwise, a value of -1 is returned, and `errno` is set to indicate the error.

Errors

If the `getsockopt()` function fails, `errno` may be set to one of the following values:

[EBADF] The `socket` parameter is not valid.
[ENOTSOCK] The `socket` parameter refers to a file, not a socket.
[ENOPROTOOPT] The option is unknown.
[EFAULT] The address pointed to by the `option_value` parameter is not in a valid (writable) part of the process space, or the `option_len` parameter is not in a valid part of the process address space.

Related Information

Functions: `bind(2)`, `close(2)`, `endprotoent(3)`, `getprotobynumber(3)`, `getprotoent(3)`, `setprotoent(3)`, `setsockopt(2)`, `socket(2)`
gettimeofday, settimeofday, ftime

Purpose

Gets and sets date and time

Library

Standard C Library (libc.a)
ftime() call: Berkeley Compatibility Library (libbsd.a)

Synopsis

#include <sys/time.h>
int gettimeofday(
    struct timeval *tp,
    struct timezone *tzp);

int settimeofday(
    struct timeval *tp,
    struct timezone *tzp);

#include <sys/time.h>
#include <sys/timeb.h>
int ftime(
    struct timeb *tp);

Parameters

tp Points to a timeval structure, defined in the sys/time.h file.
tzp Points to a timezone structure, defined in the sys/time.h file.

Description

The gettimeofday() and settimeofday() functions get and set the system's notion of the current time and time zone. The time is expressed in seconds and microseconds since midnight (0 hour), January 1, 1970. The resolution of the system clock is hardware dependent, and the time may be updated continuously or in ticks. If the tzp parameter is 0 (zero), the time zone information will not be returned or set.

The tp parameter returns a pointer to a timeval structure which contains the time since the epoch began in seconds (up to 1000 milliseconds of a more precise interval), the local time zone (measured in minutes westward from Coordinated Universal Time), and a flag that, if nonzero, indicates that daylight saving time applies.
The `timezone` structure indicates the local time zone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that daylight saving time applies locally during the appropriate part of the year.

In addition to the difference in timer granularity, the `timezone` structure distinguishes these calls from the OSF Application Environment Specification `getclock` and `setclock` calls, which deal strictly with Coordinated Universal Time.

**Notes**

A process must have superuser privilege to set the system's time.

The `gettimeofday()` and `settimeofday()` functions are supported for compatibility with BSD programs. They support a process-local time zone parameter in addition to the system-wide time and date.

The `ftime()` function is included for compatibility with older BSD programs. Its function has been made obsolete by the `gettimeofday()` function.

**Return Values**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `gettimeofday()` or `settimeofday()` function fails, `errno` may be set the following value:

- `[EFAULT]` A parameter points to an invalid address.
- `[EPERM]` The process’s effective user ID does not have superuser privilege.

**Related Information**

Functions: `adjtime(2)`, `ctime(3)`, `gettimer(3)`, `strftime(3)`

Commands: `date(1)`
gettimer

**Purpose**

Gets date and time

**Library**

Standard C Library (libc.a)

**Synopsis**

```c
#include <sys/time.h>

int gettimer(
    timer_t timerid,
    struct itimerspec *tp);
```

**Parameters**

- `timerid` Specifies the timer to get the current time from; only time TIMEOFDAY is supported.
- `tp` Points to a `itimerspec` structure.

**Description**

The `gettimer()` function gets the current value of a system time-of-day clock. The `timerid` parameter specifies the symbolic name that identifies the timer whose time is being monitored. Only one symbolic name may be specified: TIMEOFDAY, which returns the CUT (Coordinated Universal Time) time and date. The `tp` parameter points to a type `itimerspec` structure, which has members that specify the elapsed time and date in nanoseconds from 00:00:00 UCT, January 1, 1970. The `tp` structure is defined in the `sys/time.h` include file.

Actual resolution of a timer is determined by the basic system hardware clock period, which is 1/HZ. The `restimer` function returns the resolution for any particular system.

**Notes**

**AES Support Level:** Trial use

**Return Values**

Upon successful completion, the `gettimer()` function returns the value 0 (zero). Otherwise, a value of -1 is returned and `errno` is set to indicate the error.
Errors

If the `gettimer()` function fails, `errno` may be set to the following value:

`[EINVAL]` The `timerid` parameter does not specify a known timer.

Related Information

Functions: `gettimeofday(2)`, `gettimeofday(2)`, `gettimeofday(2)`, `getitimer(2)`
getuid, geteuid

Purpose
Gets the process’ real or effective user ID

Synopsis
```
#include <unistd.h>
#include <sys/types.h>
uid_t getuid( void);
uid_t geteuid( void);
```

Description
The `getuid()` function returns the real user ID of the current process.
The `geteuid()` function returns the effective user ID of the current process.

Notes
AES Support Level: Full use

Related Information
Functions: `setuid(2), setruid(3), setreuid(2)`
getusershell, setusershell, endusershell

**Purpose**

Gets names of legal user shells

**Library**

Standard C Library (libc.a)

**Synopsis**

```c
char *getusershell( void );
int setusershell( void );
int endusershell( void );
```

**Description**

The `getusershell()` function returns a pointer to a string that contains the name of a legal user shell as defined by the system manager in the `/etc/shells` file. If the `/etc/shells` file does not exist, the standard system shells are returned.

The `getusershell()` function reads the next line of the `/etc/shells` file, opening it if necessary. The `setusershell()` function rewinds the file, and the `endusershell()` function closes it.

**Notes**

The returned information is in a static area. It must be copied if it is to be saved.

**Return Values**

Upon successful completion, a pointer to a character string is returned. A null pointer is returned on EOF (End-of-File) or error.

**Files**

`/etc/shells` Contains the names of legal user shells.
getutent, getutid, getutline, pututline, setutent, endutent, utmpname

Purpose
Accesses utmp file entries

Library
Standard C Library (libc.a)

Synopsis
#include <utmp.h>
struct utmp *getutent ( void );
struct utmp *getutid (  
    struct utmp *ID );
struct utmp *getutline (  
    struct utmp *line );
void pututline (  
    struct utmp *utmp_ptr );
void setutent ( void );
void endutent ( void );
void utmpname (  
    char *file );

Parameters

ID Specifies one of RUN_LVL, BOOT_TIME, OLD_TIME, NEW_TIME, INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS or DEAD_PROCESS.

If ID is one of RUN_LVL, BOOT_TIME, OLD_TIME, or NEW_TIME, the getutid() function searches forward from the current point in the utmp file until an entry with a ut_type matching ID->ut_type is found.

If ID is one of INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS or DEAD_PROCESS, the getutid() function returns a pointer to the first entry whose type is one of these four and whose ut_id field matches ID->ut_id. If the end of the file is reached without a match, the getutid() function fails.
Description

The `getutent()`, `getutid()`, and `getutline()` functions return a pointer to a `utmp` structure.

The `getutent()` function reads the next entry from a `utmp`-like file. If the file is not already open, the `getutent()` function opens it. If the end of the file is reached, the `getutent()` function fails.

The `pututline()` function writes the supplied `utmp_ptr` parameter structure into the `utmp` file. If you have not searched for the proper place in the file using one of the `getut-` routines, the `pututline()` function calls `getutid()` to search forward for the proper place. It is expected that the user of `pututline()` searched for the proper entry using one of the `getut-` functions. If so, `pututline()` does not search. If the `pututline()` function does not find a matching slot for the entry, it adds a new entry to the end of the file.

The `setutent()` function resets the input stream to the beginning of the file. You should do this before each search for a new entry if you want to examine the entire file.

The `endutent()` function closes the currently open file.

The `utmpname()` function changes the name of the file to be examined from `/var/adm/utmp` to any other filename. The name specified is usually `/var/adm/wtmp`. If the specified file does not exist, no indication is given until the file is referenced. The `utmpname()` function does not open the file, but closes the old file (if it is currently open) and saves the new filename.

The most current entry is saved in a static structure, making the `utmpname()` function non-reentrant. To make multiple accesses, you must copy or use the structure between each access. The `getutid()` and `getutline()` functions examine the static structure first. If the contents of the static structure match what they are searching for, they do not read the `utmp` file. Therefore, you must fill the static structure with zeros after each use if you want to use these subroutines to search for multiple occurrences.
If the `pututline()` function finds that it is not already at the correct place in the file, the implicit read it performs does not overwrite the contents of the static structure returned by the `getutent()`, `getuid()`, or `getutline()` functions. This allows you to get an entry with one of these subroutines, modify the structure, and pass the pointer back to the `pututline()` function for writing.

These functions use buffered standard I/O for input, but the `pututline()` function uses an unbuffered nonstandard write to avoid race conditions between processes trying to modify the `utmp` and `wtmp` files.

**Return Values**

These functions fail and return a null pointer if a read or write fails due to the end of the file, or due to a permission conflict.

**Files**

/`etc/utmp`
/`usr/adm/wtmp`

**Related Information**

Functions: `ttyslot(3)`
getwc, fgetwc, getwchar

Purpose
Gets a character or word from an input stream

Library
Standard I/O Package (libc.a)

Synopsis
#include <stdio.h>
int getwc ( FILE *stream );
int fgetwc ( FILE *stream );
int getwchar ( void );

Parameters
stream Specifies the input data.

Description
The getwc( ), fgetwc( ), and getwchar( ) functions are provided when Japanese Language Support is installed on your system.

The getwc( ) function gets the next 1-byte or 2-byte character from the input stream specified by the stream parameter, and returns an NLchar data type as an integer. The fgetwc( ) function performs the same function as getwc( ).

The getwchar( ) function gets the next 1-byte or 2-byte character from the standard input stream and returns an NLchar as an integer.

Return Values
These functions and macros return the integer constant EOF at the end of the file or upon an error.

Related Information
Functions: fopen(3), fread(3), getc(3), gets(3), putwc(3), scanf(3), wscanf(3)
**getwd**

**Purpose**
Gets current directory pathname

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
char *getwd(
    char *path_name);
```

**Parameters**
- `path_name`
  Points to the full pathname.

**Description**
The `getwd()` function determines the absolute pathname of the current directory, then copies that pathname into the area pointed to by the `path_name` parameter.

The maximum pathname length, in characters, is set by the PATH_MAX definition, as specified in the `limits.h` file.

**Return Values**
Upon successful completion, a pointer to the absolute pathname of the current directory is returned. If an error occurs, the `getwd()` function returns a value of 0 (zero) and places a message in the `path_name` parameter.

**Related Information**
Functions: `getcwd(3)`
getws, fgetws

**Purpose**
 Gets a string from a stream

**Library**
 Standard I/O Library (libc.a)

**Synopsis**

```
#include <NLchar.h>
NLchar *getws (NLchar *string );

NLchar *fgetws (NLchar *string, int number, FILE *stream );
```

**Parameters**

- **string** Points to a string to receive characters.
- **stream** Points to the FILE structure of an open file.
- **number** Specifies an upper bound on the number of characters to read.

**Description**

The getws() and fgetws() functions are provided when Japanese Language Support is installed on your system. They parallel the gets() and fgets() functions.

The getws() function transforms multibyte character input values to uniform NLchar width. The fgetws() function also expands 1-byte and 2-byte character input values to uniform NLchar (2-byte) width.

**Return Values**

If the end of the file is encountered and no characters have been read, no characters are transferred to the string parameter and a null pointer is returned. If a read error occurs, a null pointer is returned. Otherwise, string is returned.

**Related Information**

hsearch, hcreate, hdestroy

Purpose
Manages hash tables

Library
Standard C Library (libc.a)

Synopsis
#include <search.h>
ENTRY *hsearch(
    ENTRY item,
    ACTION action);
int hcreate(
    unsigned int nel);
void hdestroy(void);

Parameters
item Identifies a structure of the type ENTRY as defined in the search.h header file. It contains two pointers:
    item.key Points to the comparison key string.
    item.data Points to any other data associated with the item.key parameter.

Pointers to types other than char should be cast as char *.

action Specifies a value for an ACTION enum type, which indicates what is to be done with an item key when it cannot be found in the hash table. The ACTION enum type specifies the following two actions that can be specified for this parameter:

    ENTER Enter the key specified by the item parameter into the hash table at the appropriate place. When the table is full, a null pointer is returned.

    FIND Do not enter the item key into the table, but return a null pointer when an item key cannot be found in the hash table.
hsearch(3)

nel Specifies an estimate of the maximum number of entries that the hash table contains. Under some circumstances, the hcreate() function may make the hash table larger than specified, to obtain mathematically favorable conditions for access to the hash table.

Description

The hsearch(), hcreate() and hdestroy() functions are used to manage hash-table operations.

The hsearch() function searches a hash table. It returns a pointer into a hash table that indicates where a given entry can be found. The hsearch() function uses "open addressing" with a hash function.

The hcreate() function initializes the hash table. You must call the hcreate() function before calling the hsearch() function.

The hdestroy() function deletes the hash table. This allows you to start a new hash table because only one table may be active at a time. After the call to hdestroy() the hash-table data should no longer be considered accessible.

Notes

AES Support Level: Trial use

Return Values

The hsearch() function returns a null pointer when the action is FIND and the key pointed to by item can not be found, or when the specified action is ENTER and the hash table is full.

Upon successful completion, the hcreate() function returns a nonzero value. Otherwise, when sufficient space for the table cannot be allocated, the hcreate() function returns 0 (zero).

Related Information

Functions: bsearch(3), lsearch(3), tsearch(3)
htonl

**Purpose**
Converts an unsigned long (32-bit) integer from host-byte order to Internet network-byte order

**Library**
Standard C Library (libc.a)

**Synopsis**

```c
#include <netinet/in.h>

unsigned long htonl (unsigned long hostlong);
```

**Parameters**

- `hostlong` Specifies a 32-bit integer in host-byte order.

**Description**

The `htonl()` function converts an unsigned long (32-bit) integer from host-byte order to Internet network-byte order.

The Internet network requires address and port reference data in network-byte order (most significant byte leftmost, least significant byte rightmost). Use the `htonl()` function to convert address and port long integers from Internet host-byte order to Internet network-byte ordered long integers.

The `htonl()` function is defined as a *big-endian* macro in the `netinet/in.h` header file for machine environments where network-byte order and host-byte order are identical.

**Return Values**

Upon successful completion, the `htonl()` function returns a 32-bit long integer in Internet network-byte order.

**Related Information**

- Functions: `htons(3)`, `ntohl(3)`, `ntrhs(3)`
htons

Purpose

Converts an unsigned short (16-bit) integer from host-byte order to a 2-byte Internet network integer

Library

Standard C Library (libc.a)

Synopsis

#include <netinet/in.h>

unsigned short htons ( unsigned short hostshort ) ;

Parameters

hostshort

Specifies a 16-bit integer in host-byte order.

Description

The htonl() function converts an unsigned short (16-bit) integer from host-byte order to Internet network-byte order.

The Internet network requires address and port reference data in network-byte order (most significant byte leftmost, least significant byte rightmost). Use the htonl() function to convert address and port short integers from host-byte order to Internet network-byte order.

The htonl() function is defined as a big-endian macro in the netinet/in.h header file for machine environments where network-byte order and host-byte order are identical.

Return Values

Upon successful completion, the htons() function returns a 16-bit short integer in Internet network-byte order.

Related Information

Functions: htonl(3), ntohl(3), ntohs(3)
hypot, cabs

**Purpose** Computes Euclidean distance function and complex absolute value

**Library** Math Library (libm.a)

**Synopsis**
```c
#include <math.h>
double hypot ( double x, double y);
double cabs ( struct {double x, y;} z);
```

**Parameters**
- **x** Specifies a double value.
- **y** Specifies a double value.
- **z** Specifies a structure that has two double elements.

**Description**
The `hypot()` and `cabs()` functions compute the length of the hypotenuse of a right angled triangle with the formula:

\[
\sqrt{x^2 + y^2}.
\]

**Notes**

AES Support Level: Trial use (hypot())
Errors

If the hypot() function fails, errno may be set to one of the following values:

[EDOM] The value of x or y is NaN.
[ERANGE] The value to be returned would cause overflow.

Related Information

Functions: exp(3), isnan(3), sqrt(3)
**inet_addr**

**Purpose**
Translates an Internet network address string to an Internet address integer

**Library**
Standard C Library (libc.a)

**Synopsis**

```c
#include <netinet/in.h>
#include <arpa/inet.h>

unsigned long inet_addr (char *string);
```

**Parameters**

- `string`: Defines an Internet dot-formatted address character string of the form `a.b.c.d`, where `a`, `b`, `c`, and `d` may be expressed as decimal, octal, or hexadecimal integers in the C idiom.

**Description**

The `inet_addr()` function translates a dot-formatted Internet character address string to an Internet address integer. The Internet address integer is returned as a network byte-ordered integer (most significant byte leftmost, least significant byte rightmost).

**Notes**

On VAX machines, the dot-formatted network-address `a.b.c.d` is returned as the machine integer `dcba`.

**Return Values**

Upon successful completion, the `inet_addr()` function returns an equivalent network byte-ordered address integer. Otherwise, -1 is returned.

**Related Information**

Functions: `inet_netof(3)`, `inet_inaof(3)`, `inet_makeaddr(3)`, `inet_network(3)`, `inet_ntoa(3)`
inet_lnaof

**Purpose**
Translates an Internet address integer into its host (local) address component

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <netinet/in.h>
#include <arpa/inet.h>

u_long inet_lnaof(
    struct in_addr net_addr);
```

**Parameters**
- `net_addr`
  Defines an Internet address as a network byte-ordered integer. May be expressed as octal (leading 0), hexadecimal (leading 0x or 0X), or decimal.

**Description**
The `inet_lnaof()` function translates an Internet network byte-ordered address into its host (local) address component. The host address integer is returned in network byte-order (most significant byte leftmost, least significant byte rightmost).

**Return Values**
Upon successful completion, the `inet_lnaof()` function returns a network byte-ordered integer that specifies the host (local) address part of the Internet network address integer. When the `inet_lnaof()` function fails, -1 is returned.

**Related Information**
Functions: `inet_addr(3)`, `inet_netof(3)`, `inet_makeaddr(3)`, `inet_network(3)`, `inet_ntoa(3)`
inet_makeaddr

Purpose  Translates an Internet address and host address into an Internet byte-ordered address integer

Library  Standard C Library (libc.a)

Synopsis  

```c
#include <netinet/in.h>
#include <arpa/inet.h>

struct in_addr inet_makeaddr (u_long net_num, u_long loc_addr);
```

Parameters

- **net_num**  Defines an Internet number in network-byte order. May be expressed as octal (leading 0), hexadecimal (leading 0x or 0X), or decimal.

- **loc_addr**  Defines a host (local) address integer. May be expressed as octal (leading 0), hexadecimal (leading 0x or 0X), or decimal.

Description

The `inet_makeaddr()` function translates a multipart Internet address and a local host address into their equivalent Internet byte-ordered address integer. The Internet network address integer is returned in network-byte order (most significant byte leftmost, least significant byte rightmost).

Return Values

Upon successful completion, the `inet_makeaddr()` function returns a machine integer that specifies the Internet network byte-ordered address. When the `inet_makeaddr()` function fails, -1 is returned.

Related Information

Functions:  `inet_addr(3)`, `inet_lnaof(3)`, `inet_netof(3)`, `inet_network(3)`, `inet_ntoa(3)`
inet_netof

Purpose

Translates an Internet address integer into its network address component

Library

Standard C Library (libc.a)

Synopsis

```
#include <netinet/in.h>
#include <arpa/inet.h>

u_long inet_netof(
    struct in_addr net_addr);
```

Parameters

net_addr

Defines an Internet address in network-byte order. May be expressed as octal (leading 0), hexadecimal (leading 0x or 0X), or decimal.

Description

The `inet_netof()` function translates an Internet address into its network address component. The network address integer is returned in network-byte order (most significant byte leftmost, least significant byte rightmost).

Return Values

Upon successful completion, the `inet_netof()` function returns a network byte-ordered integer that specifies the Internet network address. When the `inet_netof()` function fails, -1 is returned.
**inet_network**

**Purpose**  Translates an Internet dot-formatted address string to a network address integer

**Library**  Standard C Library (libc.a)

**Synopsis**

```c
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

u_long inet_network (char *string);
```

**Parameters**

- `string`  Defines an Internet dot-formatted address as the character string `a.b.c.d`, where `a`, `b`, `c` and `d` may be expressed as decimal, octal, or hexadecimal in the C-language idiom.

**Description**

The `inet_network()` function translates a dot-formatted Internet network character address string to a network byte-ordered address integer (most significant byte leftmost, least significant byte rightmost).

**Return Values**

Upon successful completion, the `inet_network()` function returns an Internet byte-ordered address integer. When the `inet_network()` function fails, -1 is returned.

**Related Information**

Functions:  `inet_netof(3)`, `inet_inaof(3)`, `inet_makeaddr(3)`, `inet_addr(3)`, `inet_ntoa(3)`
inet_ntoa

**Purpose**  Translates an Internet integer address into a dot-formatted character string

**Library**  Standard C Library (libc.a)

**Synopsis**
```
#include <netinet/in.h>
#include <arpa/inet.h>
char *inet_ntoa (struct in_addr net_addr);
```

**Parameters**

`net_addr`  Defines an Internet network-byte ordered integer address to be converted to an equivalent character string.

**Description**

The `inet_ntoa()` function translates an Internet byte-ordered address integer into a dot-formatted character string. The dot-formatted string is returned in network-byte order (most significant byte leftmost, least significant byte rightmost).

**Return Values**

Upon successful completion, the `inet_ntoa()` function returns a dot-formatted Internet character string address. When the `inet_ntoa()` function fails, -1 is returned.

**Related Information**

Functions: `inet_netof(3)`, `inet_lnaof(3)`, `inet_makeaddr(3)`, `inet_addr(3)`, `inet_network(3)`
initgroups

Purpose
Initializes concurrent group set

Library
Standard C Library (libc.a)

Synopsis
```c
int initgroups (char *user, gid_t base_gid);
```

Parameters
- `user` Specifies the user whose groups are to be used to initialize the group set.
- `base_gid` Specifies an additional group to include in the group set.

Description
The `initgroups()` function reads the defined group membership of the specified `user` and sets the concurrent group set of the current process to that value. The `base_gid` parameter is always included in the concurrent group set, and is normally the principal user's group. If the user is in more than NGROUPS_MAX groups, only NGROUPS_MAX groups are set, including the `base_gid` group.

Caution
The `initgroups()` function uses the `getgrent()` functions. If the program that invokes `initgroups()` uses any of these functions, then calling `initgroups()` overwrites the static group structure.

Return Values
Upon successful completion, the `initgroups()` function returns 0 (zero). If the `initgroups()` function fails, 1 is returned and `errno` is set to indicate the error.
initgroups(3)

Errors

If the `initgroups()` function fails, `errno` may be set to the following value:

- [EPERM] The calling process does not have the appropriate privilege in its current effective privilege set.

Related Information

- Functions: `getgroups(2)`, `setgroups(2)`, `getgid(2)`
- Commands: `groups(1)`
insque, remque

Purpose  Inserts or removes an element in a queue

Library  Standard C Library (libc.a)

Synopsis  

struct qelem [  
    struct qelem *q_forw;  
    struct qelem *q_back;  
    char q_data[ ];  
];  
insque (  
    struct qelem *element,  
    struct qelem *pred );  
remque (  
    struct qelem *element );

Parameters

pred       Points to the element in the queue immediately before the element to be inserted or deleted.

element    Points to the element in the queue immediately after the element to be inserted or deleted.

Description

The insque() and remque() functions manipulate queues built from double-linked lists. Each element in the queue must be in the form of a qelem structure. The q_forw and q_back elements of that structure must point to the elements in the queue immediately before and after the element to be inserted or deleted.

The insque() function inserts the element pointed to by the element parameter into a queue immediately after the element pointed to by the pred parameter.

The remque() function removes the element defined by the element parameter from a queue.
ioctl

**Purpose**
Controls devices

**Synopsis**
```c
#include <sys/ioctl.h>

ioctl(   
    int d,
    unsigned long request,
    char *argp);
```

**Parameters**
- `d` Specifies the file descriptor of the requested device.
- `request` Specifies the ioctl command to be performed on the device.
- `argp` Points to an parameter array for the request.

**Description**
The ioctl() function performs a variety of operations on open descriptors. In particular, many operating characteristics of character special files (for example, terminals) may be controlled with ioctl() requests.

An ioctl() request has encoded in it whether the parameter is an "in" parameter or "out" parameter, and the size of the argp parameter in bytes. Macros and defines used in specifying an ioctl() request are located in the sys/ioctl.h file.

**Return Values**
If an error occurs, a value of -1 is returned and errno is set to indicate the error.

**Errors**
If the ioctl() function fails, errno may be set to one of the following values:
- [EBADF] The `d` parameter is not a valid descriptor.
- [ENOTTY] The `d` parameter is not associated with a character special device.
[ENOTTY] The specified request does not apply to the kind of object that the descriptor \( d \) references.

[EINVAL] Either the request or argp parameter is not valid.

Related Information

Functions: exec(2), fcntl(2)
Files: tty(7) lvm(7)
isnan

Purpose  Tests for NaN (Not a Number)

Library  Math Library (libm.a)

Synopsis  

```c
#include <math.h>

int isnan ( double x );
```

Parameters  

`x`  Specifies a double value.

Description  

The `isnan()` function tests whether `x` is NaN (Not a Number).

Notes  

AES Support Level:  Trial use

Return Values  

The `isnan()` function returns a nonzero value if `x` is NaN. Otherwise, 0 (zero) is returned.
isjalpha, isjdigit, isjxdigit, isjalnum, isjspace, isjpunct

**Purpose**
Classifies characters

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <ctype.h>

int isjalpha (int c);
int isjdigit (int c);
int isjxdigit (int c);
int isjalnum (int c);
int isjspace (int c);
int isjpunct (int c);
```

**Parameters**
- `c`: Specifies the character to be tested.

**Description**
The Japanese `ctype` functions are provided when Japanese Language Support is installed on your system. The `ctype` macros classify character-coded integer values specified in a table. Each of these macros returns a nonzero value for TRUE and 0 (zero) for FALSE.

The following list shows the `ctype` macros which should be used when classifying characters of type `NLchar`:

- **isjalnum**: The `c` parameter specifies a letter or digit.
  - 0x8260 - 0x8279
  - 0x8281 - 0x829A

- **isjalpha**: The `c` parameter specifies an alphabetic SJIS character.
  - 0x8140

- **isjspace**: The `c` parameter specifies a space SJIS character.
  - 0x8140
isjpunct The $c$ parameter specifies a punctuation SJIS character, that is, neither a control character nor an alphanumeric character.
0x8141-0x8151 0x815A-0x8198 0x81F5-0x81

isjdigit The $c$ parameter specifies a digit SJIS character in the range [0-9].
0x824F-0x8258

isjxdigit The $c$ parameter specifies an Arabic hexadecimal SJIS character in the range [0-9], [A-F], or [a-f].
0x824F-0x8258
0x8260-0x8265
0x8281-0x8286

Related Information

Functions: ctype(3), setlocale(3)
kill

Purpose Sends a signal to a process or to a group of processes

Library Berkeley Compatibility Library (libbsd.a)

Synopsis

```c
#include <sys/types.h>
#include <signal.h>
int kill(
    pid_t process;
    int signal);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>process</code></td>
<td>Specifies the process or group of processes.</td>
</tr>
<tr>
<td><code>signal</code></td>
<td>Specifies the signal. If the signal parameter is a value of 0 (the null signal), error checking is performed but no signal is sent. This can be used to check the validity of the process parameter.</td>
</tr>
<tr>
<td><code>process_grp</code></td>
<td>Specifies the process group.</td>
</tr>
</tbody>
</table>

Description

The `kill()` function sends the signal specified by the `signal` parameter to the process or group of processes specified by the `process` parameter.

To send a signal to another process, at least one of the following must be true:

- The real or the effective user ID of the sending process matches the real or effective user ID of the receiving process.
- The process is trying to send the SIGCONT signal to one of its session's processes.
- The calling process has superuser privilege.

Processes can send signals to themselves.

Notes

Sending a signal does not imply that the operation is successful. All signal operations must pass the access checks prescribed by each enforced access control policy on the system.
If the `process` parameter is greater than 0 (zero), the signal specified by the `signal` parameter is sent to the process that has a process ID equal to the value of the `process` parameter.

If the `process` parameter is equal to 0 (zero), the signal specified by the `signal` parameter is sent to all of the processes (other than system processes) whose process group ID is equal to the process group ID of the sender.

If the `process` parameter is equal to -1 and the effective user ID of the sender has root privileges, the signal specified by the `signal` parameter is sent to all of the processes other than system processes.

If the `process` parameter is negative but not -1, the signal specified by the `signal` parameter is sent to all of the processes which have a process group ID equal to the absolute value of the `process` parameter.

The `killpg()` function is provided by OSF/1 for binary compatibility only.

**AES Support Level:** Full use

**Return Values**

Upon successful completion, the `kill()` function returns a value of 0 (zero). Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `kill()` function fails, no signal is sent and `errno` may be set to one of the following values:

- `[EINVAL]` The `signal` parameter is not a valid signal number.
- `[EINVAL]` The `signal` parameter is SIGKILL, SIGSTOP, SIGTSTP or SIGCONT and the `process` parameter is 1 (proc1).
- `[ESRCH]` No process can be found corresponding to that specified by the `process` parameter.
- `[EPERM]` The real or saved user ID does not match the real or effective user ID of the receiving process, the calling process does not have appropriate privilege, and the process is not sending a SIGCONT signal to one of its session's processes.
- `[EACCES]` The calling process does not have appropriate privilege.

**Related Information**

Functions: `getpid(2)`, `kill(2)`, `setpgid(2)`, `sigaction(2)`, `sigvec(2)`
ldr.entry

Purpose
Returns the entry point for a loaded module

Library
Standard C Library (libc.a)

Synopsis
#include <sys/types.h>
#include <loader.h>

ldr_entry_pt_t ldr_entry(
    ldr_module_t mod_id);

Parameters
mod_id
Identifies the loaded module. The module ID is returned when the module
is first loaded.

Description
The ldr_entry() function returns the entry point for the specified loaded module.

Return Values
Upon successful completion, the ldr_entry() function returns the entry point. If
the operation fails, the function returns null and errno is set to indicate the error.

Errors
If the ldr_entry() function fails, errno may be set to the following value:
[EINVAL] The specified module ID has no entry point or is not valid.

Related Information
Functions: load(3), ldr_xentry(3)
ldr_inq_module

**Purpose**
Returns information about a loaded module

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <sys/types.h>
#include <loader.h>
int ldr_inq_module(
    ldr_process_t process,
    ldr_module_t mod_id,
    ldr_module_info_t *info,
    size_t info_size,
    size_t *ret_size);
```

**Parameters**
- **process**
  Specifies the process whose address space contains the module for which information is required.
- **mod_id**
  Identifies the module. The module ID is returned when the module is first loaded.
- **info**
  Points to a buffer into which the information is returned.
- **info_size**
  Specifies the size of the info buffer, in bytes.
- **ret_size**
  Specifies the number of bytes returned into the info buffer.

**Description**
The `ldr_inq_module()` function returns information about a specified module contained within the address space of the specified process into the variable pointed to by the `info` parameter. The `info_size` parameter is the size of the buffer provided. The number of bytes filled in (that is, the returned structure size) is returned in the buffer pointed to by the `ret_size` parameter.
To obtain the unique process identifier for the current process, use the call:

```c
ldr_process_t ldr_my_process();
```

To obtain the unique process identifier for the kernel, use the call:

```c
ldr_process_t ldr_kernel_process();
```

**Notes**

This function is currently only implemented for the current process and the kernel.

**Return Values**

Upon successful completion, the function returns a value of 0 (zero). If the operation fails, the function returns a negative error value and sets `errno` to indicate the error.

**Errors**

If the `ldr_inq_module()` function fails, `errno` may be set to one of the following values:

- [EINVAL] The specified module ID is not valid.
- [ESRCH] The process identifier is not valid.

In addition, errors pertaining to the IPC mechanism can be returned.

**Related Information**

Functions: `ldr_inq_region(3)`, `ldr_next_module(3)`
ldr_inq_region

Purpose

Returns module information about a region in a loaded module

Library

Standard C Library (libc.a)

Synopsis

```c
#include <sys/types.h>
#include <loader.h>
int ldr_inq_region(
    ldr_process_t process,
    ldr_module_t mod_id,
    ldr_region_t region,
    ldr_region_info_t *info,
    size_t info_size,
    size_t *ret_size);
```

Parameters

- **process**
  Specifies the process whose address space contains the module for which the region information is required.

- **mod_id**
  Identifies the module. The module ID is returned when the module is first loaded.

- **region**
  Identifies the region.

- **info**
  Points to a ldr_region_info_t buffer in which the information about the loaded region is returned.

- **info_size**
  Specifies the size of the allocated ldr_region_info_t structure, in bytes.

- **ret_size**
  Specifies the number of types actually returned into the buffer pointed to by the info parameter.

Description

The ldr_inq_region() function returns information about a specified region within a specified module. The module is contained within the address space of the
specified process. The returned information includes the number and name of the region, its protection attributes, its size, and address information about the region in the process’ address space.

To obtain the unique identifier for the current process, use the call:

```c
ldr_process_t ldr_my_process();
```

To obtain the unique identifier for the kernel, use the call:

```c
ldr_process_t ldr_kernel_process();
```

The `ldr_region_t` values are unique identifiers for each loaded region for a particular module. The value of the `region` parameter ranges from 0 (zero) to (maximum number of regions) -1.

**Notes**

The loader assumes that each object module contains one or more regions. A region is a separately relocated, virtually contiguous range within a module. A region can contain text or data.

This function is currently implemented only for the current process and the kernel.

**Return Values**

Upon successful completion, the function returns a value of 0 (zero). If the operation fails, the function returns a negative error value and `errno` is set to indicate the error.

**Errors**

If the `ldr_inq_region()` function fails, `errno` may be set to one of the following values:

- `[EINVAL]` The specified module ID or region ID is not valid.
- `[ESRCH]` The process identifier is not valid.

Additional errors may be returned from the underlying IPC mechanism (for kernel/cross-process loading).

**Related Information**

Functions: `ldr_inq_module(3)`, `ldr_next_module(3)`
ldr_install

Purpose
Installs a module in the current process' private known package table

Library
Standard C Library (libc.a)

Synopsis
#include <loader.h>
int ldr_install(
    const char * mod_name);

Parameters
mod_name
Points to the name of the module to be installed.

Description
The ldr_install() function installs a specified module in the current process' private known package table. The private known package table is inherited copy-on-write by the process' children. This makes the packages exported by the module available for symbol resolution for modules loaded into this process and its children, overriding any module exporting the package in the global known package table.

Return Values
Upon successful completion, the ldr_install() function returns 0 (zero). Otherwise, a negative value is returned and errno is set to indicate the error.

Errors
If the ldr_install() function fails, errno may be set to the following value:

[EEXIST] The module was previously loaded in the known package table of this process.
[ENOENT] The named file does not exist.
[EACCES] Search permission is denied, or the file exists and the user does not have read access.
Functions

ldr_install(3)

[ENOEXEC] The file is not in a recognized format.
[ENOSPC] The process’ private known package table is full and cannot be expanded.

Related Information
Functions: load(3), ldr_remove(3)
ldr_lookup_package

Purpose
Returns the address of a symbol name in a package.

Library
Standard C Library (libc.a)

Synopsis
#include <loader.h>
void *ldr_lookup_package(
    char *package,
    char *symbol_name);

Parameters
package
    Specifies the name of the package that contains the symbol name.
symbol_name
    Specifies the name of the symbol whose address is required.

Description
The ldr_lookup_package() function returns the address of the specified symbol name within the specified package.

Notes
The loader employs a two-dimensional hierarchical symbol name space in which each symbol is represented by a package name, symbol name pair. Packages are an abstraction that allows symbol resolution at the granularity of a library or fraction of a library without having to bind symbols to library names.

Package names are attached to symbols at link time. The set of symbols exported by a library is divided among one or more packages. By default, a package name is derived from a library name, so that all symbols exported by a given library belong to the same package. However, the programmer can attach symbols to arbitrary package names to create multiple packages within a library.

When a module is linked against a library, the linker can derive the package name for each imported symbol from the package name associated with the corresponding exported symbol. The programmer, however, can also assign arbitrary package names to imported symbols at link time.
The package scheme avoids symbol name conflicts when more than one library
exports the same symbol. It assumes that each symbol name is unique within its
package and that each package name is unique across the system. Since each
imported symbol includes a package name, the symbol name can be resolved
unambiguously to the correct exported symbol.

Return Values

Upon successful completion, the address of the specified symbol is returned.
Otherwise, null is returned and errno is set to indicate the error.

Errors

If the \texttt{ldr\_lookup\_package()} function fails, errno may be set to one of the
following values:

\begin{itemize}
  \item [\texttt{ENOSYM}] The specified package does not contain the specified symbol name.
  \item [\texttt{ERANGE}] The symbol value cannot be represented as an absolute value.
  \item [\texttt{ENOPKG}] The specified package name is not known in this process.
\end{itemize}

Related Information

Functions: load(3), ldr\_xlookup\_package(3)
ldr_next_module

**Purpose**
Returns the next module ID for a process

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <sys/types.h>
#include <loader.h>
int ldr_next_module(
    ldr_process_t process,
    ldr_module_t *mod_id_ptr);
```

**Parameters**
- **process**
  Specifies the process for which the next module ID is required.
- **mod_id_ptr**
  Points to a buffer in which the module ID of a loaded module will be returned.

**Description**
The `ldr_next_module()` function returns the next module ID for the specified process, given a specified module ID. It iterates through the module IDs of all modules currently loaded in a specified process.

To get the first module ID for the process, specify `LDR_NULL_MODULE` for the initial module ID. Repeated calls to the `ldr_next_module()` function will return all the module IDs for the process. The function returns `LDR_NULL_MODULE` after returning the last module ID.

To obtain the unique identifier for the current process, use the following call:
```
ldr_process_t ldr_my_process();
```

To obtain the unique identifier for the kernel, use the following call:
```
ldr_process_t ldr_kernel_process();
```

To return the IDs for kernel modules, specify the returned identifier for the `process` parameter.
Return Values

Upon successful completion, the function returns a value of 0 (zero). If the operation fails, the function returns a negative value and `errno` is set to indicate the error.

Errors

If the `ldr_next_module()` function fails, `errno` may be set to the following value:

[EINVAL] The module ID specified by `mod_id_ptr` is not valid.

Related Information

Functions: `ldr_inq_module(3)`, `ldr_inq_region(3)`


**ldr_remove**

**Purpose**
Removes an installed module from the private known package table

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <loader.h>

int ldr_remove(
    const char *mod_name);
```

**Parameters**

`mod_name`
Points to the name of the module to be removed from known package table.

**Description**
The `ldr_remove()` function removes a specified module from the current process’ private known package table. This module must have been previously installed in the private known package table with the `ldr_install()` function, or have been inherited as loaded in the private known package table from its parent process.

**Return Values**
Upon successful completion, the `ldr_remove()` function returns 0 (zero). Otherwise, a negative value is returned and `errno` is set to indicate the error.

**Errors**
If the `ldr_remove()` function fails, `errno` may be set to the following value:

[EINVAL] The module was not found in the process’ private known package table.

**Related Information**

Functions: `ldr_install(3), load(3)`
**ldr_xattach**

**Purpose**
Attaches to another process to permit loading/unloading of modules in that process’ address space.

**Library**
Standard C Library (*libc.a*)

**Synopsis**
```
#include <sys/types.h>
#include <loader.h>

int ldr_xattach(
    ldr_process_t process);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td>Specifies the process to attach to.</td>
</tr>
</tbody>
</table>

**Description**

The `ldr_xattach()` function is used to permit a process to load, unload, query, or retrieve the contents of another process’ address space. Before a call to the `ldr_xload()`, `ldr_xunload()`, `ldr_xlookup()`, or `ldr_xlookup_package()` functions, the `ldr_xattach` function must be performed to that process.

**Notes**

This function currently works only for the current process or the kernel.

**Return Values**

If the attach operation is a success, the function returns a code of 0 (zero). If the attach fails, the function returns a negative error value and `errno` is set to indicate the error.
Errors

If the `ldr_xattach()` function fails, `errno` may be set to the following value:

[ESRCH] The process identifier is invalid.

Additional errors are possible from the underlying IPC mechanism.

Related Information

Functions: `ldr_xdetach(3)`, `ldr_xunload(3)`, `ldr_xlookup(3)`, `ldr_xlookup_package(3)`, `ldr_xload(3)`
**Functions**

**ldr_xdetach(3)**

---

**ldr_xdetach**

**Purpose**  
Detaches from an attached process

**Library**  
Standard C Library (*libc.a*)

**Synopsis**  
```c
#include <sys/types.h>  
#include <loader.h>

int ldr_xdetach(
    ldr_process_t process);
```

**Parameters**  
- **process**: Specifies the process from which to detach.

**Description**  
The `ldr_xdetach()` function detaches the calling process from `process`, with which it had been associated for cross-process loading and debugging. This procedure should be used only if a `ldr_xattach()` was previously performed on the specified `process`.

**Notes**  
This function currently works only for the current process and the kernel.

**Return Values**  
If the detach operation is a success, the function returns a value of 0 (zero). If the detach fails, the function returns a negative value and `errno` is set to indicate the error.
Errors

If the ldr_xdetach() function fails, errno may be set to the following value:

[ESRCH] The process identifier is invalid.

Additional errors are possible from the underlying IPC mechanism.

Related Information

Functions: ldr_xattach(3)
Functions

ldr_xentry(3)

ldr_xentry

Purpose
Returns the entry point for a module loaded in another process

Library
Standard C Library (libc.a)

Synopsis
#include <sys/types.h>
#include <loader.h>

int ldr_xentry(  
    ldr_process_t process,  
    ldr_module_t mod_id,  
    ldr_entry_pt_t *entry_pt);  

Parameters

process
Specifies the process whose address space contains the module for which
the entry point is required.

mod_id
Identifies the loaded module. The module ID is returned when the module
is first loaded.

entry_pt
Points to a buffer in which the entry point will be returned.

Description

The ldr_xentry( ) function returns the entry point for the specified module in the
address space of the specified process.

To obtain the unique identifier for the current process, use the following call:

    ldr_process_t ldr_my_process( );

To obtain the unique identifier for the kernel, use the following call:

    ldr_process_t ldr_kernel_process( );

Notes

This function currently works only for the current process and the kernel process.
Return Values

Upon successful completion, the function returns a value of 0 (zero). If the operation fails, the function returns a negative value and errno is set to indicate the error.

Errors

If the ldr_xentry() function fails, errno may be set to one of the following values:

- [EINVAL] The specified module ID is not valid.
- [EINVAL] There is no entry point for the loaded module.
- [ESRCH] The process identifier is invalid.

Additional errors may occur due to the underlying IPC mechanism.

Related Information

Functions: ldr_entry(3), ldr_xload(3), load(3)
**ldr_xload**

**Purpose**
Loads a module in another process and returns the module ID

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <sys/types.h>
#include <loader.h>
int ldr_xload(
    ldr_process_t process,
    char *file_pathname,
    ldr_load_flags_t load_flags,
    ldr_module_t *mod_id_ptr);
```

**Parameters**
- **process**
  Specifies the process into whose address space the object module is to be loaded.
- **file_pathname**
  Specifies the pathname of the object module.
- **load_flags**
  Specifies options on the load. Valid values are:
  - **LDR_WIRE**
    Wire the module in physical memory so that it will not be paged out.
  - **LDR_NOFLAGS**
    No flags are specified.
  - **LDR_NOUNREFS**
    Allow no unresolved references after resolving shared library references.
ldr_xload(3)

LDR_PREEXIST
The module must have been already loaded.

LDR_NOPREEXIST
Return an error if the module is already loaded.

mod_id_ptr
Points to a variable in which the module ID of the loaded module is returned.

Description

The ldr_xload() function loads the specified object module into the virtual address space of the specified process. It can be used to load both relocatable and absolute modules.

If the object module is already loaded, the function does not load it again, but it does return its ID. Using the LDR_NOPREXST load flag forces an error if the module is already loaded.

To obtain the unique identifier for the current process, use the following call:

ldr_process_t ldr_my_process();

To obtain the unique identifier for the kernel, use the following call:

ldr_process_t ldr_kernel_process();

Notes

The loader assigns a unique identifier to each module when it is loaded. Module IDs provide a convenient way of referencing loaded modules in other loader-related functions.

The loader can link unresolved references in dynamically loaded kernel modules, relocate the code as necessary, and call an initialization entry point. The loader, however, cannot automatically load further modules to resolve unresolved references. Each kernel module must link completely against symbols exported by the kernel or by previously loaded modules. Circular dependencies are not allowed for dynamically loaded kernel modules.

This function currently works only for the current process and for the kernel.

Return Values

Upon successful completion, the module is loaded and 0 (zero) is returned. Otherwise, a negative value is returned and errno is set to indicate the error.
Errors

If the ldr_xload() function fails, errno may be set to one of the following values:

[ENOEXEC] The file_pathname parameter specifies a file with an unrecognizable object file format.

[EINVAL] The load_flags parameter specified an invalid option or an invalid ldr_module_t has been specified.

[EEXIST] The LDR_NOPREXST load flag was specified and the module was already loaded.

[ESRCH] The process identifier is invalid.

[ENOPKG] One or more unresolved package names were found.

[ENOSYM] One or more unresolved symbol names were found.

[EDUPPKG] The loaded module exported a package which duplicated the package name of a module already loaded in the same process.

Related Information

Functions: ldr_xunload(3), ldr_xentry(3), ldr_xlookup(3), load(3)
Returns the address of a symbol name within a specified package in another process

Standard C Library (libc.a)

#include <sys/types.h>
#include <loader.h>
int ldr_xlookup_package(
    ldr_process_t process,
    char *package_name,
    char *symbol_name,
    void **symbol_addr_ptr);

Specifies the process whose address space contains the package with the symbol whose address is required.

Specifies the name of the package that contains the symbol name.

Specifies the name of the symbol whose address is required.

Points to a void* variable. The function returns the address for the symbol name in this variable.

The ldr_xlookup_package( ) function returns the address of the specified symbol name within the specified package. The package is contained within the address space of the specified process.

To obtain the unique identifier for the current process, use the following call:

    ldr_process_t ldr_my_process( );

To obtain the unique identifier for the kernel, use the following call:

    ldr_process_t ldr_kernel_process( );
Notes

This call currently only supports lookup in the current process or the kernel.

The loader employs a two-dimensional hierarchical symbol name space in which each symbol is represented by a package name, symbol name pair. Package names are attached to symbols at link time. The package scheme assumes that each symbol name is unique within its package and that each package name is unique across the system.

Return Values

If the operation is a success, the function returns a value of 0 (zero). If the operation fails, the function returns a negative value and errno is set to indicate the error.

Errors

If the ldr_xlookup_package() function fails, errno may be set to one of the following values:

[ENOPKG] The specified package was not found.
[ENOSYM] The specified symbol name was not found in the specified package.
[ESRCH] The process identifier is invalid.
[ERANGE] The symbol address could not be converted into an absolute value.

Additional errors may be returned from the underlying IPC mechanism.

Related Information

Functions: ldr_lookup_package(3), ldr_xload(3), load(3)
ldr_xunload

**Purpose**
Unloads a module previously loaded in another process

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <sys/types.h>
#include <loader.h>
int ldr_xunload(
    ldr_process_t process,
    ldr_module_t mod_id);
```

**Parameters**

- **process**
  Specifies the process from whose address space the module is to be unloaded.

- **mod_id**
  Identifies the module to be unloaded. The module ID is returned when the module is first loaded.

**Description**
The ldr_xunload() function unloads the specified module from the virtual address space of the specified process. The function unmaps the module's regions and discards the loader data structures that describe the module.

The module is unloaded even if any references to it remain in other modules. The loader does not keep track of such dangling references or attempt to unsnap any invalidated links. These housekeeping tasks are the responsibility of the calling process. Attempts to refer to addresses in an unloaded module can result in indeterminate errors.

To obtain the unique identifier for the current process, use the following call:

```c
ldr_process_t ldr_my_process( );
```

To obtain the unique identifier for the kernel, use the following call:

```c
ldr_process_t ldr_kernel_process( );
```
Notes

This function currently works only for the current process and the kernel process. Once a module has been unloaded, its module ID is no longer valid.

Return Values

If the unload operation is a success, the function returns a value of 0 (zero). If the unload fails, the function returns a negative value and errno is set to indicate the error.

Errors

If the ldr_xunload() function fails, errno may be set to one of the following values:

[EINVAL] The specified module ID is not valid.
[EINVAL] The specified module cannot be unloaded (that is, it was loaded with the flag LDR_NOUNLOAD).
[ESRCH] The process identifier is invalid.

Related Information

Functions: ldr_xload(3), load(3), unload(3)
Programmers Workbench Library

Purpose

Provides functions for compatibility with existing programs

Library

Programmers Workbench Library (libPW.a)

Description

The libpw functions are provided for compatibility with existing programs. Their use in new programs is not recommended.

any (character, string)
   Determines whether string contains character.

anystr (string1, string2)
   Determines the offset in string1 of the first character that also occurs in string2.

balbrk (string, open, close, end)
   Determines the offset in string of the first character in the string end that occurs outside of a balanced string as defined by open and close.

cat (destination, source1, ..., 0)
   Concatenates the source strings and copies them to destination.

clean_up ()
   Defaults the cleanup routine.

curdir (string)
   Puts the full pathname of the current directory in string.

dname (p)
   Determines which directory contains the file p.

fatal (message)
   General purpose error handler.

fdfopen (fd, mode)
   Same as the stdio fopen() function.

giveup (dump)
   Forces a core dump.

imatch (pref, string)
   Determines if the string pref is an initial substring of string.
index (string1, string2)
Determines the offset of the first occurrence in string1 of string2.

lockit (lockfile, count, pid)
Creates a lock file.

logname ()
Returns caller’s login name.

move (string1, string2, n)
Copies the first n characters of string1 to string2.

patoi (string)
Converts string to integer.

patol (string)
Converts string to long.

repeat (destination, string, n)
Sets destination to string repeated n times.

repl (string, old, new)
Replaces each occurrence of the character old in string with the character new.

satoi (string, ip)
Converts string to integer and saves it in *ip.

setsig ()
Causes signals to be caught by the setsig1() function.

setsig1 (signal)
General purpose signal handling routine.

sname (s)
Gets a pointer to the simple name of full pathname s.

strend (string)
Finds the end of string.

substr (s, destination, origin, length)
Places a substring of string s in destination using the offset origin and length.

translat (s, old, new, destination)
Copies string s into destination and replaces any character in old with the corresponding characters in new.

unlockit (lockfile, pid)
Deletes the lock file.

userdir (uid)
Gets the user’s login directory.

userexit (code)
Defaults user exit routine.
username (uid)
   Gets the user's login name.

verify (string1, string2)
   Determines the offset in string1 of the first character that is not also in string2.

xalloc (asize)
   Allocates memory.

xcreat (name, mode)
   Creates a file.

xfree (aptr)
   Frees memory.

xfreeall ()
   Frees all memory.

xlink (fl, f2)
   Links files.

xmsg (file, func)
   Calls the fatal() function with an appropriate error message.

xopen (name, mode)
   Opens a file.

xpipe (t)
   Creates a pipe.

xunlink (f)
   Removes a directory entry.

xwrite (fd, buffer, n)
   Writes n bytes to the file associated with fd from buffer.

zero (p, n)
   Zeros n bytes starting at address p.

zeropad (s)
   Replaces the initial blanks with the character '0' in string s.
**link**

**Purpose**
Creates an additional directory entry for an existing file on current file system

**Synopsis**

```c
int link(
    const char *path1,
    const char *path2);
```

**Parameters**

- `path1` Points to the pathname of an existing file.
- `path2` Points to the pathname for the directory entry to be created. If the `path2` parameter names a symbolic link, an error is returned.

**Description**

The `link()` function creates an additional hard link (directory entry) for an existing file. Both the old and the new link share equal access rights to the underlying object. The `link()` function atomically creates a new link for the existing file and increments the link count of the file by one.

Both the `path1` and `path2` parameters must reside on the same file system. A process must have superuser privilege to make a directory hard link.

Upon successful completion, the `link()` function marks the `st_ctime` field of the file for update, and marks the `st_ctime` and `st_mtime` fields of the directory containing the new entry for update.

**Notes**

**AES Support Level:** Full use

**Return Values**

Upon successful completion, the `link()` function returns a value of 0 (zero). If the `link()` function fails, a value of -1 is returned, no link is created, and `errno` is set to indicate the error.
Errors

If the `link()` function fails, `errno` may be set to one of the following values:

- **[ENOENT]** The file named by the `path1` parameter does not exist or the `path1` or `path2` parameter is an empty string.
- **[EFAULT]** Either the `path1` or `path2` parameter is an invalid address.
- **[EEXIST]** The link named by the `path2` parameter already exists.
- **[EPERM]** The file named by the `path1` parameter is a directory and the calling process does not have appropriate privilege.
- **[EXDEV]** The link named by the `path2` parameter and the file named by the `path1` parameter are on different file systems.
- **[EACCES]** The requested link requires writing in a directory with a mode that denies write permission, or a component of either the `path1` or `path2` parameter denies search permission.
- **[EMLINK]** The number of links to the file named by `path1` would exceed LINK_MAX.
- **[EROFS]** The requested link requires writing in a directory on a read-only file system.
- **[ENOSPC]** The directory in which the entry for the new link is being placed cannot be extended because there is no space left on the file system containing the directory.
- **[ELOOP]** Too many links were encountered in translating either `path1` or `path2`.
- **[ENAMETOOLONG]** The length of the `path1` or `path2` string exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- **[ENOTDIR]** A component of either path prefix is not a directory.
- **[EDQUOT]** The directory in which the entry for the new link is being placed cannot be extended because the user’s quota of disk blocks on the file system containing the directory has been exhausted.

Related Information

Functions: `unlink(2)`

Commands: `link(1)`
listen

Purpose  
Listens for socket connections and limits the backlog of incoming connections

Synopsis  
```c
int listen (  
    int socket,  
    int backlog );
```

Parameters

- `socket`  
  Specifies the unique name for the socket.
- `backlog`  
  Specifies the maximum number of outstanding connection requests.

Description

The `listen()` function identifies the socket that receives the connections, marks the socket as accepting connections, and limits the number (`backlog`) of outstanding connection requests in the system queue.

The maximum queue length (`backlog`) that the `listen()` function can specify is five. The maximum queue length is indicated by the SOMAXCONN value in the `sys/socket.h` header file.

Return Values

Upon successful completion, the `listen()` function returns a value of 0 (zero). Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `listen()` function fails, `errno` may be set to one of the following values:

- `[EBADF]`  
  The `socket` parameter is not valid.
- `[ENOTSOCK]`  
  The `socket` parameter refers to a file, not a socket.
- `[EOPNOTSUPP]`  
  The referenced socket is not a type that supports the `listen()` function.
Related Information

Functions: accept(2), connect(2), socket(2)
load

Purpose Loads a module and returns the module ID

Library Standard C Library (libc.a)

Synopsis #include <sys/types.h>
#include <loader.h>
ldr_module_t load(
    char *file_pathname,
    ldr_load_flags_t load_flags);

Parameters

file_pathname
   Specifies the pathname of the object module to be loaded.

load_flags
   Specifies options on the load. Valid values are:
   LDR_NOFLAGS
      No flags are specified.
   LDR_NOINIT
      Do not run initialization routines.
   LDR_NOUNREFS
      Allow no unresolved references after resolving shared
      library references.
   LDR_PREXIST
      The module must have been already loaded.
   LDR_NOPREXIST
      Return an error if the module is already loaded.

Description

The load() function loads the specified object module into the virtual address
space of the calling process.

If the object module is already loaded, the function does not load it again, but it
returns its module ID unless the LDR_NOPREXIST load flag is specified. To use
the LDR_PREXIST load flag, the module must already be loaded and its module
ID returned.
The loader assigns a unique identifier to each module when it is loaded. This identifier is called a module ID, and is defined as type \texttt{ldr\_module\_t}. Module IDs provide a convenient way to reference loaded modules in other loader-related functions. For example, the \texttt{ldr\_entry()} function returns the entry point for the loaded module associated with a specified module ID.

Upon successful completion, a nonzero module ID of type \texttt{ldr\_module\_t} is returned. Otherwise, a module ID of 0 (zero) is returned and \texttt{errno} is set to indicate the error.

If the \texttt{load()} function fails, \texttt{errno} may be set to one of the following values:

[ENOEXEC] The \texttt{file\_pathname} parameter specifies a file with a bad object file format.

[EINVAL] The \texttt{load\_flags} parameter specified an invalid option.

[EEXIST] The LDR_NOPREXST load flag was specified and the module was already loaded.

[ENOSYM] One or more unresolved external symbols were found.

[ENOPKG] One or more unresolved package names were found.

Related Information

Functions: \texttt{unload(3), ldr\_entry(3), ldr\_lookup(3), ldr\_xload(3)}
localeconv, localeconv_r

Purpose
Retrieves locale-dependent formatting parameters

Library
Standard C Library (libc.a)

Synopsis
#include <locale.h>
struct lconv *localeconv ( void )

int localeconv_r( 
    struct lconv *result,
    char *buf,
    int len );

Parameters

result Points to a lconv structure in which to return the conventions.

buf Points to a buffer used for constructing char *'s.

len Specifies the length of buf.

Description
The localeconv() function provides access to the object that specifies the current locale’s conventions for the format of numeric quantities.

The lconv structure contains values appropriate for formatting numeric quantities (monetary and otherwise) according to the rules of the current locale. The members of the structure with the type char * are strings, any of which (except decimal_point) can point to a null string, to indicate that the value is not available in the current locale or is of zero length. The members with type char are nonnegative numbers, any of which can be CHAR_MAX to indicate that the value is not available in the current locale. The members include the following:

char *decimal_point
    The decimal-point character used to format nonmonetary quantities.

cchar *thousands_sep
    The separator for groups of digits to the left of the decimal point in formatted nonmonetary quantities.
char *grouping
   A string whose elements indicate the size of each group of digits in formatted nonmonetary quantities.

char *int_curr_symbol
   The international currency symbol applicable to the current locale, left justified within a four-character, space-padded field. The character sequences are in accordance with those specified in ISO 4217 Codes for the Representation of Currency and Funds.

char *currency_symbol
   The currency symbol applicable to the current locale.

char *mon_decimal_point
   The decimal point used to format monetary quantities.

char *mon_thousands_sep
   The separator for groups of digits to the left of the decimal point in formatted monetary quantities.

char *mon_grouping
   A string whose elements indicate the size of each group of digits in formatted monetary quantities.

char *positive_sign
   The string used to indicate a nonnegative formatted monetary quantity.

char *negative_sign
   The string used to indicate a negative formatted monetary quantity.

char int_frac_digits
   The number of fractional digits (those to the right of the decimal point) to be displayed in a formatted monetary quantity.

char frac_digits
   The number of fractional digits (those to the right of the decimal points) to be displayed in a formatted monetary quantity.

char p_cs_precedes
   Set to 1 or 0 (zero) if the currency_symbol respectively precedes or succeeds the value for a nonnegative formatted monetary quantity.

char p_sep_by_space
   Set to 1 or 0 (zero) if the currency_symbol respectively is or is not separated by a space from the value for a nonnegative formatted monetary quantity.
Functions
localeconv(3)

char n_cs_precedes
Set to 1 or 0 (zero) if the currency_symbol respectively precedes or succeeds the value for a negative formatted monetary quantity.

char n_sep_by_space
Set to 1 or 0 (zero) if the currency_symbol respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

char p_sign_posn
Set to a value indicating the positioning of the positive_sign for nonnegative formatted monetary quantity.

char n_sign_posn
Set to a value indicating the positioning of the negative_sign for a negative formatted monetary quantity.

The elements of grouping and mon_grouping are interpreted according to the following:

MAX_CHAR
No further grouping is to be performed.

0 The previous element is to be repeatedly used for the remainder of the digits.

other The value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits to the left of the current group.

The value of p_sign_posn and n_sign_posn is interpreted according to the following:

0 Parenthesis surround the quantity and currency_symbol.
1 The sign string precedes the quantity and currency_symbol.
2 The sign string succeeds the quantity and currency_symbol.
3 The sign string immediately precedes the currency_symbol.
4 The sign string immediately succeeds the currency_symbol.

The localeconv_r() function is the reentrant version of localeconv(). The conventions are filled into the structure pointed to by the result parameter. The buf parameter is used to construct all the members of the structure with type char *.
Notes

The `localeconv()` function is not supported for multi-threaded applications. Instead, its reentrant version, `localeconv_r()`, should be used with multiple threads.

Library functions do not call the `localeconv()` function.

AES Support Level: Full use (`localeconv()`)

Return Values

Upon successful completion, the `localeconv()` function returns a pointer to the filled-in object. The structure pointed to by the return value will not be modified by the program, but may be overwritten by a subsequent call to the `localeconv()` function. In addition, calls to the `setlocale()` function with categories LC_ALL, LC_MONETARY or LC_NUMERIC may overwrite the contents of the structure.

Upon successful completion, the `localeconv_r()` function returns a value of 0 (zero). Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors

If the `localeconv_r()` function fails, `errno` may be set to the following value:

[EINVAL] Either the result or buffer parameters are null pointers.

[ENOMEM] The buffer parameter is too small.

Related Information

Functions: `setlocale(3)`
lockf

**Purpose**
Controls open file descriptors

**Synopsis**

```c
#include <fcntl.h>

int lockf(
    int filedes,
    int request,
    off_t size);
```

**Parameters**

- `filedes` Specifies the file to which the lock is to be applied or removed. The file descriptor is returned by a successful `open()` or `fcntl()` function.
- `request` Specifies one of the following constants for the `lockf()` function:
  - `F_ULOCK` Unlocks a previously locked region in the file.
  - `F_LOCK` Locks the region for exclusive use. This request causes the calling process to sleep if the region overlaps a locked region, and to resume when it is granted the lock.
  - `F_TLOCK` Same as `F_LOCK`, except that the request returns an error if the region overlaps a locked region.
  - `F_TEST` Tests to see if another process has already locked a region. The `lockf()` function returns 0 (zero) if the region is unlocked. If the region is locked, then -1 is returned and `errno` is set to [EACCES].
- `size` The number of bytes to be locked or unlocked for the `lockf()` function. The region starts at the current location in the open file and extends forward if `size` is positive and backward if `size` is negative. If the `size` parameter is 0 (zero), the region starts at the current location and extends forward to the maximum possible file size, including the unallocated space after the end of the file.

**Description**

The `lockf()` function locks and unlocks sections of an open file. Unlike the `fcntl()` function, however, its interface is limited to setting only write (exclusive) locks.
Although the `lockf()` and `fcntl()` functions are different, the implementations are fully integrated. Therefore, locks obtained from one function are honored and enforced by the other lock function.

Each lock is either an enforced lock or an advisory lock, and must also be either a read lock or a write lock.

Locks on a file are advisory or enforced depending on the mode of the file (see the `chmod()` function). A given file can have advisory or enforced locks, but not both. See the `sys/mode.h` header file for a description of file attributes.

When a process holds an enforced exclusive lock on a section of a file, no other process can access that section of the file with the `read()` or `write()` functions. In addition, the `open()`, `truncate()`, and `ftruncate()` functions cannot truncate the locked section of the file. If another process attempts to read or modify the locked section of the file, it sleeps until the section is unlocked or returns with an error indication.

The file descriptor on which an exclusive lock is being placed must have been opened with write access.

Some general rules about file locking include the following:

- Changing or unlocking part of a file in the middle of a locked section leaves two smaller sections locked at each end of the originally locked section.
- All locks associated with a file for a given process are removed when the process closes any file descriptor for that file.
- Locks are not inherited by a child process after running a `fork()` function.

Locks can start and extend beyond the current end of a file, but cannot be negative relative to the beginning of the file. A lock can be set to extend to the end of the file by setting the `l_len` field to 0 (zero). If a lock is specified with the `l_start` field set to 0 and the `l_whence` field set to SEEK_SET, the whole file is locked.

**Notes**

Buffered I/O does not work properly when used with file locking. Do not use the standard I/O package routines on files that will be locked.

Deadlocks due to file locks in a distributed system are not always detected. When such deadlocks are possible, the programs requesting the locks should set time-out timers.
Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the lockf() function fails, errno may be set to one of the following values:

- [EACCESS] The file region is locked and F_TEST was specified.
- [EINVAL] The request parameter is not valid.
- [EBADF] The filedes parameter is not a valid open file descriptor; or the request parameter is F_SETLK or F_SETLKW, the type of lock (l_type) is a shared lock (F_RDLCK) and filedes is not a valid file descriptor open for reading; or the type of lock (l_type) is an exclusive lock (F_WRLCK) and filedes is not a valid file descriptor open for writing.
- [EINTR] The command parameter is F_SETLKW and the fcntl() function was interrupted by a signal which was caught.
- [EDEADLK] The lock is blocked by some lock from another process. Putting the calling process to sleep while waiting for that lock to become free would cause a deadlock.

Related Information

Functions: chmod(2), close(2), exec(2), fcntl(2), flock(2), fork(2), open(2), read(2), write(2)
lsearch, lfind

Purpose
Performs a linear search and update

Library
Standard C Library (libc.a)

Synopsis
#include <search.h>
#include <sys/types.h>

void *lsearch(
    const void *key,
    const void *base,
    size_t *nelp,
    size_t width,
    int (*compar) (const void *, const void *)) ;

void *lfind(
    const void *key,
    const void *base,
    size_t *nelp,
    size_t width,
    int (*compar) (const void *, const void *)) ;

Parameters

key
Points to an entry containing the key that specifies the entry to be searched for in the table.

base
Points to the first entry in the table to be searched.

nelp
Points to an integer that specifies the number of entries in the table to be searched. This integer is incremented whenever an entry is added to the table.

width
Specifies the size of each entry, in bytes.

compar
Points to the user-specified function to be used for comparing two table entries (strcmp(), for example). This function must return 0 (zero) when called with arguments that point to entries whose keys compare equal, and nonzero otherwise.
Description

The `lsearch()` function performs a linear search of a table. This function returns a pointer into a table indicating where a specified key is located in the table. When the key is not found in the table, it is added to the end of the table. Free space must be available at the end of the table, or other program information may be corrupted.

The `lfind()` function is similar to the `lsearch()` function, except that when a key is not found in a table, an entry for it is not added to the table. In this case, a null pointer is returned.

Notes

Pointers to the `key` parameter and the entry at the base of the table should be of type `pointer-to-element` and cast to type `pointer-to-character`. Although it is declared as type `pointer-to-character`, the returned value should be cast into type `pointer-to-element`.

The comparison function need not compare every byte; therefore, the table entries can contain arbitrary data in addition to the values undergoing comparison.

AES Support Level: Trial use

Return Values

Upon successful completion, both the `lsearch()` and `lfind()` functions return a pointer to its location in the table. Otherwise, the `lfind()` function returns a null pointer and the `lsearch()` function returns a pointer to the location of the newly added table entry.

Related Information

Functions: `bsearch(3), hsearch(3), tsearch(3), qsort(3)`
lseek

Purpose
Moves read-write file offset

Synopsis
#include <sys/types.h>
#include <unistd.h>
off_t lseek (  
    int filedes,  
    off_t offset,  
    int whence );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filedes</td>
<td>Specifies a file descriptor obtained from a successful open() or fcntl() function.</td>
</tr>
<tr>
<td>offset</td>
<td>Specifies a value, in bytes, that is used in conjunction with the whence parameter to set the file pointer. A negative value causes seeking in the reverse direction. The resulting file position may also be negative.</td>
</tr>
<tr>
<td>whence</td>
<td>Specifies how to interpret the offset parameter in setting the file pointer associated with the filedes parameter. Values for the whence parameter are as follows: SEEK_SET</td>
</tr>
</tbody>
</table>

Description
The lseek() function sets the file offset for the open file specified by the filedes parameter. The whence parameter determines how the offset is to be interpreted.
The `lseek()` function allows the file offset to be set beyond the end of existing data in the file. If data is later written at this point, subsequently reading data in the gap returns bytes with the value 0 (zero) until data is actually written into the gap.

The `lseek()` function does not, by itself, extend the size of the file.

Notes

AES Support Level: Full use

Return Values

Upon successful completion, the resulting pointer location, measured in bytes from the beginning of the file, is returned. If the `lseek()` function fails, the file offset remains unchanged, a value of `(off_t) - 1` is returned, and `errno` is set to indicate the error.

Errors

If the `lseek()` function fails, the file offset remains unchanged and `errno` may be set to one of the following values:

- **[EBADF]** The `filedes` parameter is not an open file descriptor.
- **[ESPIPE]** The `filedes` parameter is associated with a pipe (FIFO), a socket, or a multiplexed special file.
- **[EINVAL]** The `whence` parameter is an invalid value, or the resulting file offset would be invalid.

Related Information

Functions: `fcntl(2)`, `fseek(3)`, `open(2)`, `read(2)`, `write(2)`
madvise

Purpose
Advise the system of a process’ expected paging behavior

Synopsis
```
#include <sys/types.h>
#include <sys/mman.h>

int madvise (                          
        caddr_t addr,                  
        size_t len,                   
        int behav );                   
```

Parameters
- `addr` Specifies the address of the region to which the advice refers.
- `len` Specifies the length in bytes of the region specified by the `addr` parameter.
- `behav` Specifies the behavior of the region. The following values for the `behav` parameter are defined in the `sys/mman.h` header file:
  - MADV_NORMAL
    No further special treatment
  - MADV_RANDOM
    Expect random page references
  - MADV_SEQUENTIAL
    Expect sequential references
  - MADV_WILLNEED
    Will need these pages
  - MADV_DONTNEED
    Do not need these pages
  - MADV_SPACEAVAIL
    Ensure that resources are reserved

Description
The `madvise()` function permits a process to advise the system about its expected future behavior in referencing a mapped file or shared memory region.
Notes

The `madvise()` function has no functionality in OSF/1. It is supported for compatibility only.

AES Support Level: Trial use

Return Values

Upon successful completion, the `madvise()` function returns zero. Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors

If the `madvise()` function fails, `errno` may be set to one of the following values:

- [EINVAL] The `behav` parameter is invalid.
- [ENOSPC] The `behav` parameter specifies MADV_SPACEAVAIL and resources can not be reserved.

Related Information

Functions: `mmap(2)`
malloc, free, realloc, calloc, mallopt, mallinfo, alloca

Purpose
Provides a memory allocator

Library
Standard C Library (libc.a)
Berkeley Compatibility Library (libbsd.a)
Pthreads library (libpthreads.a)

Synopsis
#include <malloc.h>

void *malloc(
    size_t size);
char *alloca(
    int size);
void free(
    void *pointer);
void *realloc(
    void *pointer,
    size_t size);
int mallopt(
    int command,
    int value);
struct mallinfo mallinfo( void );
void *calloc(
    size_t num_of_elts,
    size_t elt_size);

Parameters
size Specifies a number of bytes of memory.
pointer Points to the block of memory that was returned by the malloc() or calloc() function.
command Specifies a mallopt() function command.
value Specifies M_MXFAST, M_NLBLKS, M_GRAIN, or M_KEEP.
Functions

malloc(3)

\[\text{num_of_elts}\] \quad \text{Specifies the number of elements in the array.}

\[\text{elt_size}\] \quad \text{Specifies the size of each element in the array.}

Description

The \text{malloc()} and \text{free()} functions provide a simple, general-purpose memory allocation package.

The \text{malloc()} function returns a pointer to a block of memory of at least the number of bytes specified by the \text{size} parameter. The block is aligned so that it can be used for any type of data.

The \text{free()} function frees the block of memory pointed to by the \text{pointer} parameter for further allocation. The block pointed to by the \text{pointer} parameter must have been previously allocated by either the \text{malloc()}, \text{realloc()}, or \text{calloc()} functions.

The \text{realloc()} function changes the size of the block of memory pointed to by the \text{pointer} parameter to the number of bytes specified by the \text{size} parameter, and returns a pointer to the block. The contents of the block remain unchanged up to the lesser of the old and new sizes. If necessary, a new block is allocated, and data is copied to it. If the \text{pointer} parameter is a null pointer, the \text{realloc()} function simply allocates a new block of the requested size. If the \text{size} parameter is 0 (zero), the \text{realloc()} function frees the specified block.

The \text{calloc()} function allocates space for an array with the number of elements specified by the \text{num_of_elts} parameter, where each element is of the size specified by the \text{elt_size} parameter. The space is initialized to zeros.

The \text{alloca()} function allocates the number of bytes of space specified by the \text{size} parameter in the stack frame of the caller. This space is automatically freed when the function that called the \text{alloca()} function returns to its caller.

The \text{mallopt()} and \text{mallinfo()} functions allow tuning the allocation algorithm at execution time.

The \text{mallopt()} function initiates a mechanism that can be used to allocate small blocks of memory quickly. You can use the \text{mallopt()} function to allocate a large group (called a holding block) of these small blocks at one time. Then, each time a program requests a small amount of memory, a pointer to one of the preallocated small blocks is returned. Different holding blocks are created for different sizes of
small blocks and are created when needed. This function allows the programmer to set the following three parameters to maximize efficient small block allocation for a particular application:

**size**
Below this value, requests to the `malloc()` function are filled using the special small block algorithm. Initially, this value, which is called `maxfast`, is zero, which means that the small block option is not normally in use by `malloc()`.

**number**
The number of small blocks in a holding block. If holding blocks have many more small blocks than the program is using, space will be wasted. If holding blocks are too small or have too few small blocks in each, performance gain is lost.

**grain**
The grain of small block sizes. This value determines what range of small block sizes is considered the same size, which influences the number of separate holding blocks allocated. For example, if the `grain` parameter is 16 bytes, all small blocks of 16 bytes or less belong to one holding block and blocks from 17 to 32 bytes belong to another holding block. Thus, if the `grain` parameter is too small, space may be wasted because many holding blocks are created.

The values for the *command* parameter to the `mallopt()` function are:

- **M_MXFAST** Sets `maxfast` to the `value` parameter. The algorithm allocates all blocks below the size of `maxfast` in large groups and then doles them out very quickly. The default value for `maxfast` is 0 (zero).

- **M_NLBLKS** Sets `numblks` to the `value` parameter. The aforementioned large groups each contain `numblks` blocks. The value for `numblks` must be greater than 1. The default value is 100.

- **M_GRAIN** Sets `grain` to the `value` parameter (must be greater than 0 (zero)). The sizes of all blocks smaller than `maxfast` are considered to be rounded up to the nearest multiple of `grain`. The default value for the `grain` parameter is the smallest number of bytes that allows alignment of any data type. When the `grain` parameter is set, the `value` parameter is rounded up to a multiple of the default

- **M_KEEP** Preserves data in a free block until the next call to the `malloc()`, `realloc()`, or `calloc()` function. This option is provided only for compatibility with the older version of the `malloc()` function and is not recommended.

The `mallopt()` function may be called repeatedly, but parameters cannot be changed after the first small block is allocated from a holding block. If the `mallopt()` function is called again after the first small block is allocated, it returns an error.
The `malloc()` function can be used during program development to determine the best settings of these parameters for a particular application. It must only be called after some storage has been allocated. Information describing space usage is returned. Refer to the `malloc.h` file for details of the `mallinfo` structure.

Notes

The `mallopt()` and `mallinfo()` functions are not supported for multi-threaded applications.

The `mallopt()` and `mallinfo()` functions are provided for System V compatibility only, and should not be used by new, portable applications. The behavior of the `malloc()` and `free()` functions may not be affected by calls to `mallopt()`. The structure returned by the `mallinfo()` function may not contain any useful information. The `mallopt()` and `mallinfo()` functions are designed for tuning a specific algorithm. OSF/1 uses a new, more efficient algorithm.

The `valloc()` function found in many BSD systems is supported as a compatibility interface in the Berkeley Compatibility Library (`libbsd.a`). The function of the `valloc()` function is superceded by the `malloc()` function, which automatically page aligns large (>= 1 page) requests. The `valloc()` syntax follows:

```c
char *valloc (size)
unsigned int size;
```

AES Support Level: Full use (`calloc()`, `free()`, `malloc()`, `realloc()`)

Return Values

Each of the allocation functions returns a pointer to space suitably aligned for storage of any type of object. Cast the pointer to the type pointer-to-element before using it.

The `malloc()`, `realloc()`, and `calloc()` functions return a null pointer if there is no available memory or if the memory arena has been corrupted by storing outside the bounds of a block. When this happens, the block pointed to by the `pointer` parameter could be destroyed.

Upon successful completion, the `mallopt()` function returns 0 (zero). Otherwise, a nonzero value is returned.

The `mallinfo()` function returns a pointer to a `mallinfo()` structure, defined in the `malloc.h` header file.
mblen

Purpose  Determines the length in bytes of a multibyte character

Library  Standard C Library (libc.a)

Synopsis  

```c
#include <stdlib.h>

int mblen(
    const char *mbs,
    size_t n);
```

Parameters

- `mbs` - Points to a multibyte character string.
- `n` - Specifies the maximum number of bytes to consider.

Description

The `mblen()` function determines the number of bytes in a multibyte character. The behavior of the `mblen()` function is affected by the LC_CTYPE category of the current locale. In environments with shift-state dependent encoding, calls to `mblen()` with a null value for the `mbs` parameter place the function in the initial shift state. Subsequent calls with the `mbs` parameter set to nonnull values alter the state of the function as necessary. Changing the LC_CTYPE category of the locale causes the shift state of the function to be indeterminate.

The implementation behaves as though no other function calls the `mblen()` function.

Notes

AES Support Level: Full use
Return Values

If the \textit{mbs} parameter does not have a null pointer value, the \texttt{mblen()} function returns a value determined as follows:

- If \textit{mbs} points to a valid multibyte character other than null, \texttt{mblen()} returns the number of bytes in the character unless the number of bytes is greater than \textit{n}.
- If \textit{mbs} points to the null character, \texttt{mblen()} returns 0 (zero).
- If \textit{mbs} does not point to a valid multibyte character or points to a character of more than \textit{n} bytes, \texttt{mblen()} returns -1 and sets \texttt{errno} to indicate the error.

When the \textit{mbs} parameter is a null pointer, the return value depends on the environment, as follows:

- In environments where encoding is not shift-state dependent, \texttt{mblen()} returns 0 (zero).
- In environments where encoding is shift-state dependent, \texttt{mblen()} returns a nonzero value.

Errors

If the \texttt{mblen()} function fails, \texttt{errno} may be set to the following value:

\[
\text{[EINVAL]} \quad \text{The mbs parameter points to an invalid multibyte character.}
\]

Related Information

Functions: \texttt{mbtowc(3), wctomb(3), mbstowcs(3), wcstombs(3)}
mbstowcs

Purpose
Converting a multibyte (single-byte or double-byte) character string to a wide character string

Library
Standard C Library (libc.a)

Synopsis
```
#include <stdlib.h>
size_t mbstowcs(
    wchar_t *pwcs,
    const char *s,
    size_t n);
```

Parameters
- `pwcs` Points to the array where the result of the conversion is stored.
- `s` Points to the multibyte character string to be converted.
- `n` Specifies the number of wide characters in the string to be converted.

Description
The `mbstowcs()` function converts a multibyte character string into a wide character string, which is stored at a specified location. The `mbstowcs()` function does not convert characters occurring after a null character in the input string (which is converted to value 0 (zero)). When operating on overlapping strings, the behavior of this function is undefined.

Behavior of the `mbstowcs()` function is affected by the LC_CTYPE category of the current locale. In environments that use shift-state dependent encoding, the array pointed to by the `s` parameter begins in the initial shift state.

Notes

AES Support Level: Full use
Return Values

When `mbstowcs()` encounters an invalid multibyte character during conversion, `(size_t) -1` is returned and `errno` is set to indicate the error. Otherwise, `mbstowcs()` returns the number of wide characters stored in the output array, not including a terminating null. (When the return value is `n`, the output array is not null-terminated.)

Errors

If the `mbstowcs()` function fails, `errno` may be set to the following value:

[EINVAL] The `s` parameter points to a string containing an invalid multibyte character.

Related Information

Functions: `mblen(3), mbtowc(3), wctomb(3), wcstombs(3)`
mbtowc

Purpose

Converts a multibyte character to a wide character

Library

Standard C Library (libc.a)

Synopsis

#include <stdlib.h>

int mbtowc(
    wchar_t *pwc,
    const char *s,
    size_t n);

Parameters

pwc  Points to the wide character variable location.

s    Points to multibyte character to be converted.

n    Specifies the number of bytes in the multibyte character.

Description

The mbtowc() function converts a multibyte character to a wide character and returns the number of bytes of the multibyte character, which is stored as an output variable. In environments with shift-state dependent encoding, calls to mbtowc() with the s parameter set to null, places the function in its initial shift state. Subsequent calls with the s parameter set to nonnull values alter the state of the function as necessary. Changing the LC_CTYPE category of the locale causes the shift state of the function to be unreliable.

The implementation behaves as though no other function calls the mbtowc() function.

Notes

AES Support Level: Full use
Return Values

When the $s$ parameter is not a null pointer, the $\text{mbtowc}$ function returns the following values:

- When $s$ points to a valid multibyte character other than null, $\text{mbtowc}$ returns the number of bytes in the character unless the character contains more than the number of bytes specified by the $n$ parameter.
- When $s$ points to a null character, $\text{mbtowc}$ returns 0 (zero).
- When $s$ does not point to a valid multibyte character or points to a character having more than the number of bytes expressed by the $n$ parameter, $\text{mbtowc}$ returns -1 and sets $\text{errno}$ to indicate the error.

When the $s$ parameter is a null pointer, the return value depends on the environment in which the $\text{mbtowc}$ function is called, as follows:

- In environments where encoding is not state dependent, $\text{mbtowc}$ returns 0 (zero).
- In environments where encoding is state dependent, $\text{mbtowc}$ returns a nonzero value.

In no case is the return value greater than the value specified by the $n$ parameter or the value of the $\text{MB_CUR_MAX}$ macro.

Errors

If the $\text{mbtowc}$ function fails, $\text{errno}$ may be set to the following value:

$\text{[EINVAL]}$ The $s$ parameter points to an invalid multibyte character.

Related Information

Functions: $\text{mblen(3), wctomb(3), mbstowcs(3), wcstombs(3)}$
memccpy(3)

**Purpose**
Performs memory operations

**Library**
Standard C Library (libc.a)

**Synopsis**
#include <string.h>

void *memccpy(
    void *sl,
    const void *s2,
    int c,
    size_t n);

void *memchr(
    const void *s,
    int c,
    size_t n);

int memcmp(
    const void *sl,
    const void *s2,
    size_t n);

void *memcpy(
    void *sl,
    const void *s2,
    size_t n);

void *memmove(
    void *sl,
    const void *s2,
    size_t n);

void *memset(
    void *s,
    int c,
    size_t n);
Parameters

- $s$  Points to the location of a string.
- $sl$  Points to the location of a destination string.
- $s2$  Points to the location of a source string.
- $c$  Specifies a character for which to search.
- $n$  Specifies the number of characters to search.

Description

The `memccpy()`, `memchr()`, `memcmp()`, `memcpy()`, `memset()`, and `memmove()` functions operate on strings in memory areas. A memory area is a group of contiguous characters bound by a count and not terminated by a null character. These memory functions do not check for overflow of the receiving memory area. All of these memory functions are declared in the `string.h` header file.

The `memccpy()` function sequentially copies characters from the location pointed to by the $sl$ parameter into the location pointed to by the $s2$ parameter until one of the following occurs:

- The character specified by the $c$ parameter (which is converted to an `unsigned int`) is encountered.
- The number of characters specified by the $n$ parameter have been copied to the string at location $sl$.

A pointer to character $c$ in the string pointed to by $sl$ is returned. When character $c$ is not encountered after $n$ characters have been copied to the string at location $sl$, a null pointer is returned.

The `memchr()` function sequentially searches the string at the location pointed to by the $s$ parameter until one of the following occurs:

- The character specified by the $c$ parameter (which is converted to an `unsigned int`) is encountered.
- The number of characters specified by the $n$ parameter have been copied to the string at location $s$.

A pointer to character $c$ in the string pointed to by $s$ is returned. When character $c$ is not encountered after $n$ characters have been copied to the string at location $s$, a null pointer is returned.

The `memcmp()` function compares the first $n$ characters (which are converted to `unsigned char`) of the string pointed to by the $sl$ parameter with the first $n$ characters (also interpreted as `unsigned char`) of the string pointed to by the $s2$ parameter.
The `memcmp()` function uses native character comparison, which may have signed values on some machines. This function returns one of the following values:

**Less than 0**  When `s1` is less than `s2`
**Equal to 0**   When `s1` is equal to `s2`
**Greater than 0**  When `s1` is greater than `s2`

The `memcpy()` function copies `n` characters from the string pointed to by the `s2` parameter into the location pointed to by the `s1` parameter. When copying overlapping strings, the behavior of this function is unreliable.

The `memset()` function copies the value of the character specified by the `c` parameter (which is converted to an `unsigned char`) into each of the first `n` locations of the string pointed to by the `s` parameter.

The `memmove()` function copies `n` characters from the string at the location pointed to by the `s2` parameter to the string at the location pointed to by the `s1` parameter. Copying takes place as though the `n` number of characters from string `s2` are first copied into a temporary location having `n` bytes that do not overlap either of the strings pointed to by `s1` and `s2`. Then, `n` number of characters from the temporary location are copied to the string pointed to by `s1`. Consequently, this operation is nondestructive and proceeds from left to right.

**Notes**

**AES Support Level:** Full use (`memchr()`, `memcmp()`, `memcpy()`, `memmove()`, `memset()`)
Trial use (`memccpy()`)  

**Return Values**

The `memccpy()` function returns a pointer to the character following the character specified by the `c` parameter in the string pointed to by the `s1` parameter. When character `c` is not found after the number of characters specified by the `n` parameter are scanned, a null pointer is returned.

The `memccpy()` function returns a pointer to the character specified by the `c` parameter. When character `c` does not occur after `n` characters in the string pointed to by the `s` parameter are scanned, a null pointer is returned.

The `memcmp()` function returns a value greater than, equal to, or less than 0 (zero), accordingly as the string pointed to by the `s1` parameter has a value greater than, equal to, or less than the string pointed to by the `s2` parameter.
The `memcpy()` and `memmove()` functions return the string pointed to by the `s1` parameter.

The `memset()` function returns the string pointed to by the `s` parameter.

**Related Information**

Functions: `string(3), swab(3)`
mkdir

Purpose
Creates a directory

Synopsis
#include <sys/stat.h>
#include <sys/types.h>

int mkdir (  
    const char *path,
    mode_t mode);

Parameters

path
Specifies the name of the new directory. If NFS is installed on your system, this path can cross into another node. In this case, the new directory is created at that node. If the final component of the path parameter refers to a symbolic link, the link is traversed and pathname resolution continues.

mode
Specifies the mask for the read, write, and execute (RWX) flags for owner, group, and others.

Description

The mkdir() function creates a new directory with the following attributes:

- The owner ID is set to the process’s effective user ID.
- The group ID is set to the group ID of its parent directory.
- Permission and attribute bits are set according to the value of the mode parameter modified by the process’s file creation mask (see the umask() function). This parameter is constructed by logically ORing values described in the sys/stat.h header file.
- The new directory is empty, except for . (dot) and .. (dot-dot).

To execute the mkdir() function, a process must have search permission to get to the parent directory of the path parameter and write permission in the parent directory of the path parameter with respect to all of the system’s configured access control policies.
Upon successful completion, the `mkdir()` function marks the `st_atime`, `st_ctime`, and `st_mtime` fields of the directory for update, and marks the `st_ctime` and `st_mtime` fields of the new directory's parent directory for update.

### Notes

**AES Support Level:** Full use

### Return Values

Upon successful completion, the `mkdir()` function returns a value of 0 (zero). If the `mkdir()` function fails, a value of -1 is returned, and `errno` is set to indicate the error.

### Errors

If the `mkdir()` function fails, the directory is not created and `errno` may be set to one of the following values:

- **[EACCES]** Creating the requested directory requires writing in a directory with a mode that denies write permission, or search permission is denied on the parent directory of the directory to be created.
- **[EEXIST]** The named file already exists.
- **[EMLINK]** The link count of the parent directory would exceed LINK_MAX.
- **[ELOOP]** Too many links were encountered in translating `path`.
- **[EFAULT]** The `path` parameter is an invalid address.
- **[ENAMETOOLONG]** The length of the `path` parameter exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- **[ENOENT]** A component of the `path` parameter does not exist or points to an empty string.
- **[EROFS]** The named file resides on a read-only file system.
- **[ENOSPC]** The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.
- **[EDQUOT]** The directory in which the entry for the new link is being placed cannot be extended because the user's quota of disk blocks or i-nodes on the file system containing the directory is exhausted.
- **[ENOTDIR]** A component of the path prefix is not a directory.
Related Information

Functions: chmod(2), mknod(2), rmdir(2), umask(2)
Commands: chmod(1), mkdir(1), mknod(8)
mkfifo

Purpose
Creates a FIFO

Library
Standard C Library (libc.a)

Synopsis
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo (const char *path,
            mode_t mode);

Parameters
path
Names the new file. If the final component of the path parameter
names a symbolic link, the link will be traversed and pathname
resolution will continue.

mode
Specifies the type, attributes, and access permissions of the file. This
parameter is constructed by logically ORing values described in the
sys/mode.h header file.

Description
The mkfifo() function is an interface to the mknod() function, where the file to be
created is a FIFO special file. No special system privileges are required.

Upon successful completion, the mkfifo() function marks the st_atime, st_ctime,
and st_mtime fields of the file for update, and sets the st_ctime and st_mtime
fields of the directory that contains the new entry for update.

Notes
AES Support Level: Full use
Return Values

Upon successful completion of mkfifo(), a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the mkfifo() function fails, the new file is not created and errno may be set to one of the following values:

- [EACCES] A component of the path prefix denies search permission, or write permission is denied on the parent directory of the FIFO to be created.
- [EPERM] The mode parameter specifies a file type other than S_IFIFO and the calling process does not have the DEV_CONFIG system privilege.
- [EEXIST] The named file exists.
- [EROFS] The directory in which the file is to be created is located on a readonly file system.
- [ENOSPC] The directory that would contain the new file cannot be extended or the file system is out of file allocation resources.
- [EDQUOT] The directory in which the entry for the new link is being placed cannot be extended because the user's quota of disk blocks or inodes on the file system is exhausted.
- [ELOOP] Too many links were encountered in translating path.
- [ENAMETOOLONG] The length of the path parameter exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- [ENOENT] A component of the path prefix does not exist or the path parameter points to an empty string.
- [ENOTDIR] A component of the path prefix is not a directory.

Related Information

Functions: chmod(2), mkdir(2), mknod(2), open(2), stat(2), umask(2)
Commands: chmod(1), mkdir(1)
mknod

Purpose
Creates an FIFO or special file

Library
Standard C Library (libc.a)

Synopsis

```c
#include <sys/types.h>
#include <sys/stat.h>

int mknod (const char *path, int mode, dev_t device);
```

Parameters

- **path**
  Names the new file. If the final component of the path parameter names a symbolic link, the link will be traversed and pathname resolution will continue.

- **mode**
  Specifies the file type, attributes, and access permissions. This parameter is constructed by logically ORing values described in the sys/mode.h header file.

- **device**
  Depends upon the configuration and is used only if the mode parameter specifies a block or character special file. If the file you specify is a remote file, the value of the device parameter must be meaningful on the node where the file resides.

Description

The **mknod**() function creates a special file or FIFO. Using the **mknod**() function to create file types other than FIFO special requires superuser privilege.

For the **mknod**() function to complete successfully, a process must have search permission and write permission in the parent directory of the path parameter.

The new file has the following characteristics:

- File type as specified by the mode parameter.
- Owner ID set to the process effective user ID.
Group ID set to the group ID of its parent directory.

- Permission and attribute bits set according to the value of the mode parameter. All bits set in the process file mode creation mask are cleared. See the umask() function.

**Return Values**

Upon successful completion of the mknod() function a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

**Errors**

If the mknod() function fails, the new file is not created and errno may be set to one of the following values:

- [EACCES] A component of the path prefix denies search permission, or write permission is denied on the parent directory of the FIFO to be created.
- [EPERM] The mode parameter specifies a file type other than FIFO and the calling process does not have the sufficient privilege.
- [EEXIST] The named file exists.
- [EROFS] The directory in which the file is to be created is located on a read-only file system.
- [ENOSPC] The directory that would contain the new file cannot be extended or the file system is out of file allocation resources.
- [EDQUOT] The directory in which the entry for the new link is being placed cannot be extended because the user's quota of disk blocks or inodes on the file system is exhausted.
- [ENAMETOOLONG] The length of the path parameter exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- [ENOENT] A component of the path prefix does not exist or the path parameter points to an empty string.
- [ENOTDIR] A component of the path prefix is not a directory.
Related Information

Functions: chmod(2), mkdir(2), open(2), umask(2), stat(2)

Commands: chmod(1), mkdir(1)
mktemp, mkstemp

Purpose
Constructs a unique filename

Library
Standard C Library (libc.a),
Berkeley Compatibility Library (libbsd.a)

Synopsis

```c
char *mktemp (char *template);
char *mkstemp (char *template);
```

Parameters

- `template` Points to a string to be replaced with a unique filename. The string in the `template` parameter must be a filename with six trailing "X"s.

Description
The `mktemp()` function replaces the contents of the string pointed to by the `template` parameter with a unique filename.

Notes
To get the BSD version of this function, compile with the Berkeley Compatibility Library (libbsd.a).

The `mkstemp()` function performs the same substitution to the template name and also opens the file for reading and writing.

In BSD systems, the `mkstemp()` function was intended to avoid a race condition between generating a temporary name and creating the file. Because the name generation in this system is more random, this race condition is less likely.

Return Values
Upon successful completion, the `mktemp()` function returns the address of the string pointed to by the `template` parameter.
If the string pointed to by the \emph{template} parameter contains no "X"s, or if the \texttt{mktemp()} function is unable to construct a unique filename, the first character of the \emph{template} parameter string is replaced with a null character, and a null pointer is returned.

Upon successful completion, the \texttt{mkstemp()} function returns an open file descriptor. If the \texttt{mkstemp()} function fails, it returns a value of -1.

\textbf{Related Information}

Functions: \texttt{tmpfile(3), tmpnam(3), getpid(2)}
mktimer

Purpose
Allocates a per-process timer

Library
Standard C Library (libc.a)

Synopsis
#include <sys/timers.h>

timer_t mktimer(
    int clock_type,
    int notify_type,
    void *reserved);

Parameters

clock_type Specifies the system-wide clock to be used as a per-process time base for the new timer.

notify_type Specifies the mechanism by which a process is to be notified when the per-process timer times out.

reserved Not used.

Description
The mktimer() function is used to allocate a per-process timer using a specified system-wide clock as its timebase. The mktimer() function returns a unique timer ID of type timer_t, which is used to identify the timer in per-process timer requests.

Each implementation of per-process timers defines a set of clocks that can be used as a time base for per-process timers, and one or more mechanisms for notifying the process that a per-process timer has expired. OSF/1 allows each process to allocate one per-process timer whose clock_type parameter is specified by the TIMEOFDAY symbolic constant, which is defined in the timers.h include file, using the notification mechanism whose notify_type parameter is specified by the DELIVERY_SIGNALS symbolic constant.

When the notify_type parameter is specified as DELIVERY_SIGNALS, the system sends a SIGALRM signal to the process whenever the timer expires.
Notes

Per-process timers are not inherited by a child process across `fork()` or `exec()` functions.

The `reserved` parameter is not currently used, but is specified for future support of other delivery mechanisms.

The `mktimer()` function is part of the POSIX 1003.4 real time extensions, which is not an approved standard. As such, it is liable to change.

AES Support Level: Trial use

Return Values

Upon successful completion, the `mktimer()` function returns a `timer_t` value, which may be passed to a per-process timer call. Otherwise, `mktimer()` returns a value of `(timer_t)-1` and sets `errno` to indicate the error.

Errors

If the `mktimer()` function fails, `errno` may be set to one of the following values:

[EAGAIN] The calling process has already allocated all available timers.

[EINVAL] The `clock_type` or `notify_type` parameter is invalid.

Related Information

Functions: `exec(2)`, `fork(2)`, `getclock(3)`, `gettimer(3)`, `reltimer(3)`, `rmtimer(3)`, `setclock(3)`
mmap

Purpose
Maps file system object into virtual memory

Synopsis
```
#include <sys/types.h>
#include <sys/mman.h>
caddr_t mmap (  
caddr_t addr,  
size_t len,  
int prot,  
int flags,  
int filedes,  
off_t off);
```

Parameters

- **addr**: Specifies the starting address of the new region.
- **len**: Specifies the length in bytes of the new region.
- **prot**: Specifies access permissions as any combination of PROT_READ, PROT_WRITE and PROT_EXEC ORed together, or PROT_NONE.
- **flags**: Specifies attributes of the mapped region with any combination of MAP_FILE, MAP_ANONYMOUS, MAP_VARIABLE, MAP_FIXED, MAP_SHARED, or MAP_PRIVATE, ORed together.
- **filedes**: Specifies the file to be mapped to the new mapped file region.
- **off**: Specifies the offset for the address.

Description

The `mmap()` function creates a new mapped file or shared memory region.

The `addr` and `len` parameters specify the requested starting address and length in bytes for the new region. This address is a multiple of the page size returned by `sysconf(_SC_PAGE_SIZE)`.

If the `len` parameter is not a multiple of the page size returned by `sysconf(_SC_PAGE_SIZE)`, then the result of any reference to an address between the end of the region and the end of the page containing the end of the region is undefined.
The *flags* parameter specifies attributes of the mapped region. Values of the *flags* parameter are constructed by bitwise-inclusive ORing flags from the following list of symbolic names defined in the `sys/mman.h` file:

- **MAP_FILE** Create a mapped file region.
- **MAP_ANONYMOUS** Create an unnamed memory region.
- **MAP_VARIABLE** Place region at the computed address.
- **MAP_FIXED** Place region at fixed address.
- **MAP_SHARED** Share changes.
- **MAP_PRIVATE** Changes are private.

The **MAP_FILE** and **MAP_ANONYMOUS** flags control whether the region to be mapped is a mapped file region or an anonymous shared memory region. Exactly one of these flags must be selected.

If **MAP_FILE** is set in the *flags* parameter:

- A new mapped file region is created, mapping the file associated with the `filedes` parameter.
- The `off` parameter specifies the file byte offset at which the mapping starts. This offset must be a multiple of the page size returned by `sysconf(_SC_PAGE_SIZE)`.
- If the end of the mapped file region is beyond the end of the file, the result of any reference to an address in the mapped file region corresponding to an offset beyond the end of the file is unspecified.

If **MAP_ANONYMOUS** is set in the *flags* parameter:

- A new memory region is created and initialized to all zeros. This memory region can be shared only with descendents of the current process.
- If the `filedes` parameter is not -1, the **mmap()** function fails.

The new region is placed at the requested address if the requested address is not null and it is possible to place the region at this address. The **MAP_VARIABLE** and **MAP_FIXED** flags control the placement of the region when the requested address is null or the region cannot be placed at the requested address. A region is never placed at address zero, or at an address where it would overlap with an existing region. Exactly one of these flags must be selected.
If MAP_VARIABLE is set in the flags parameter:
- If the requested address is null, or if it is not possible for the system to place the region at the requested address, the region is placed at an address selected by the system.

If MAP_FIXED is set in the flags parameter:
- If the requested address is not null, and it is not possible for the region to be placed at this address, the mmap() function fails.
- If the requested address is null, the region is placed at the default exact mapping address for the region. If there is no default exact mapping address for the region, the region is placed at an address selected by the system, and this address becomes the default exact mapping address for all subsequent attempts to map the same region, until all mappings of the region are unmapped. If it is not possible to place the region at the default exact mapping address, the mmap() function fails. Two mapped file regions are considered the same region for the purpose of default exact mapping if they map the same file and start at the same file offset.

The MAP_PRIVATE and MAP_SHARED flags control the visibility of modifications to the mapped file or shared memory region. Exactly one of these flags must be selected.

If MAP_SHARED is set in the flags parameter:
- If the region is a mapped file region, modifications to the region are visible to other processes which have mapped the same region using MAP_SHARED.
- If the region is a mapped file region, modifications to the region are written to the file.

If MAP_PRIVATE is set in the flags parameter:
- Modifications to the mapped region by the calling process are not visible to other processes which have mapped the same region using either MAP_PRIVATE or MAP_SHARED.
- Modifications to the mapped region by the calling process are not written to the file.

It is unspecified whether modifications by processes which have mapped the region using MAP_SHARED are visible to other processes which have mapped the same region using MAP_PRIVATE.
The `prot` parameter specifies the mapped region’s access permissions. The `sys/mman.h` header file defines the following access options:

**PROT_READ**

The mapped region can be read.

**PROT_WRITE**

The mapped region can be written.

**PROT_EXEC**

The mapped region can be executed.

**PROT_NONE**

The mapped region cannot be accessed.

The `prot` parameter can be `PROT_NONE` or any combination of `PROT_READ`, `PROT_WRITE`, and `PROT_EXEC` ORed together. If `PROT_NONE` is not specified, access permissions may be granted to the region in addition to those explicitly requested, except that write access is not granted unless `PROT_WRITE` is specified.

If the region is a mapped file that was mapped with `MAP_SHARED`, the `mmap()` function grants read or execute access permission only if the file descriptor used to map the file is open for reading, and grants write access permission only if the file descriptor used to map the file is open for writing. If the region is a mapped file which was mapped with `MAP_PRIVATE`, the `mmap()` function grants read, write, or execute access permission only if the file descriptor used to map the file is open for reading. If the region is a shared memory region which was mapped with `MAP_ANONYMOUS`, the `mmap()` function grants all requested access permissions.

After the successful completion of the `mmap()` function, the `filedes` parameter may be closed without effect on the mapped region or on the contents of the mapped file. Each mapped region creates a file reference, similar to an open file descriptor, which prevents the file data from being deallocated.

Whether modifications made to the file using the `write()` function are visible to mapped regions, and whether modifications to a mapped region are visible with the `read()` function, is undefined, except for the effect of the `msync()` function.

After a call to the `fork()` function, the child process inherits all mapped regions with the same sharing and protection attributes as in the parent process. Each mapped file and shared memory region created with the `mmap()` function is unmapped by a successful call to any of the `exec` functions, unless that region is made inheritable across `exec`. 
Notes

Note that memory acquired with the `mmap()` function is not locked, regardless of the previous use of the `plock()` function.

AES Support Level: Trial use

Return Values

Upon successful completion, the `mmap()` function returns the address at which the mapping was placed. Otherwise, `mmap()` returns `(caddr_t)-1` and sets `errno` to indicate the the error.

Errors

If the `mmap()` function fails, `errno` may be set to one of the following values:

- **[EACCES]** The file referred to by `filedes` is not open for read access, or the file is not open for write access and PROT_WRITE was set for a MAP_SHARED mapping operation.
- **[EBADF]** The `filedes` parameter is not a valid file descriptor.
- **[EINVAL]** The `flags` or `prot` parameter is invalid, or the `addr` parameter or `off` parameter is not a multiple of the page size returned by `sysconf(_SC_PAGE_SIZE)`.
- **[ENODEV]** The file descriptor `filedes` refers to an object that cannot be mapped, such as a terminal.
- **[ENOMEM]** There is not enough address space to map `len` bytes, or MAP_FIXED was set and part of the address space range `[addr, addr + len)` is already allocated.
- **[ENXIO]** The addresses specified by the range `[off, off + len)` are invalid for `filedes`.
- **[EINVAL]** MAP_ANONYMOUS was specified in `flags` and `filedes` is not -1.
- **[EFAULT]** The `addr` parameter is an invalid address.

Related Information

Functions: `fcntl(2)`, `fork(2)`, `madvise(2)`, `mprotect(2)`, `msync(2)`, `munmap(2)`, `plock(2)`, `sysconf(3)`
mount, umount

Purpose     Mounts or unmounds a file system

Synopsis    #include <sys/mount.h>

mount(
    int type,
    char *dir,
    int mnt_flag,
    caddr_t data);

umount(
    char *dir,
    int umnt_flag);

Parameters

*type*     Defines the type of the file system. The types of file systems defined
          in the sys/mount.h header file are MOUNT_NONE, MOUNT_UFS,
          MOUNT_NFS, MOUNT_MFS, and MOUNT_S5FS.

dir        Points to a null-terminated string containing the appropriate
          pathname.

mnt_flag    Specifies whether certain semantics should be suppressed when
          accessing the file system. Valid flags are:

M_RDONLY    The file system should be treated as read-only; no writing is
          allowed (even by a process with appropriate privilege). Physically
          write-protected and magnetic tape file systems must be mounted
          read-only or errors will occur when access times are updated,
          whether or not any explicit write is attempted.

M_NOEXEC    Do not allow files to be executed from the file system.

M_NOSUID    Do not honor setuid or setgid bits on files when executing
          them.
Description

The mount() function mounts a file system on the directory pointed to by the dir parameter. Following the mount, references to dir refer to the root directory on the newly mounted file system.

The dir parameter must point to a directory that already exists. Its old contents are inaccessible while the file system is mounted.

The umount() function unmounts a file system mounted at the directory pointed to by the dir parameter. The associated directory reverts to its ordinary interpretation.

To call either the mount() or umount() function, the calling process must have superuser privilege.
Notes

Two \texttt{mount()} functions are supported by OSF/1: the BSD \texttt{mount()} and the System V \texttt{mount()}. The default \texttt{mount()} function is the BSD \texttt{mount()} documented on this reference page. To use the System V version of \texttt{mount()}, you must link with the \texttt{libsys5} library before you link with \texttt{libc}.

Return Value

The \texttt{mount()} function returns 0 (zero) if the file system was successfully mounted. Otherwise, -1 is returned. The mount can fail if the \texttt{dir} parameter does not exist or is not a directory. For a UFS or S5FS file system, the mount can fail if the special device specified in the \texttt{ufs\_args} structure is inaccessible, is not an appropriate file, or is already mounted. A UFS, MFS, or S5FS mount can also fail if there are already too many file systems mounted.

The \texttt{umount()} function returns 0 (zero) if the file system was successfully unmounted. Otherwise, -1 is returned. The unmount will fail if there are active files in the mounted file system.

Errors

If the \texttt{mount()} function fails, \texttt{errno} may be set to one of the following values:

- \texttt{EPERM} The caller does not have appropriate privilege.
- \texttt{ENAMETOOLONG} A component of a pathname exceeded \texttt{Name\_Max} characters, or an entire pathname exceeded \texttt{Path\_Max} characters.
- \texttt{ELOOP} Too many symbolic links were encountered in translating a pathname.
- \texttt{ENOENT} A component of the \texttt{dir} parameter does not exist.
- \texttt{ENOTDIR} A component of the \texttt{name} parameter is not a directory, or a path prefix of the \texttt{special} parameter is not a directory.
- \texttt{EINVAL} Another process currently holds a reference to the \texttt{dir} parameter.
- \texttt{EBUSY} The file system is not clean and \texttt{M\_FORCE} is not set.
- \texttt{EFAULT} The \texttt{dir} parameter points outside the process' allocated address space.

The following errors can occur for a UFS or S5FS file system mount:

- \texttt{ENODEV} A component of \texttt{ufs\_args fspec} does not exist.
- \texttt{ENOTBLK} The \texttt{fspec} field is not a block device.
The major device number of \texttt{fspec} is out of range (this indicates no device driver exists for the associated hardware).

The device pointed to by the \texttt{fspec} field is already mounted.

No space remains in the mount table.

The super block for the file system had a bad magic number or an out of range block size.

Not enough memory was available to read the cylinder group information for the file system.

An I/O error occurred while reading the super block or cylinder group information.

The \texttt{fspec} field points outside the process’ allocated address space.

The following errors can occur for a NFS compatible file system mount:

NFS timed out trying to contact the server.

Some part of the information described by \texttt{nfs_args} points outside the process’ allocated address space.

The following errors can occur for a MFS file system mount:

No space remains in the mount table.

The super block for the file system had a bad magic number or an out of range block size.

Not enough memory was available to read the cylinder group information for the file system.

A paging error occurred while reading the super block or cylinder group information.

The \texttt{name} field points outside the process’ allocated address space.

If the \texttt{umount()} function fails, \texttt{errno} may be set to one of the following values:

The caller does not have appropriate privilege.

A component of the path is not a directory.
[EINVAL] The pathname contains a character with the high-order bit set.
[ENAMETOOLONG] A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters.
[ELOOP] Too many symbolic links were encountered in translating the pathname.
[EINVAL] The requested directory is not in the mount table.
[EBUSY] A process is holding a reference to a file located on the file system.
[EIO] An I/O error occurred while writing cached file system information.
[EFAULT] The dir parameter points outside the process’ allocated address space.

Related Information

Commands: mount(8)
mount

Purpose
Mounts a file system

Library
System V Compatibility Library (libsys5.a)

Synopsis
```c
#include <sys/mount.h>

int mount(
    char *spec,
    char *dir,
    int rwflag);
```

Parameters
- `spec` Points to the pathname of the file system to be mounted.
- `dir` Points to the pathname of the directory on which `spec` will be mounted.
- `rwflag` Specifies whether write permission is permitted on the mounted file system.

Description
The `mount()` function mounts a removable file system contained on the block special file pointed to by the `spec` parameter onto the directory pointed to by the `dir` parameter.

The `rwflag` parameter controls whether write permission is permitted on the new mounted file system. If `rwflag` is specified as 1, writing is forbidden. Otherwise, writing is permitted according to individual file accessibility.

The `mount()` function can only be invoked by the superuser.
Notes

Two `mount()` functions are supported by OSF/1: the BSD `mount()` and the System V `mount()`. The default `mount()` function is the BSD `mount()`. To use the version of `mount()` documented on this reference page, you must link with the `libsys5` library before you link with `libc`.

Return Value

The `mount()` function returns 0 (zero) if the file system was successfully mounted. Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors

If the `mount()` function fails, `errno` may be set to one of the following values:

- **[EPERM]** The effective user ID of the calling process is not root.
- **[ENOENT]** The `spec` or `dir` parameter points to a pathname that does not exist.
- **[ENOTDIR]** A component of the path prefix of either `spec` or `dir` is not a directory.
- **[ENOTBLK]** The device identified by `spec` is not a block-special device.
- **[ENXIO]** The device identified by `spec` does not exist.
- **[ENOTDIR]** The pathname pointed to by `dir` is not a directory.
- **[EBUSY]** Either `dir` has already been mounted onto, `dir` is a current working directory for some process, or `dir` is otherwise busy; or `spec` is already mounted; or the system mount table is full.

Related Information

Commands: `mount(8)`
madd, msub, mult, mdiv, pow, gcd, invert, rpow, msqrt, mcmp, move, min, omin, fmin, m_in, mout, omout, fmout, m_out, sdiv, itom

**Purpose**
Performs multiple precision integer arithmetic

**Library**
Object Code Library (libmp.a)

**Synopsis**
```c
#include <mp.h>
#include <stdio.h>

typedef struct mint { int len; short *val; } MINT;

madd(
    MINT *a,
    MINT *b,
    MINT *c);

msub(
    MINT *a,
    MINT *b,
    MINT *c);

mult(
    MINT *a,
    MINT *b,
    MINT *c);

mdiv(
    MINT *a,
    MINT *b,
    MINT *q,
    MINT *r);

pow(
    MINT *a,
    MINT *b,
    MINT *m,
    MINT *c);
```
Functions

\texttt{mp(3)}

\begin{verbatim}
gcd(
    MINT *a,
    MINT *b,
    MINT *c);
invert(
    MINT *a,
    MINT *b,
    MINT *c);

rpow(
    MINT *a,
    int n,
    MINT *c);

msqrt(
    MINT *a,
    MINT *b,
    MINT *r);
mcmp(
    MINT *a,
    MINT *b);
move(
    MINT *a,
    MINT *b);

min(
    MINT *a);
omin(
    MINT *a);
fmin(
    MINT *a,
    FILE *f);
m_in(
    MINT *a,
    int n,
    FILE *f);
mout(
    MINT *a);
omout(
    MINT *a);
\end{verbatim}


Description

These functions perform arithmetic on integers of arbitrary length. The integers are stored using the defined type MINT. Pointers to a MINT can be initialized using the itom() function, which sets the initial value to \( n \). After that, space is managed automatically by the routines.

The madd(), msub(), and mult() functions assign to \( c \) the sum, difference, and product, respectively, of \( a \) and \( b \).

The mdiv() function assigns to \( q \) and \( r \) the quotient and remainder obtained from dividing \( a \) by \( b \). The sdiv() function is like the mdiv() function except that the divisor is a short integer \( n \) and the remainder is placed in a short integer whose address is given as \( r \).

The msqrt() function produces the integer square root of \( a \) in \( b \) and places the remainder in \( r \).

The rpow() function calculates in \( c \) the value of \( a \) raised to the ("regular" integral) power \( n \), while the pow() function calculates this with a full multiple precision exponent \( b \) and the result is reduced modulo \( m \).

The gcd() function returns the greatest common denominator of \( a \) and \( b \) in \( c \), and the invert() function computes \( c \) such that \( a*c \mod b = 1 \), for \( a \) and \( b \) relatively prime.

The mcmp() function returns a negative, zero, or positive integer value when \( a \) is less than, equal to, or greater than \( b \), respectively.

The move() function copies \( a \) to \( b \).
The \texttt{min()} and \texttt{mout()} functions do decimal input and output while the \texttt{omin()} and \texttt{omout()} functions do octal input and output. More generally, the \texttt{fmin()} and \texttt{fmout()} functions do decimal input and output using file $f$, and \texttt{m_in()} and \texttt{m_out()} do input and output with arbitrary radix $n$.

On input, records should have the form of strings of digits terminated by a newline; output records have a similar form.

**Notes**

Programs which use the multiple-precision arithmetic library must be compiled with \texttt{-Imp}.  


mprotect

Purpose  Modifies access protections of memory mapping

Synopsis  
```
#include <sys/types.h>
#include <sys/mman.h>

int mprotect (  
    caddr_t addr,
    size_t len,
    int prot );
```

Parameters  
- **addr**: Specifies the address of the region to be modified.
- **len**: Specifies the length in bytes of the region to be modified.
- **prot**: Specifies access permissions as any combination of PROT_READ, PROT_WRITE, and PROT_EXEC ORed together, or PROT_NONE.

Description  
The mprotect() function modifies the access protection of a mapped file or shared memory region. The *addr* and *len* parameters specify the address and length in bytes of the region to be modified. The *len* parameter must be a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`. If *len* is not a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`, the length of the region will be rounded up to the next multiple of the page size.

The *prot* parameter specifies the new access protection for the region. The `sys/mman.h` header file defines the following access options:

- **PROT_READ**: The mapped region can be read.
- **PROT_WRITE**: The mapped region can be written.
- **PROT_EXEC**: The mapped region can be executed.
- **PROT_NONE**: The mapped region cannot be accessed.

The *prot* parameter can be PROT_NONE, or any combination of PROT_READ, PROT_WRITE, and PROT_EXEC ORed together. If PROT_NONE is not
specified, access permissions may be granted to the region in addition to those explicitly requested, except that write access will not be granted unless PROT_WRITE is specified.

If the region is a mapped file which was mapped with MAP_SHARED, the mprotect() function grants read or execute access permission only if the file descriptor used to map the file is open for reading, and grants write access permission only if the file descriptor used to map the file is open for writing. If the region is a mapped file which was mapped with MAP_PRIVATE, the mprotect() function grants read, write, or execute access permission only if the file descriptor used to map the file is open for reading. If the region is a shared memory region which was mapped with MAP_ANONYMOUS, the mprotect() function grants all requested access permissions.

The mprotect() function does not modify the access permission of any region which lies outside of the specified region, except that the effect on addresses between the end of the region and the end of the page containing the end of the region is unspecified.

If the mprotect() function fails under a condition other than that specified by [EINVAL], the access protection of some of the pages in the range [addr, addr + len) may have been changed. Suppose the error occurs on some page at an addr2; mprotect() may have modified the protections of all whole pages in the range [addr, addr2).

Notes

AES Support Level: Trial use

Return Values

Upon successful completion, the mprotect() function returns 0 (zero). Otherwise, mprotect() returns -1 and sets errno to indicate the error.

Errors

If the mprotect() function fails, errno may be set to one of the following values:

[EACCES] The prot parameter specifies a protection that conflicts with the access permission set for the underlying file.

[EINVAL] The prot parameter is invalid, or the addr parameter is not a multiple of the page size as returned by sysconf(_SC_PAGE_SIZE).
[EFAULT] The range \([addr, addr + len]\) includes an invalid address.

Related Information

Functions: \(\text{getpagesize}(2), \text{mmap}(2), \text{msync}(2), \text{sysconf}(3)\)
msem_init

Purpose
Initializes a semaphore in a mapped file or shared memory region

Library
Standard C Library (libc.a)

Synopsis
#include <sys/mman.h>
msemaphore *msem_init (msemaphore *sem,
   int initial_value);

Parameters
sem Points to an msemaphore structure in which the state of the semaphore is stored.
initial_value Determines whether the semaphore is locked or unlocked at allocation.

Description
The msem_init() function allocates a new binary semaphore and initializes the state of the new semaphore.

If the initial_value parameter is MSEM_LOCKED, the new semaphore is initialized in the locked state. If the initial_value parameter is MSEM_UNLOCKED, the new semaphore is initialized in the unlocked state.

The msemaphore structure is located within a mapped file or shared memory region created by a successful call to the mmap() function and having both read and write access.

If a semaphore is created in a mapped file region, any reference by a process which has mapped the same file, using a (struct msemaphore *) pointer which resolves to the same file offset, is taken as a reference to the same semaphore. If a semaphore is created in an anonymous shared memory region, any reference by a process which shares the same region, using a (struct msemaphore *) pointer which resolves to the same offset from the start of the region, is taken as a reference to the same semaphore.

Any previous semaphore state stored in the msemaphore structure is ignored and overwritten.
msem_init(3)

Notes

AES Support Level: Trial use

Return Values

Upon successful completion, the msem_init() function returns a pointer to the initialized msemaphore structure. On error, the msem_init() function returns null and sets errno to indicate the error.

Errors

If the msem_init() function fails, errno may be set to one of the following values:

[EINVAL] The initial_value parameter is not valid.
[ENOMEM] A new semaphore could not be created.

Related Information

Functions: mmap(2), msem_lock(3), msem_remove(3), msem_unlock(3)
**msem_lock**

**Purpose**  
Locks a semaphore

**Library**  
Standard C Library (*libc.a*)

**Synopsis**  
```c
#include <sys/mman.h>
int msem_lock (
    msemaphore *sem,
    int condition);
```

**Parameters**

- `sem`  
  Points to an `msemaphore` structure which specifies the semaphore to be locked.

- `condition`  
  Determines whether the `msem_lock()` function waits for a currently locked semaphore to unlock.

**Description**

The `msem_lock()` function attempts to lock a binary semaphore.

If the semaphore is not currently locked, it is locked and the `msem_lock()` function returns successfully.

If the semaphore is currently locked, and the `condition` parameter is `MSEM_IF_NOWAIT`, then the `msem_lock()` function returns with an error. If the semaphore is currently locked, and the `condition` parameter is 0 (zero), then `msem_lock()` will not return until either the calling process is able to successfully lock the semaphore, or an error condition occurs.

All calls to `msem_lock()` and `msem_unlock()` by multiple processes sharing a common `msemaphore` structure behave as if the calls were serialized.

If the `msemaphore` structure contains any value not resulting from a call to `msem_init()` followed by a (possibly empty) sequence of calls to `msem_lock()` and `msem_unlock()`, the results are undefined. The address of an `msemaphore` structure may be significant. If the `msemaphore` structure contains any value copied from an `msemaphore` structure at a different address, the result is undefined.
Notes

AES Support Level: Trial use

Return Values

On successful completion, the msem_lock() function returns 0 (zero). On error, the msem_lock() function returns -1 and sets errno to indicate the error.

Errors

If the msem_lock() function fails, errno may be set to one of the following values:

[EAGAIN] MSEM_IF_NOWAIT was specified and the semaphore was already locked.

[EINVAL] The sem parameter points to an msemaphore structure which specifies a semaphore which has been removed, or the condition parameter is invalid.

[EINTR] The msem_lock() function was interrupted by a signal which was caught.

Related Information

Functions: msem_init(3), msem_remove(3), msem_unlock(3)
msema_remove

**Purpose**
Removes a semaphore

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <sys/mman.h>
int msem_remove(
    msemaphore *sem);
```

**Parameters**

*sem* Points to an msemaphore structure which specifies the semaphore to be removed.

**Description**

The msem_remove() function removes a binary semaphore. Any subsequent use of the msemaphore structure before it is again initialized by calling the msem_init() function will have undefined results.

The msem_remove() function also causes any process waiting in the msem_lock() function on the removed semaphore to return with an error.

If the msemaphore structure contains any value not resulting from a call to the msem_init() function followed by a (possibly empty) sequence of calls to the msem_lock() and msem_unlock() functions, the result is undefined. The address of an msemaphore structure may be significant. If the msemaphore structure contains any value copied from an msemaphore structure at a different address, the result is undefined.

**Notes**

AES Support Level: Trial use
Return Values

On successful completion, the `msem_remove()` function returns 0 (zero). On error, the `msem_remove()` function returns -1 and sets `errno` to indicate the error.

Errors

If the `msem_remove()` function fails, `errno` may be set to the following value:

`[EINVAL]` The `sem` parameter points to an `msemaphore` structure which specifies a semaphore which has been removed.

Related Information

Functions: `msem_init(3), msem_lock(3), msem_unlock(3), munmap(2)`
msem_unlock

Purpose
Unlocks a semaphore

Library
Standard C Library (libc.a)

Synopsis
#include <sys/mman.h>

int msem_unlock (msemaphore *sem, int condition);

Parameters

sem       Points to an msemaphore structure which specifies the semaphore
to be unlocked.

condition Determines whether the msem_unlock() function unlocks the
semaphore if no other processes are waiting to lock it.

Description

The msem_unlock() function unlocks a binary semaphore.

If the condition parameter is 0 (zero), the semaphore is unlocked, whether or not
any other processes are currently attempting to lock it. If the condition parameter
is MSEM_IF_WAITERS, and another process is waiting to lock the semaphore or
it cannot be reliably determined whether some process is waiting to lock the
semaphore, the semaphore is unlocked by the calling process. If the condition
parameter is MSEM_IF_WAITERS, and no process is waiting to lock the
semaphore, the semaphore will not be unlocked and an error will be returned.

All calls to the msem_lock() and msem_unlock() functions by multiple processes
sharing a common msemaphore structure behave as if the calls were serialized.
If the `msemaphore` structure contains any value not resulting from a call to the `msem_init()` function followed by a (possibly empty) sequence of calls to the `msem_lock()` and `msem_unlock()` functions, the results are undefined. The address of an `msemaphore` structure may be significant. If the `msemaphore` structure contains any value copied from an `msemaphore` structure at a different address, the result is undefined.

**Notes**

**AES Support Level:** Trial use

**Return Values**

On successful completion, the `msem_unlock()` function returns 0 (zero). On error, the `msem_unlock()` function returns -1 and sets `errno` to indicate the error.

**Errors**

If the `msem_unlock()` function fails, `errno` may be set to one of the following values:

- `[EAGAIN]` MSEM_IF_WAITERS was specified and there were no waiters.
- `[EINVAL]` The `sem` parameter points to an `msemaphore` structure which specifies a semaphore which has been removed, or the `condition` parameter is invalid.

**Related Information**

Functions: `msem_init(3), msem_lock(3), msem_remove(3)`
msgctl

**Purpose**  Performs message control operations

**Synopsis**
```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl(
    int msqid,
    int cmd,
    struct msqid_ds *buf);
```

**Parameters**

- `msqid`  Specifies the message queue ID.
- `cmd`  Specifies the type of command. The possible commands and the operations they perform are as follows:

  **IPC_STAT**
  Queries the message queue ID by copying the contents of its associated data structure into the `buf` structure.

  **IPC_SET**
  Sets the message queue ID by copying values found in the `buf` structure into corresponding fields in the `msqid_ds` structure associated with the message queue ID. This is a restricted operation. The effective user ID of the calling process must be equal to that of superuser or equal to the value of `msg_perm.uid` or `msg_perm.cuid` in the associated `msqid_ds` structure. Only superuser can raise the value of `msg_qbytes`.

  **IPC_RMID**
  Removes the message queue ID and deallocates its associated `msqid_ds` structure. This is a restricted operation. The effective user ID of the calling process must be equal to that of superuser or equal to the value of `msg_perm.uid` or `msg_perm.cuid` in the associated `msqid_ds` structure.
Description

The `msgctl()` function allows a process to query or set the contents of the `msqid_ds` structure associated with the specified message queue ID. It also allows a process to remove the message queue ID and its associated `msqid_ds` structure. The `cmd` value determines which operation is performed.

The IPC_SET command uses the user-supplied contents of the `buf` structure to set the following members of the `msqid_ds` structure associated with the message queue ID:

- `msg_perm.uid`  
  The owner’s user ID.

- `msg_perm.gid`  
  The owner’s group ID.

- `msg_perm.mode`  
  The access modes for the queue. Only the low-order nine bits are set.

- `msg_qbytes`  
  The maximum number of bytes on the queue.

- `msg_ctime`  
  The time of the last `msgctl()` operation that changed the structure.

Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `msgctl()` function fails, `errno` may be set to one of the following values:

- [EINVAL] The `msqid` parameter is not a valid message queue ID, or the `cmd` parameter is not a valid command.

- [EACCES] The `cmd` parameter is IPC_STAT, but the calling process does not have read permission.
[EPERM] The \textit{cmd} parameter is equal to either IPC_RMID or IPC_SET, and the calling process does not have appropriate privilege.

[EPERM] The \textit{cmd} parameter is equal to IPC_SET, and an attempt is being made to increase the value of the \textit{msg_qbytes} parameter when the effective user ID of the calling process does not have the SET_OBJ_STAT system privilege.

[EFAULT] The \textit{cmd} parameter is IPC_STAT or IPC_SET. An error occurred in accessing the \textit{buf} structure.

\textbf{Related Information}

Functions: \texttt{msgget(2)}, \texttt{msgrcv(2)}, \texttt{msgsnd(2)}

Data Structures: \texttt{msqid_ds(4)}
Purpose

Returns (and possibly creates) the ID for a message queue.

Synopsis

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(
    key_t key,
    int msgflg);
```

Parameters

- `key` Specifies the key that identifies the message queue. The IPC_PRIVATE key can be used to assure the return of a new, unused, message queue ID.
- `msgflg` Specifies creation flags. Possible values are:
  - IPC_CREAT
    - If the key does not exist, the `msgget()` function creates a message queue ID using the given key. If the key does exist, it forces an error notification.
  - IPC_EXCL
    - If the key already exists, the `msgget()` function fails and returns an error notification.

The low-order nine bits of `msg_perm.mode` are set equal to the low-order nine bits of `msgflg`.

Description

The `msgget()` function returns (and possibly creates) the message queue ID for the message queue identified by the `key` parameter. If IPC_PRIVATE is used for `key`, the `msgget()` function returns the ID for a private (that is, newly created) message queue. The `msgflg` parameter supplies creation options for the `msgget()` function. If the `key` parameter does not already exist, IPC_CREAT instructs the `msgget()` function to create a new message queue for the key and return the kernel-assigned ID for the message queue.
After creating a new message queue ID, the \texttt{msgget()} function initializes the \texttt{msqid_ds} structure associated with the ID as follows:

- The \texttt{msg_perm.cuid} and \texttt{msg_perm.uid} members are set equal to the effective user ID of the calling process.
- The \texttt{msg_perm.cgid} and \texttt{msg_perm.gid} members are set equal to the effective group ID of the calling process.
- The low-order nine bits of the \texttt{msg_perm.mode} member are set equal to the low-order nine bits of \texttt{msgflg}.
- The \texttt{msg_qnum}, \texttt{msg_lspid}, \texttt{msg_lrpid}, \texttt{msg_stime}, and \texttt{msg_rtime} members are all set equal to zero.
- The \texttt{msg_ctime} member is set equal to the current time.
- The \texttt{msg_qbytes} member is set equal to the system limit.

\textbf{Return Value}

Upon successful completion, a message queue identifier is returned. If the \texttt{msgget()} function fails, a value of -1 is returned and \texttt{errno} is set to indicate the error.

\textbf{Errors}

If the \texttt{msgget()} function fails, \texttt{errno} may be set to one of the following values:

- \texttt{[EACCES]} A message queue identifier exists for the \texttt{key} parameter but operation permission, which is specified by the low-order nine bits of the \texttt{msgflg} parameter, is not granted.
- \texttt{[ENOENT]} A message queue identifier does not exist for the \texttt{key} parameter and the IPC_CREAT value is not set.
- \texttt{[ENOSPC]} A message queue identifier can be created, but the system-imposed limit on the maximum number of allowed message queue identifiers has been exceeded.
- \texttt{[EEXIST]} A message queue identifier exists for the \texttt{key} parameter, and both IPC_CREAT and IPC_EXCL are set.

\textbf{Related Information}

Functions: \texttt{msgctl(2), msgrcv(2), msgsnd(2)}

Data Structures: \texttt{msqid_ds(4)}
msgrcv

Purpose
Receives a message from a message queue

Synopsis
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgrcv(
    int msqid,
    struct msgbuf *msgp,
    int msgsiz,
    long msgtyp,
    int msgflg);

Parameters

msqid Specifies the ID of the message queue from which to receive the message.

msgp Specifies a pointer to the msgbuf structure that is to receive the message. See NOTES.

msgsiz Specifies the maximum number of bytes allowed for the received data.

msgtyp Specifies the message type to read from the queue.

msgflg Specifies the action to be taken by the kernel if there are no msgtyp messages on the queue.

Description
The msgrcv() function receives a message from the queue associated with the msqid parameter. It returns the number of bytes in the received message.

The msgp parameter points to a user-defined msgbuf structure. The structure will receive the message read from the queue.
The `msgsz` parameter specifies the maximum size allowed for the received data. If the message is longer than `msgsz`, the kernel will take one of the following actions, depending on whether the MSG_NOERROR flag is set:

- If MSG_NOERROR is not set, the kernel returns an [E2BIG] error to the calling process and leaves the message on the queue.
- If MSG_NOERROR is set, the kernel truncates the message to `msgsz` and discards the truncated portion without notifying the calling process.

The `msgtyp` parameter specifies the message type that the process wants to receive. Possible values and their results are as follows:

- 0: The process receives the message at the head of the queue.
- >0: The process receives the first message of the requested positive-integer type.
- <0: The process receives the first message of the lowest type on the queue. To qualify as the lowest type, the negative-integer type must be less than or equal to the absolute value of `msgtyp`.

The `msgflg` parameter specifies the action that the kernel should take if the queue does not contain a message of the requested type. Either of two kernel actions can be specified, as follows:

- If IPC_NOWAIT is set, the kernel returns immediately with a return value of -1 and `errno` set to [ENOMSG].
- If IPC_NOWAIT is not set, the kernel suspends the calling process. The process remains suspended until one of the following occurs:
  - A message of the requested type appears on the queue. In this case, the kernel wakes the process to receive the message.
  - The specified message queue ID is removed from the system. In this case, the kernel sets `errno` to [EIDRM] and returns -1 to the calling process.
  - The process catches a signal. In this case, the process does not receive the message and, instead, resumes execution as directed by the `signal()` call.

**Notes**

The user-specified `msgbuf` structure, used to store received messages, is defined as follows:

```c
struct msgbuf {
    mtyp_t mtype;
    char mtext[];
}
```

The `mtype` field is set to the message type supplied by the sender.
The `mtext` field is set to the message text. Unless MSG_NOERROR is set, the message size will be less than or equal to the `msgsz` specified on the call to `msgrcv()`.

**Return Values**

Upon successful completion, the `msgrcv()` function returns a value equal to the number of bytes actually stored in `mtext`. Also, the kernel updates the `msqid_ds` structure associated with the message queue ID as follows:

- Decrements `msg_qnum` by 1.
- Sets `msg_lrpid` equal to the process ID of the calling process.
- Sets `msg_rtime` equal to the current time.
- Decrements `msg_cbytes` by the message text size.

When the `msgrcv()` function fails, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `msgrcv()` function fails, `errno` may be set to one of the following values:

- **[EINVAL]** The `msqid` parameter is not a valid message queue ID, or the `msgsz` parameter is less than 0 (zero).
- **[EACCES]** The calling process does not have permission for the operation.
- **[EIDRM]** The `msgid` parameter has been removed from the system.
- **[E2BIG]** The number of bytes to be received in `mtext` is greater than `msgsz` and the MSG_NOERROR flag is not set.
- **[ENOMSG]** The queue does not contain a message of the requested type and the IPC_NOWAIT flag is set.
- **[EINTR]** The operation was interrupted by a signal.

**Related Information**

Functions: `msgctl(2)`, `msgget(2)`, `msgsnd(2)`, `sigaction(2)`

Data Structures: `msqid_ds(4)`
msgsnd

Purpose
Sends a message to a message queue

Synopsis
```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd(  
    int msqid,  
    struct msgbuf *msgp,  
    int msgsz,  
    int msgflg);
```

Parameters
- `msqid`: Specifies the ID of the message queue on which to place the message. The ID is typically returned by a previous `msgget()` function.
- `msgp`: Specifies a pointer to the `msgbuf` structure that contains the message. See NOTES.
- `msgsz`: Specifies the size of the data array in the `msgbuf` structure.
- `msgflg`: Specifies the action to be taken by the kernel if it runs out of internal buffer space.

Description
The `msgsnd()` function sends a message to the queue associated with the `msqid` parameter.

The `msgp` parameter points to a user-defined `msgbuf` structure. The structure identifies the message type and contains a data array with the message text.

The size of the data array is specified by the `msgsz` parameter. The `msgsz` value can be from zero to a system-defined maximum.

The `msgflg` parameter specifies the action that the kernel should take if either or both of the following are true:
- The current number of bytes in the message queue is equal to `msg_qbytes` (in the `msqid_ds` structure).
- The total number of messages on all message queues is equal to the system-defined limit.
Either of two kernel actions can be specified, as follows:

- If IPC_NOWAIT is set, the kernel does not send the message and returns to the calling process immediately.

- If IPC_NOWAIT is not set, the kernel suspends the calling process. The process remains suspended until one of the following occurs:
  - The blocking condition is removed. In this case, the kernel sends the message.
  - The specified message queue ID is removed from the system. In this case, the kernel sets errno to [EIDRM] and returns -1 to the calling process.
  - The process catches a signal. In this case, the message is not sent and the process resumes execution as directed by the signal() function.

If the msgsnd() function completes successfully, the kernel updates the msqid_ds structure associated with the msgid parameter. Specifically, it:

- Increments msg_qnum by 1.
- Increments msg_cbytes by the message text size.
- Sets msg_lspid equal to the process ID of the calling process.
- Sets msg_stime equal to the current time.

**Notes**

The user-specified msgbuf structure is defined as follows:

```c
struct msgbuf {
    mtyp_t mtype;
    char mtext[];
}
```

The mtype field is a user-chosen positive integer. A receiving process can use the message type to select only those messages it wants to receive from the queue. See the msgrcv() function.

The mtext field is any text of length msgsz.

When the kernel sends a message, it allocates space for the message and copies the data from user space. The kernel then allocates a msg (message header) structure, sets its fields, and inserts the structure at the tail of the message queue associated with the message queue ID. The msg structure is defined as follows:

```c
struct msg {
    struct msg *msg_next;
    long msg_type;
    long msg_ts;
    caddr_t msg_addr;
}
```
The `msg_next` field is a pointer to the next message in the queue. The `msg_type` field is the message type supplied in the user-specified `msgbuf` structure. The `msg_ts` field is the size of the message text. The `msg_addr` field is the address of the message text.

**Return Values**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `msgsnd()` function fails, `errno` may be set to one of the following values:

- **[EINVAL]** The `msqid` parameter is not a valid message queue ID, `mtype` is less than 1, or `msgsz` is less than 0 (zero) or greater than the system-defined limit.
- **[EACCES]** The calling process does not have permission for the operation.
- **[EAGAIN]** If the maximum number of message headers has been allocated or if the bytes for the message exceed the maximum number of bytes on the queue, the message cannot be sent and the IPC_NOWAIT flag is set.
- **[EINTR]** The operation was interrupted by a signal.
- **[EIDRM]** The `msqid` parameter has been removed from the system.

**Related Information**

Functions: `msgctl(2), msgget(2), msgrcv(2), sigaction(2)`

Data Structures: `msqid_ds(4)`
msync

Purpose
Synchronizes a mapped file

Synopsis
#include <sys/types.h>
#include <sys/mman.h>

int msync (caddr_t addr, size_t len, int flags);

Parameters
addr
Specifies the address of the region to be synchronized.
len
Specifies the length in bytes of the region to be synchronized.
flags Specifies one of the following symbolic constants defined in the sys/mman.h file:
   MS_SYNC
      Synchronous cache flush
   MS_ASYNC
      Asynchronous cache flush
   MS_INVALIDATE
      Invalidate cached pages

Description
The msync() function controls the caching operations of a mapped file region. The msync() function can be used to ensure that modified pages in the region are transferred to the file's underlying storage device, or to control the visibility of modifications with respect to file system operations.

The addr and len parameters specify the region to be synchronized. The len parameter must be a multiple of the page size as returned by sysconf(_SC_PAGE_SIZE). If len is not a multiple of the page size as returned by sysconf(_SC_PAGE_SIZE), the length of the region will be rounded up to the next multiple of the page size.

If the flags parameter is set to MS_SYNC, the msync() function does not return until the system completes all I/O operations. If the flags parameter is set to MS_ASYNC, the msync() function returns after the system schedules all I/O
functions. If the flags parameter is set to MS_INVALIDATE, the msync() function invalidates all cached copies of the pages. New copies of the pages then must be obtained from the file system the next time they are referenced.

After a successful call to the msync() function with the flags parameter set to MS_SYNC, all previous modifications to the mapped region are visible to processes using the read() parameter. Previous modifications to the file using the write() function may be lost.

After a successful call to the msync() function with the flags parameter set to MS_INVALIDATE, all previous modifications to the file using the write() function are visible to the mapped region. Previous direct modifications to the mapped region may be lost.

Notes

AES Support Level: Trial use

Return Values

Upon successful completion, the msync() function returns 0 (zero). Otherwise, the msync() function returns -1 and sets errno to indicate the error.

Errors

If the msync() function fails, errno may be set to one of the following values:

[EIO] An I/O error occurred while reading from or writing to the file system.

[ENOMEM] The range specified by [addr, addr + len) is invalid for a process’ address space, or the range specifies one or more unmapped pages.

[EINVAL] The addr parameter is not a multiple of the page size as returned by sysconf(_SC_PAGE_SIZE), or the flags parameter is MS_SYNC or MS_ASYNC and the region was mapped with MAP_PRIVATE.

[EFAULT] The range [addr, addr + len) includes an invalid address.

Related Information

Functions: fsync(2), mmap(2), read(2), sysconf(3), write(2)
munmap

Purpose Unmaps a mapped region

Synopsis

```
#include <sys/types.h>
#include <sys/mman.h>

int munmap (caddr_t addr, size_t len);
```

Parameters

- `addr` Specifies the address of the region to be unmapped.
- `len` Specifies the length in bytes of the region to be unmapped.

Description

The `munmap()` function unmaps a mapped file or shared memory region.

The `addr` and `len` parameters specify the address and length in bytes, respectively, of the region to be unmapped. The `len` parameter must be a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`. If `len` is not a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`, the length of the region will be rounded up to the next multiple of the page size.

The result of using an address which lies in an unmapped region and not in any subsequently mapped region is undefined.

Notes

**AES Support Level:** Trial use

Return Values

Upon successful completion, the `munmap()` function returns 0 (zero). Otherwise, `munmap()` returns -1 and sets `errno` to indicate the error.
Errors

If the `munmap()` function fails, `errno` may be set to one of the following values:

- **[EINVAL]** The `addr` parameter is not a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`.

- **[EFAULT]** The range `[addr, addr + len)` includes an invalid address.

Related Information

Functions: `mmap(2), sysconf(3)`
mvalid

Purpose Checks memory region for validity

Synopsis

```
#include <sys/types.h>
#include <sys/mman.h>

int mvalid (caddr_t addr, size_t len, int prot);
```

Parameters

- **addr**: Specifies the address of the region whose validity is to be checked.
- **len**: Specifies length in bytes of the region specified by the `addr` parameter.
- **prot**: Specifies the desired access protection for the region.

Description

The `mvalid()` function checks the validity of a memory region. A region is considered to be valid if accesses of the requested type are allowed to all addresses in the region.

The `sys/mman.h` header file defines the following access options:

- **PROT_READ**: The mapped region can be read.
- **PROT_WRITE**: The mapped region can be written.
- **PROT_EXEC**: The mapped region can be executed.

The `prot` parameter can be any combination of PROT_READ, PROT_WRITE, and PROT_EXEC ORed together.

Return Values

The `mvalid()` function returns 0 (zero) if accesses requiring the specified protection are allowed to all addresses within the specified range of addresses. Otherwise, the `mvalid()` function returns -1 and sets `errno` to indicate the error.
Errors

If the `mvalid()` function fails, `errno` may be set to one of the following values:

[EACCES] The range specified by `[addr, addr + len)` is invalid for the process' address space, or the range specifies one or more unmapped pages, or one or more pages of the range disallows accesses of the specified protection.

[EINVAL] The `prot` parameter is invalid, or the `addr` parameter is not a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`.

Related Information

Functions: `mmap(2), mprotect(2), sysconf(3)`
dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_forder, dbm_error, dbm_clearerr

Purpose
Database subroutines

Synopsis
#include <ndbm.h>

typedef struct {
   char *dptr;
   int dsize;
} datum;

DBM *dbm_open(
   char *file,
   int flags,
   int mode);

void dbm_close(
   DBM *db);

datum dbm_fetch(
   DBM *db,
   datum key);

int dbm_store(
   DBM *db,
   datum key,
   datum content,
   int flags);

int dbm_delete(
   DBM *db,
   datum key);

datum dbm_firstkey(
   DBM *db);

datum dbm_nextkey(
   DBM *db);

long dbm_forder(
   datum key);
Parameters

$db$  Specifies the database.

$file$  Specifies the file to be opened. If the $file$ parameter refers to a symbolic link, the $dbm\_open()$ function opens the file pointed to by the symbolic link. See the $open()$ manual page for further details.

$mode$  Specifies the read, write, and execute permissions of the file to be created (requested by the O_CREAT flag). If the file already exists, this parameter is ignored. This parameter is constructed by logically ORing values described in the $sys/mode.h$ header file. See the $open()$ manual page for further details.

$flags$  Specifies one of the following flags for opening:

1. **DBM\_INSERT**
   
   Only insert new entries into the database. Do not change an existing entry with the same key.

2. **DBM\_REPLACE**
   
   Replace an existing entry if it has the same key.

$key$  Specifies the key.

$content$  Specifies a value associated with $key$.

Description

The $dbm\_open()$, $dbm\_close()$, $dbm\_fetch()$, $dbm\_store()$, $dbm\_delete()$, $dbm\_firstkey()$, $dbm\_nextkey()$, $dbm\_forder()$, $dbm\_error()$, and $dbm\_clearerr()$ functions maintain key/content pairs in a database. The functions handle very large databases (a billion blocks) and access a keyed item in one or two file system accesses. Arbitrary binary data, as well as normal ASCII strings, are allowed.

The database is stored in two files. One file is a directory containing a bit map and has .dir as its suffix. The second file contains all data and has .pag as its suffix.

Before a database can be accessed, it must be opened by the $dbm\_open()$ function. The $dbm\_open()$ function opens (and if necessary, creates) the file.dir and file.pag files, depending on the $flags$ parameter.
Once open, the data stored under a key is accessed by the \texttt{dbm\_fetch()} function and data is placed under a key by the \texttt{dbm\_store()} function. A key (and its associated contents) is deleted by the \texttt{dbm\_delete()} function. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of the \texttt{dbm\_firstkey()} and \texttt{dbm\_nextkey()} functions. The \texttt{dbm\_firstkey()} function returns the first key in the database. The \texttt{dbm\_nextkey()} function returns the next key in the database. The order of keys presented by the \texttt{dbm\_firstkey()} and \texttt{dbm\_nextkey()} functions depends on a hashing function. The following code traverses the database:

\begin{verbatim}
for (key = dbm\_firstkey(db); key\texttt{.dptr} != NULL; key = dbm\_nextkey(db))
\end{verbatim}

The \texttt{dbm\_error()} function returns nonzero value when an error has occurred reading or writing the database. The \texttt{dbm\_clearerr()} function resets the error condition on the named database.

The \texttt{dbm\_forder()} function returns the block number in the .pag file that the specified key will map to.

\textbf{Return Values}

Upon successful completion, all functions that return an \texttt{int} return a value of 0 (zero). Otherwise, a negative value is returned. Routines that return a \texttt{datum} indicate errors with a null (0) \texttt{dptr}. If the \texttt{dbm\_store()} function is called with a \texttt{flags} value of DBM\_INSERT, and finds an existing entry with the same key, it returns 1.

\textbf{Related Information}

Functions: \texttt{dbm(3), open(2)}
Purpose Negates and returns the value of the double operand x

Library Math Library (libm.a)

Synopsis double neg(
    double x);

Parameters

    x Specifies some double value.

Description

The neg() function returns a negative of the value of the double operand x.
nfssvc

Purpose

Creates a remote NFS server

Synopsis

nfssvc(
    int sock,
    int mask,
    int match);

Parameters

sock Specifies the socket. The sock parameter must be in the AF_INET family and of type SOCK_DGRAM.

mask Specifies a mask to be supplied to the client host address.

match Specifies a value to be compared against the value of the client host address ANDed with the value in the mask parameter. If the values are equal, the daemon is started; otherwise, the request is dropped by the server.

Description

The nfssvc() function starts an NFS daemon listening on a specified socket.

Return Values

Normally this function does not return unless the server is terminated by a signal, at which time a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.

Errors

If the nfssvc() function fails, errno may be set to one of the following values:

[EBADF] An invalid file descriptor has been passed to the nfssvc() function.

[EPERM] The caller is not the superuser.

Related Information

Functions: async_daemon(2)
nice

Purpose Changes scheduling priority of a process

Library
Standard C Library (libc.a),
Berkeley Compatibility Library (libbsd.a)

Synopsis
int nice(
    int increment);

Parameters
increment
Specifications a value that is added to the current process priority. Negative values can be specified, although values exceeding either the high or low limit are truncated.

Description
The nice() function adds an increment to the nice value of the calling process. The nice value is a nonnegative number; by incrementing the nice value, a process is given lower CPU priority.

Notes
Process priorities in are defined in the range of 0 to 39 in AT&T System V systems, and in the range -20 to 20 in BSD systems. For that reason, two versions of the nice() function are supported by OSF/1. The default version, in libc, behaves like the AT&T System V version, with the increment parameter treated as the modifier of a value in the range of 0 to 39.

If the behavior of the BSD version is desired, compile with the Berkeley Compatibility Library (libbsd.a) and the increment parameter is treated as the modifier of a value in the range -20 to 20.

AES Support Level: Trial use
Return Values

Upon successful completion, the nice() function returns the new nice value minus the value of NZERO. Otherwise, -1 is returned and errno is set to indicate the error.

Errors

If the libc version of nice() fails, errno may be set to the following value:

[EPERM] The calling process does not have appropriate privilege.

If the libbsd version of nice() fails, errno may be set to the same values as the setpriority() function.

Related Information

Functions: exec(2), getpriority(2)
nl_langinfo

Purpose
Language information

Library
Standard C Library (libc.a)

Synopsis
#include <nl_types.h>
#include <langinfo.h>

char *nl_langinfo (  
    nl_item item );

char *nl_langinfo_r (  
    nl_item item ,  
    char *buf ,  
    int len );

Parameters

item    Specifies constant names and values.
buf     Points to a string containing the requested information.
len     Specifies the length of buf.

Description

The nl_langinfo() function returns a pointer to a string containing information relevant to the particular language or cultural area defined in the program's locale. The manifest, constant names and values of the item parameter are defined in the langinfo.h header file.

The nl_langinfo_r() function is the reentrant version of nl_langinfo().

Notes

AES Support Level: Trial use (nl_langinfo( ))
Example

For example, the following returns a pointer to the abbreviated name of the first
day of the week in the current locale:

```
nl_langinfo(ABDAY_1)
```

This function call would return a pointer to the string ‘‘Dom’’ if the identified
language was Portuguese, ‘‘Sun’’ if the identified language was English, and so on.

Return Values

In a locale where `langinfo` data is not defined, the `nl_langinfo()` function returns a
pointer to the corresponding string in the C locale. In all locales, the `nl_langinfo()`
function returns a pointer to an empty string if the `item` parameter contains an
invalid setting.

Upon successful completion, the `nl_langinfo_r()` function returns a value of 0
(zero) and places the requested information in `buf`. Otherwise, -1 is returned and
`errno` is set to indicate the error.

Errors

If the `nl_langinfo_r()` function fails, `errno` may be set to the following value:

```
EINVAL The `item` parameter is invalid.
```

Related Information

Functions: `setlocale(3)`
ns_addr, ns_ntoa

**Purpose**
Xerox NS address conversion routines

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <sys/types.h>
#include <netns/ns.h>

struct ns_addr ns_addr(
    char *cp);

char *ns_ntoa(
    struct ns_addr ns);
```

**Parameters**
- `cp` Points to a character string representing an XNS address.
- `ns` Specifies an XNS address.

**Description**
The `ns_addr()` function interprets character strings representing Xerox NS addresses, and returns binary information suitable for use in functions. The `ns_ntoa()` function takes XNS addresses and returns ASCII strings representing the address in a notation in common use in the Xerox development environment:

```
<network number> . <host number> . <port number>
```

Trailing zero fields are suppressed, and each number is printed in hexadecimal, in a format suitable for input to the `ns_addr()` function. Any fields lacking superdecimal digits will have a trailing "H" appended.

Unfortunately, no universal standard exists for representing XNS addresses. An effort has been made to insure that the `ns_addr()` function be compatible with most formats in common use.

The `ns_addr()` function first separates an address into one to three fields using a . (period), a : (colon), or a # (number sign) single delimiter. Each field is then examined for byte separators (colon or period). If there are byte separators, each subfield separated is taken to be a small hexadecimal number, and the entirety is taken as a network-byte-ordered quantity to be zero extended in the high-network-order bytes.
Next, the field is inspected for hyphens. If there are hyphens, the field is assumed to be a number in decimal notation with hyphens separating the millenia. Next, the field is assumed to be a number. It is interpreted as hexadecimal if there is a leading "0x" (as in C), a trailing "H" (as in Mesa), or if there are any superdecimal digits present. It is interpreted as octal if there is a leading 0 (zero) and there are no superoctal digits. Otherwise, it is converted as a decimal number.

Related Information

Files: hosts(4), networks(4)
ntohl

Purpose
Converts an unsigned long (32-bit) integer from Internet network-byte order to host-byte order

Library
Standard C Library (libc.a)

Synopsis
#include <netinet/in.h>
unsigned long ntohl (unsigned long netlong);

Parameters
netlong Specifies a 32-bit integer in network-byte order.

Description
The ntohl() (network-to-host long) function converts an unsigned long (32-bit) integer from Internet network-byte order to host-byte order.

The Internet network requires address and port reference data in network-byte order (most significant byte leftmost, least significant byte rightmost). You can use the ntohl() function to convert Internet network address and port data to host byte-ordered integers.

The ntohl() function is defined as a little-endian function in the netinet/in.h header file for machine environments where network-byte order and host-byte order are not identical.

Return Values
The ntohl() function returns a 32-bit long integer in host-byte order.

Related Information
Functions: ntohs(3), htonl(3), htons(3)
ntohs

Purpose
Converts an unsigned short (16-bit) integer from Internet network-byte order to host-byte order

Library
Standard C Library (libc.a)

Synopsis
#include <netinet/in.h>

unsigned short ntohs (unsigned short netshort);

Parameters
netshort Specifies a 16-bit integer in host-byte order.

Description
The ntohs() (network-to-host short) function converts an unsigned short (16-bit) integer from Internet network-byte order to host-byte order.

The Internet network requires address and port reference data in network-byte order (most significant byte leftmost, least significant byte rightmost). You can use the ntohs() function to convert Internet network address and port data to host-byte ordered integers.

The ntohs() function is defined as a little-endian function in thenetinet/in.h header file for machine environments where network-byte order and host-byte order are not identical.

Return Values
The ntohs() function returns a 16-bit short integer in host-byte order.

Related Information
Functions: ntohl(3), htonl(3), htons(3)
open, creat

Purpose
Opens a file for reading or writing

Synopsis
```
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

int open (const char *path, int oflag [, mode_t mode ]);  
int creat (const char *path, mode_t mode );
```

Parameters
- **path**: Specifies the file to be opened or created. If the path parameter refers to a symbolic link, the open() function opens the file pointed to by the symbolic link.
- **oflag**: Specifies the type of access, special open processing, the type of update, and the initial state of the open file. The parameter value is constructed by logically ORing special open processing flags. These flags are defined in the fcntl.h header file and are described below.
- **mode**: Specifies the read, write, and execute permissions of the file to be created (requested by the O_CREAT flag in the open() interface). If the file already exists, this parameter is ignored. This parameter is constructed by logically ORing values described in the sys/mode.h header file.

Description
The open() and creat() functions establish a connection between the file named by the path parameter and a file descriptor. The opened file descriptor is used by subsequent I/O functions, such as read() and write(), to access that file.

The returned file descriptor is the lowest file descriptor not previously open for that process. No process can have more than OPEN_MAX file descriptors open simultaneously.
The open() and creat() functions, which suspend the calling process until the request is completed, are redefined so that only the calling thread is suspended.

The file offset, marking the current position within the file, is set to the beginning of the file. The new file descriptor is set to remain open across exec functions. (See the fcntl() function.)

The file status flags and file access flags are designated by the oflag parameter. The oflag parameter is constructed by bitwise-inclusive ORing exactly one of the file access flags (O_RDONLY, O_WRONLY, or O_RDWR) with one or more of the file status flags.

File Access Flags

The file access flags are as follows:

- O_RDONLY The file is open for reading only.
- O_WRONLY The file is open for writing only.
- O_RDWR The file is open for reading and writing.

Exactly one of the file access values (O_RDONLY, O_WRONLY, or O_RDWR) must be specified. If none is set, O_RDONLY is assumed.

File Status Flags

File status flags that specify special open processing are as follows:

- O_CREAT If the file exists, this flag has no effect except as noted under O_EXCL. If the file does not exist, a regular file is created with the following characteristics:

  - The owner ID of the file is set to the effective user ID of the process.
  - The group ID of the file is set to the group ID of its parent directory.
  - The file permission and attribute bits are set to the value of the mode parameter, modified as follows:

    - All bits set in the process file mode creation mask are cleared.
    - The set-user ID attribute (S_ISUID bit) is cleared.
    - The set-group ID attribute (S_ISGID bit) is cleared.
    - The S_ISVTX attribute bit is cleared.
The calling process must have write permission to the file's parent directory with respect to all access control policies to create a new file.

O_EXCL  If O_EXCL and O_CREAT are set, the open fails if the file exists.

O_NOCTTY  If the path parameter identifies a terminal device, this flag assures that the terminal device does not become the controlling terminal for the process.

O_TRUNC  If the file does not exist, this flag has no effect. If the file exists and is a regular file, and if the file is successfully opened O_RDWR or O_WRONLY:
  - The length of the file is truncated to 0 (zero).
  - The owner and group of the file are unchanged.
  - The set-user ID attribute of the file mode is cleared.
  - The set-user ID attribute of the file is cleared.

The open fails if either of the following conditions are true:
  - The file supports enforced record locks and another process has locked a portion of the file.
  - The file does not allow write access.

If the oflag parameter also specifies O_SYNC, the truncation is a synchronous update.

A program can request some control over when updates should be made permanent for a regular file opened for write access.

File status flags that define the initial state of the open file are as follows:

O_SYNC  If set, updates and writes to regular files and block devices are synchronous updates. File update is performed by:
  - fclear()
  - ftruncate()
  - open() with O_TRUNC
  - write()

On return from a function that performs a synchronous update (any of the above system calls, when O_SYNC is set), the program is assured that all data for the file has been written to permanent storage, even if the file is also open for deferred update.
O_APPEND  If set, the file pointer is set to the end of the file prior to each write.

O_NONBLOCK, O_NDELAY
If set, the call to open() will not block, and subsequent read() or write() operations on the file will be nonblocking.

General Notes on oflag Parameter Flag Values

The effect of O_CREAT is immediate.

When opening a FIFO with O_RDONLY:

• If neither O_NDELAY nor O_NONBLOCK is set, the open() function blocks until another process opens the file for writing. If the file is already open for writing (even by the calling process), the open() function returns without delay.

• If O_NDELAY or O_NONBLOCK is set, the open() function returns immediately.

When opening a FIFO with O_WRONLY:

• If neither O_NDELAY nor O_NONBLOCK is set, the open() function blocks until another process opens the file for reading. If the file is already open for reading (even by the calling process), the open() function returns without delay.

• If O_NDELAY or O_NONBLOCK is set, the open() function returns an error if no process currently has the file open for reading.

When opening a block special or character special file that supports nonblocking opens, such as a terminal device:

• If neither O_NDELAY nor O_NONBLOCK is set, the open() function blocks until the device is ready or available.

• If O_NDELAY or O_NONBLOCK is set, the open() function returns without waiting for the device to be ready or available. Subsequent behavior of the device is device-specific.

Notes

AES Support Level:  Full use

Return Values

Upon successful completion, the open() and creat() functions return the file descriptor, a nonnegative integer. Otherwise, a value of -1 is returned and errno is set to indicate the error.
Errors

If the `open()` or `creat()` function fails, `errno` may be set to one of the following values:

- **ENOENT** The O_CREAT flag is not set and the named file does not exist, or O_CREAT is set and the path prefix does not exist, or the path parameter points to the empty string.
- **EACCES** Search permission is denied on a component of the path prefix, or the type of access specified by the `oflag` parameter is denied for the named file, or the file does not exist and write permission is denied for the parent directory, or O_TRUNC is specified and write permission is denied.
- **EISDIR** The named file is a directory and write access is requested.
- **EMFILE** The system limit for open file descriptors per process has already reached OPEN_MAX.
- **EFAULT** The path parameter is an invalid address.
- **ENFILE** The system file table is full.
- **ENXIO** The named file is a character special or block special file, and the device associated with this special file does not exist.
- **ENXIO** The named file is a multiplexed special file and either the channel number is outside of the valid range or no more channels are available.
- **ENXIO** The O_NONBLOCK flag is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading.
- **EEXIST** The O_CREAT and O_EXCL flags are set and the named file exists.
- **EAGAIN** The O_TRUNC flag is set, the named file exists with enforced record locking enabled, and there are record locks on the file.
- **EINTR** A signal was caught during the `open()` function.
- **EROFS** The named file resides on a read-only file system and write access is required.
- **ENOSPC** The directory that would contain the new file cannot be extended, the file does not exist, and O_CREAT is requested.
- **EDQUOT** The directory in which the entry for the new link is being placed cannot be extended because the quota of disk blocks or i-nodes defined for the user on the file system containing the directory has been exhausted.
[ENOTDIR] A component of the path prefix is not a directory.

[ELOOP] Too many links were encountered in translating path.

[ENAMETOOLONG] The length of the path string exceeds PATH_MAX, or a pathname component is longer than NAME_MAX. [ETXTBSY] The file is being executed and oflag is O_WRONLY or O_RDWR.

Related Information

Functions: chmod(2), close(2), fcntl(2), lockf(3), lseek(2), read(2), stat(2), truncate(2), umask(2), write(2)
open dir, readdir, tell dir, seek dir, rewind dir, closedir

Purpose
Performs operations on directories

Library
Standard C Library (libc.a)

Synopsis
#include <sys/types.h>
#include <sys/dirent.h>

DIR *opendir (
    const char *dir_name);

int opendir_t (
    char *dir_name,
    DIR *dir_pointer);

struct dirent *readdir (
    DIR *dir_pointer);

int readdir_r (
    DIR *dir_pointer,
    struct dirent result);

long tell dir (
    DIR *dir_pointer);

int seek dir (
    DIR *dir_pointer,
    long location);

int rewind dir (
    DIR *dir_pointer);

int closedir (
    DIR *dir_pointer);

Parameters

- dir_name Names the directory. If the final component of dir_name names a symbolic link, the link will be traversed and pathname resolution will continue.
- dir_pointer Points to the dir structure of an open directory.
location Specifies the number of an entry relative to the start of the directory.
result Contains the next directory entry on return from the readdir_r() function.

Description

The opendir() function opens the directory designated by the dir_name parameter and associates a directory stream with it. The directory stream is positioned at the first entry. The type DIR, which is defined in the dirent.h header file, represents a directory stream, which is an ordered sequence of all the directory entries in a particular directory. If a file descriptor is used, the FD_CLOEXEC flag will be set on that file descriptor.

The opendir() function also returns a pointer to identify the directory stream in subsequent operations. The null pointer is returned when the directory named by the dir_name parameter cannot be accessed or when not enough memory is available to hold the entire stream.

The type DIR, which is defined in the dirent.h header file, represents a directory stream, which is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files; files may be removed from a directory or added to a directory asynchronously to the operation of the readdir() function.

The readdir() function returns a pointer to a structure representing the directory entry at the current position in the directory stream specified by the dir_pointer parameter, and positions the directory stream at the next entry. It returns a null pointer upon reaching the end of the directory stream. The dirent structure defined in the dirent.h header file describes a directory entry.

The readdir() function will not return directory entries containing empty names. If entries for . (dot) or .. (dot-dot) exist, one entry will be returned for . (dot) and one entry will be returned for .. (dot-dot); otherwise, they will not be returned.

The pointer returned by the readdir() function points to data which may be overwritten by another call to readdir() on the same directory stream. This data will not be overwritten by another call to readdir() on a different directory stream.

If a file is removed from or added to the directory after the most recent call to the opendir() or rewinddir() function, whether a subsequent call to the readdir() function returns an entry for that file is unspecified.
The `readdir()` function may buffer several directory entries per actual read operation; the `readdir()` function marks for update the `st_atime` field of the directory each time the directory is actually read.

When it reaches the end of the directory, or when it detects an invalid `seekdir()` operation, the `readdir()` function returns the null value.

The `telldir()` function returns the current location associated with the specified directory stream.

The `seekdir()` function sets the position of the next `readdir()` operation on the directory stream specified by the `dir_pointer` parameter to the position specified by the `location` parameter.

If the value of the `location` parameter was not returned by a call to the `telldir()` function, or if there was an intervening call to the `rewinddir()` function on this directory stream, the effect is undefined. The new position reverts to the one associated with the directory stream when the `telldir()` operation was performed.

An attempt to seek to an invalid location causes the `readdir()` function to return the null value the next time it is called. The position should be that returned by a previous `telldir()` function call.

The `rewinddir()` function resets the position of the specified directory stream to the beginning of the directory. It also causes the directory stream to refer to the current state of the corresponding directory, as a call to the `opendir()` function would have done. If the `dir_pointer` parameter does not refer to a directory stream, the effect is undefined.

The `closedir()` function closes a directory stream and frees the structure associated with the `dir_pointer` parameter. Upon return, the value of `dir_pointer` may no longer point to an accessible object of the type `DIR`. If a file descriptor is used to implement type `DIR`, that file descriptor will be closed.

The `opendir_r()` and `readdir_r()` functions are the reentrant versions of the `opendir()` and `readdir()` functions, respectively. The `opendir_r()` function stores the new directory stream associated with `dir_name` at `dir_pointer`. The `readdir_r()` function stores the next directory entry at `result`.
Example

To search a directory for the entry name:

```
len = strlen(name);
dir_pointer = opendir(".");
for (dp = readdir(dir_pointer); dp != NULL; dp =
    readdir(dir_pointer))
    if (dp->d_namlen == len && !strcmp(dp->d_name, name)) {
        closedir(dir_pointer);
        return FOUND;
    }
```

Notes

An open directory must always be closed with the closedir() function to ensure that the next attempt to open that directory is successful.

The use of the seekdir() and telldir() functions is not recommended in OSF/1, as the results can be unpredictable.

AES Support Level: Full use (opendir(), closedir(), readdir(), rewinddir())
Trial use (seekdir(), telldir())

Return Values

Upon successful completion, the opendir() function returns a pointer to an object of type DIR. Otherwise, null is returned and errno set to indicate the error.

Upon successful completion, the readdir() function returns a pointer to an object of type struct dirent. When an error is encountered, a null pointer is returned and errno is set to indicate the error. When the end of the directory is encountered, a null pointer is returned and errno is not changed.

Upon successful completion, the telldir() function returns the current location. Otherwise, -1 is returned.

Upon successful completion, the seekdir() function returns 0 (zero). Otherwise, -1 is returned.

Upon successful completion, the rewinddir() function returns 0 (zero). Otherwise, -1 is returned.

Upon successful completion, the closedir() function returns 0 (zero). Otherwise, -1 is returned.
Upon successful completion, the `opendir_r()` function returns 0 (zero). Otherwise, -1 is returned.

Upon successful completion, the `readdir_r()` function returns 0 (zero). Otherwise, -1 is returned.

**Errors**

If the `opendir()` function fails, `errno` may be set to one of the following values:

- **[EACCES]** Search permission is denied for any component of `dir_name` or read permission is denied for `dir_name`.
- **[ELOOP]** Too many links were encountered in translating `dir_name`.
- **[ENAMETOOLONG]** The length of the `dir_name` string exceeds PATH_MAX, or a pathname component is longer than NAME_MAX.
- **[ENOENT]** The `dir_name` parameter points to the name of a file which does not exist, or the parameter points to an empty string.
- **[ENOTDIR]** A component of `dir_name` is not a directory.

**Related Information**

Functions: `close(2)`, `lseek(2)`, `open(2)`, `read(2)`, `scandir(3)`
pathconf, fpathconf

Purpose
Retrieves file implementation characteristics

Library
Standard C Library (libc.a)

Synopsis
#include <unistd.h>

long pathconf(
    const char *path,
    int name);

long fpathconf(
    int filedes,
    int name);

Parameters
path Specifies the pathname. If the final component of path is a symbolic link, it will be traversed and filename resolution will continue.

filedes Specifies an open file descriptor.

name Specifies the configuration attribute to be queried. If this attribute is not applicable to the file specified by the path or filedes parameter, the pathconf() function returns an error.

Description
The pathconf() function allows an application to determine the characteristics of operations supported by the file system underlying the file named by the path parameter. Read, write, or execute permission of the named file is not required, but all directories in the path leading to the file must be searchable.

The fpathconf() function allows an application to retrieve the same information for an open file.
Symbolic values for the name parameter are defined in the unistd.h header file, as follows:

_PC_LINK_MAX
   The maximum number of links to the file. If the path or filedes parameter refers to a directory, the value returned applies to the directory itself.

_PC_MAX_CANON
   The maximum number of bytes in a canonical input line. This is applicable only to terminal devices.

_PC_MAX_INPUT
   The number of types allowed in an input queue. This is applicable only to terminal devices.

_PC_NAME_MAX
   Maximum number of bytes in a filename (not including a terminating null). This may be as small as 13, but is never larger than 255. This is applicable only to a directory file. The value returned applies to filenames within the directory.

_PC_PATH_MAX
   Maximum number of bytes in a pathname (not including a terminating null). This is never larger than 65,535. This is applicable only to a directory file. The value returned is the maximum length of a relative pathname when the specified directory is the working directory.

_PC_PIPE_BUF
   Maximum number of bytes guaranteed to be written atomically. This is applicable only to a FIFO. The value returned applies to the referenced object. If the path or filedes parameter refers to a directory, the value returned applies to any FIFO that exists or can be created within the directory.

_PC_CHOWN_RESTRICTED
   This is applicable only to a directory file. The value returned applies to any files (other than directories) that exist or can be created within the directory.
pathconf(3)

_PC_NO_TRUNC
Returns 1 if supplying a component name longer than allowed by NAME_MAX will cause an error. Returns 0 (zero) if long component names are truncated. This is applicable only to a directory file.

_PC_VDISABLE
This is always 0 (zero); no disabling character is defined. This is applicable only to a terminal device.

Notes

AES Support Level: Full use

Return Values

Upon successful completion, the pathconf() or fpathconf() function returns the specified parameter. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the pathconf() function fails, errno may be set to the following value:

[EACCES] Search permission is denied for a component of the path prefix.
[ELOOP] Too many links were encountered in translating a pathname.
[EINVAL] The name parameter specifies an unknown or inapplicable characteristic.
[EFAULT] The path argument is an invalid address.
[ENAMETOOLONG] The length of the path string exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
[ENOENT] The named file does not exist or the path argument points to an empty string.
[ENOTDIR] A component of the path prefix is not a directory.
If the `fpathconf()` function fails, `errno` may be set to the following value:

[ELOOP] Too many links were encountered in translating a pathname.

[EINVAL] The name parameter specifies an unknown or inapplicable characteristic.

[EBADF] The `fildes` argument is not a valid file descriptor.
pause

Purpose  Suspending a process until a signal is received

Library  Standard C Library (libc.a)

Synopsis  int pause( void );

Description  The `pause()` function suspends the calling process until it receives a signal whose action is either to execute a signal-catching function or terminate the process. The signal must not be one that is not acknowledged by the calling process. The `pause()` function does not affect the action taken when a signal is received.

The `pause()` function, which suspends the calling process until the request is completed, is redefined so that only the calling thread is suspended.

Notes  The `pause()` function is not supported for multi-threaded applications.

AES Support Level:  Full use

Return Values  When the received signal causes the calling process to end, the `pause()` function does not return.

When the signal is caught by the calling process and control is returned from the signal-catching function, the calling process resumes execution from the point of suspension, and the `pause()` function returns a value of -1 and sets `errno` to the value `[EINTR]`.

Errors  If the `pause()` function fails, `errno` may be set to the following value:

[EINTR]  The signal is caught by the calling process and control is returned from the signal-catching function.
Related Information

Functions: `alarm(3)`, `kill(2)`, `sigaction(2)`, `sigvec(2)`, `wait(2)`
pclose

Purpose
Closes a pipe to a process

Library
Standard I/O Package (libc.a)

Synopsis
#include <stdio.h>

int pclose (FILE *stream);

Parameters
stream	Points to a FILE structure for an open pipe returned by a previous call to the popen( ) function.

Description
The pclose( ) function closes a pipe between the calling program and a shell command to be executed. Use the pclose( ) function to close any stream you have opened with the popen( ) function. The pclose( ) function waits for the associated process to end, and then returns the exit status of the command.

Notes
AES Support Level: Trial use

Caution
If the original processes and the process started with the popen( ) function concurrently read or write a common file, neither should use buffered I/O. If they do, the results are unpredictable.

Return Values
Upon successful completion, the pclose( ) function returns the exit status of the command. If the stream parameter is not associated with a popen( ) command, a value of -1 is returned.
Errors

If the `pclose()` function fails, `errno` may be set to the following value:

ECHILD      The status of the child process could not be obtained.

Related Information

Functions: `fclose(3)`, `popen(3)`, `wait(2)`
perror

Purpose  Writes a message explaining a function error

Library   Standard C Library (libc.a)

Synopsis  #include <errno.h>
          
          void perror (
          
          char *string);

          extern char *sys_errlist[];
          extern int sys_nerr;

Parameters

string     A parameter string that contains the name of the program that caused the error. The ensuing printed message contains this string, a colon, and an explanation of the error.

Description

The perror( ) function writes a message on the standard error output that describes the last error encountered by a function or library function. The error message includes the string parameter string followed by a : (colon), a blank, the message, and a newline character. The string parameter string should include the name of the program that caused the error. The error number is taken from errno, which is set when an error occurs, but is not cleared when a successful call is made.

Use errno as an index into this table to get the message string without the newline character. The largest message number provided in the table is sys_nerr. Be sure to check sys_nerr because new error codes can be added to the system before they are added to the table.

Notes

AES Support Level:  Full use

Related Information

Functions: printf(3)
pipe

Purpose  Creates an interprocess channel

Synopsis  int pipe (  
                      int *filedes[2] );

Parameters

    filedes  Specifies the address of an array of two integers into which the new
             file descriptors are placed.

Description

The pipe( ) function creates a unidirectional interprocess channel called a pipe, and
returns two file descriptors, filedes[0] and filedes[1]. The file descriptor specified
by the filedes[0] parameter is opened for reading and the file descriptor specified
by the filedes[1] parameter is opened for writing. Their integer values will be the
two lowest available at the time of the call to the pipe( ) function. The
O_NONBLOCK flag will be clear on both file descriptors. (The fcntl( ) function
can be used to set the O_NONBLOCK flag.)

A process has the pipe open for reading if it has a file descriptor open that refers to
the read end, filedes[0]. A process has the pipe open for writing if it has a file
descriptor open that refers to the write end, filedes[1]. A read on file descriptor
filedes[0] accesses the data written to filedes[1] on a first-in, first-out (FIFO) basis.

Upon successful completion, the pipe( ) function marks the st_atime, st_ctime and
st_mtime fields of the pipe for update.

The FD_CLOEXEC flag will be clear on both file descriptors.

Notes

 AES Support Level:  Full use

Return Values

Upon successful completion, a value of 0 (zero) is returned. If the pipe( ) function
fails, a value of -1 is returned and errno is set to indicate the error.
Errors

If the pipe() function fails, errno may be set to one of the following values:

- [EFAULT] The filedes parameter is an invalid address.
- [EMFILE] More than OPEN_MAX-2 file descriptors are already open by this process.
- [ENFILE] The system file table is full, or the device containing pipes has no free i-nodes.

Related Information

Functions: read(2), fcntl(2), select(2), write(2)

Commands: sh(1)
plock

Purpose  Locks a process' text and/or data segments in memory

Synopsis  #include <sys/lock.h>

    int plock(
        int opr);

Parameters

    opr  Specifies one of the following operations:

        PROCLOCK  
            Locks the text and data segments into memory.

        TXTLOCK  
            Locks the text segment into memory.

        DATLOCK  
            Locks the data segment into memory.

        UNLOCK  
            Removes locks.

Description  The plock() function locks or unlocks a process' text segments, data segments, or both in physical memory. When locked, the physical pages containing the text or data segment will not be paged out. It is an error to lock a segment that is already locked.

    The caller must have superuser privilege to use the plock() function.

    Note that memory acquired subsequent to a plock() function may or may not be locked in memory, depending on the specific acquisition method. Memory acquired using the brk() function (or the sbrk() function) will be locked if the data segment was locked. Memory acquired via the mmap() or vm_allocate() functions will not be locked.
Return Values

Upon successful completion, a value of 0 (zero) is returned to the calling process. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `plock()` function fails, `errno` may be set to one of the following values:

- `[EPERM]` The caller does not have appropriate privilege.
- `[EINVAL]` The opr parameter is PROCLOCK, but the text segment or the data segment is already locked.
- `[EINVAL]` The opr parameter is TXTLOCK, but the text segment is already locked.
- `[EINVAL]` The opr parameter is DATLOCK, but the data segment is already locked.
- `[EINVAL]` The opr parameter is UNLOCK, but neither the text segment nor the data segment is locked.

Related Information

Functions: `brk(2), mmap(2)`
poll

Purpose  Monitors conditions on multiple file descriptors

Synopsis  
#include <sys/poll.h>
int poll(
    struct pollfd filedes[ ],
    unsigned int nfds,
    int timeout);

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filedes</td>
<td>Points to an array of pollfd structures, one for each file descriptor of interest.</td>
</tr>
<tr>
<td>nfds</td>
<td>Specifies the number of pollfd structures in the filedes array.</td>
</tr>
<tr>
<td>timeout</td>
<td>Specifies the maximum length of time (in milliseconds) to wait for at least one of the specified events to occur.</td>
</tr>
</tbody>
</table>

Description

The poll() function provides a general mechanism for reporting I/O conditions associated with a set of file descriptors and for waiting until one or more specified conditions becomes true. Specified conditions include the ability to read or write data without blocking, and error conditions.

Each pollfd structure includes the following members:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int fd</td>
<td>The file descriptor</td>
</tr>
<tr>
<td>short events</td>
<td>The requested conditions</td>
</tr>
<tr>
<td>short revents</td>
<td>The reported conditions</td>
</tr>
</tbody>
</table>

The fd member of each pollfd structure specifies an open file descriptor. The poll() function uses the events member to determine what conditions to report for this file descriptor. If one or more of these conditions is true, the poll() function sets the associated revents member.

The poll() function ignores any pollfd structure whose fd member is less than 0 (zero). If the fd member of all pollfd structures is less than 0, the poll() function will return 0 and have no other results.
The `events` and `revents` members of the `pollfd` structure are bitmasks. The calling process sets the `events` bitmask, and `poll()` sets the `revents` bitmasks. These bitmasks contain ORed combinations of condition flags. The following condition flags are defined:

- **POLLNORM** Data may be read without blocking.
- **POLLOUT** Data may be written without blocking.
- **POLLERR** An error has occurred on the file descriptor.
- **POLLHUP** The device has been disconnected.
- **POLLNVAL** The value specified for `fd` is invalid.

The conditions indicated by `POLLNORM` and `POLLOUT` are true if and only if at least one byte of data can be read or written without blocking. The exception is regular files, which always poll true for `POLLNORM` and `POLLOUT`.

The condition flags `POLLERR`, `POLLHUP`, and `POLLNVAL` are always set in `revents` if the conditions they indicate are true for the specified file descriptor, whether or not these flags are set in `events`.

For each call to the `poll()` function, the set of reportable conditions for each file descriptor consists of those conditions that are always reported, together with any further conditions for which flags are set in `events`. If any reportable condition is true for any file descriptor, the `poll()` function will return with flags set in `revents` for each true condition for that file descriptor.

If no reportable condition is true for any of the file descriptors, the `poll()` function waits up to `timeout` milliseconds for a reportable condition to become true. If, in that time interval, a reportable condition becomes true for any of the file descriptors, `poll()` reports the condition in the file descriptor’s associated `revents` member and returns. If no reportable condition becomes true, `poll()` returns without setting any `revents` bitmasks.

If the `timeout` parameter is a value of -1, the `poll()` function does not return until at least one specified event has occurred. If the value of the `timeout` parameter is 0 (zero), the `poll()` function does not wait for an event to occur but returns immediately, even if no specified event has occurred. The behavior of the `poll()` function is not affected by whether the O_NONBLOCK flag is set on any of the specified file descriptors.

**Notes**

For compatibility with BSD systems, the `select()` function is also supported.

**AES Support Level:** Trial use
Return Values
Upon successful completion, the poll() function returns a nonnegative value. If the call returns 0 (zero), poll() has timed out and has not set any of the revents bitmasks. A positive value indicates the number of file descriptors for which poll() has set the revents bitmask. If the poll() function fails, -1 is returned and errno is set to indicate the error.

Errors
If the poll() function fails, errno may be set to one of the following values:

[EAGAIN] Allocation of internal data structures failed. A later call to the poll() function may complete successfully.

[EINTR] A signal was caught during the poll() function and the signal handler was installed with an indication that functions are not to be restarted.

[EINVAL] The timeout parameter is a negative number other than -1.

[EFAULT] The filedes parameter in conjunction with the nfds parameter addresses a location outside of the allocated address space of the process.

Related Information
Functions: read(2), write(2)
popen

Purpose
Initiates a pipe to a process

Library
Standard I/O Package (libc.a)

Synopsis
```
#include <stdio.h>

FILE *popen (const char *command, const char *type);
```

Parameters
- `command`: Points to a null-terminated string containing a shell command line.
- `type`: Points to a null-terminated string containing an I/O mode. If the type parameter is the value `r`, the calling program can read from the standard output of the command by reading from the returned file stream. If the type parameter is the value `w`, the calling program can write to the standard input of the command by writing to the returned file stream.

Description
The `popen()` function creates a pipe between the calling program and a shell command to be executed. It returns a pointer to a `FILE` structure for the stream.

Because open files are shared, a type `r` command can be used as an input filter and a type `w` command as an output filter.
Notes

Programs using the `popen()` function to invoke an output filter should beware of possible deadlock caused by output data remaining in the program’s buffer. This can be avoided by either using the `setbuf()` function to ensure that the output stream is unbuffered, or by using the `fflush()` function to ensure that all buffered data is flushed before calling the `pclose()` function.

AES Support Level: Trial use

Caution

If the original processes and the process started with the `popen()` function concurrently read or write a common file, neither should use buffered I/O. If they do, the results are unpredictable.

Return Values

Upon successful completion, the `popen()` function returns a pointer to the FILE structure for the opened stream. In case of error because files or processes could not be created, the `popen()` function returns a null pointer.

Related Information

Functions: `exec(2)`, `fork(2)`, `fclose(3)`, `fopen(3)`, `pclose(3)`, `pipe(2)`, `setbuf(3)`
printf, fprintf, sprintf

Purpose
Prints formatted output

Library
Standard I/O Package (libc.a)

Synopsis
#include <stdio.h>

int printf (const char *format [, value, ...]);
intp fprintf (FILE *stream, const char *format [, value, ...]);
inint sprintf (char *string, const char *format [, value, ...]);

Parameters
format Specifies a character string combining literal characters with conversion specifications.
value Specifies the data to be converted according to the format parameter.
stream Points to a FILE structure specifying an open stream to which converted values will be written.
string Points to a character array in which the converted values will be stored.

Description
The printf() function converts, formats, and writes its value parameters, under control of the format parameter, to the standard output stream stdout.
The fprintf() function converts, formats, and writes its value parameters, under control of the format parameter, to the output stream specified by its stream parameter.
The `sprintf()` function converts, formats, and stores its `value` parameters, under control of the `format` parameter, into consecutive bytes starting at the address specified by the `string` parameter. The `sprintf()` function places a '\0' (null character) at the end. You must ensure that enough storage space is available to contain the formatted string.

The `format` parameter is a character string that contains two types of objects:

- Literal characters, which are copied to the output stream.
- Conversion specifications, each of which causes zero or more items to be fetched from the `value` parameter list.

If there are not enough items for `format` in the `value` parameter list, the results are unpredictable. If more `values` remain after the entire `format` has been processed, they are ignored.

Conversion Specifications

Each conversion specification in the `format` parameter has the following syntax:

- A `%` (percent) sign.
- Zero or more `options`, which modify the meaning of the conversion specification. The `option` characters and their meanings are:
  - `-` Left align within the field the result of the conversion.
  - `+` Begin the result of a signed conversion with a sign (+ or -).
  - `(space)` Prefix a space character to the result if the first character of a signed conversion is not a sign. If both the (space) and + options appear, the (space) option is ignored.
  - `#` Convert the value to an alternate form. For c, d, i, s, and u conversions, the option has no effect. For o conversion, it increases the precision to force the first digit of the result to be a 0 (zero). For x and X conversions, a nonzero result has 0x or 0X prefixed to it. For e, E, f, g, and G conversions, the result always contains a decimal point, even if no digits follow it. For g and G conversions, trailing zeros are not removed from the result.
  - `B` Give field width and precision in bytes, rather than in code points, for conversions using the s or S conversion characters.
  - `N` Convert each international character support code point in the converted string into a printable ASCII escape sequence that uniquely identifies the code point. This option affects the s and S conversion characters.
Pad to field width using leading zeros (following any indication of sign or base) for \texttt{d, i, o, u, x, X, e, E, f, g, and G} conversions; no space padding is performed. If the \texttt{0} and - (dash) flags both appear, the \texttt{0} flag will be ignored. For \texttt{d, i, o u, x, and X} conversions, if a precision is specified, the \texttt{0} flag is also ignored. For other conversions, the behavior is undefined.

For Japanese language support. This option can be used with all conversion characters that take an \texttt{int, long, double, or float} value as an argument. The \texttt{J} flag, appearing with any of these numeric conversions, indicates that output such as characters, digits, signs, or padding blanks will be 2-byte codes and two columns wide. The \texttt{J} flag can also be used with the \texttt{%c, %s, and %S} conversion characters to indicate that padding should use double-width spaces.

- An optional decimal digit string that specifies the minimum field width. If the converted value has fewer characters than the field width, the field is padded on the left to the length specified by the field width. If the left-adjustment option is specified, the field is padded on the right.

- An optional precision. The precision is a . (dot) followed by a decimal digit string. If no precision is given, it is treated as 0 (zero). The precision specifies:
  - The minimum number of digits to appear for the \texttt{d, u, o, x, or X} conversions.
  - The number of digits to appear after the decimal point for the \texttt{e, E, and f} conversions.
  - The maximum number of significant digits for the \texttt{g and G} conversions.
  - The maximum number of characters to be printed from a string in the \texttt{s} conversion.

- An optional \texttt{h, l, or L} specifying that a following \texttt{d, i, u, o, x, or X} conversion character applies to, respectively, a \texttt{long} integer value, a \texttt{short} integer value, or a \texttt{double} integer value. The \texttt{h} and \texttt{l} options can also be used with the \texttt{n} conversion specifier to indicate a pointer to a \texttt{short int} or \texttt{long int} argument, respectively.

- A character that indicates the type of conversion to be applied:
  \texttt{\%} Performs no conversion. Prints \%.
  \texttt{d, i} Accepts an integer value and converts it to signed decimal notation. The precision specifies the minimum number of digits to appear. If the
value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as a leading character causes the field width value to be padded with leading zeros.

u  Accepts an integer value and converts it to unsigned decimal notation. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as a leading character causes the field width value to be padded with leading zeros.

o  Accepts an integer value and converts it to unsigned octal notation. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as a leading character causes the field width value to be padded with leading zeros. An octal value for field width is not implied.

x, X  Accepts an integer value and converts it to unsigned hexadecimal notation. The letters abcdef are used for the x conversion and the letters ABCDEF are used for the X conversion. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as a leading character causes the field width value to be padded with leading zeros.

f  Accepts a float or double value and converts it to decimal notation in the format [-]ddd.ddd. The number of digits after the decimal point is equal to the precision specification. If no precision is specified, six digits are output. If the precision is zero, no decimal point appears. If a decimal point is output, at least one digit is output before it. The value is rounded to the appropriate number of digits.

e, E  Accepts a float or double value and converts it to the exponential form [-]d.ddde+/-dd. There is one digit before the decimal point and the number of digits after the decimal point is equal to the precision specification. If no precision is specified, six digits are output. If the
precision is zero, no decimal point appears. The E conversion character produces a number with E instead of e before the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.

g, G    Accepts a float or double value and converts it in the style of the e, E, or f conversion characters, with the precision specifying the number of significant digits. Trailing zeros are removed from the result. A decimal point appears only if it is followed by a digit. The style used depends on the value converted. Style e (E, if G is the flag used) results only if the exponent resulting from the conversion is less than -4, or if it is greater or equal to the precision.

c    Accepts and prints an integer value converted to an unsigned char.

s    Accepts a value as a string (character pointer), and characters from the string are printed until a ' ' (null character) is encountered or the number of characters indicated by the precision is reached. If no precision is specified, all characters up to the first null character are printed. If the string pointer value has a value of 0 (zero) or null, the results are undefined.

p    Accepts a pointer to void. The value of the pointer is converted to a sequence of printable characters, the same as unsigned hexadecimal (x).

n    Accepts a pointer to an integer into which is written the number of characters written to the output stream so far by this call. No argument is converted.

A field width or precision can be indicated by an * (asterisk) instead of a digit string. In this case, an integer value parameter supplies the field width or precision. The value parameter converted for output is not fetched until the conversion letter is reached, so the parameters specifying field width or precision must appear before the value (if any) to be converted.

If the result of a conversion is wider than the field width, the field is expanded to contain the converted result. No truncation occurs. However, a small precision can cause truncation on the right.

The e, E, f, and g formats represent the special floating-point values as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quiet NaN</td>
<td>+NaNQ or -NaNQ</td>
</tr>
<tr>
<td>Signaling NaN</td>
<td>+NaNs or -NaNs</td>
</tr>
<tr>
<td>+/-INF</td>
<td>+INF or -INF</td>
</tr>
<tr>
<td>+/-0</td>
<td>+0 or -0</td>
</tr>
</tbody>
</table>
The representation of the plus sign depends on whether the + or (space) formatting option is specified.

The printf() functions can handle a format string that enables the system to process elements of the argument list in variable order. In such a case, the normal conversion character % (percent sign) is replaced by "%digit$", where digit is a decimal number in the range [1, NL_ARGMAX]. Conversion is then applied to the argument, rather than to the next unused argument. This feature provides for the definition of format strings in an order appropriate to specific languages. When variable ordering is used, the * (asterisk) specification for field width in precision is replaced by "%digit$". If the variable ordering feature is used, it must be specified for all conversions.

The * (asterisk) specification for field width or precision is not permitted with the variable order %digit$ format.

All forms of the printf() functions allow for the insertion of a language-dependent radix character in the output string. The radix character is defined by langinfo data in the program’s locale (category LC_NUMERIC). In the “C” locale, or in a locale where the radix character is not defined, the radix character defaults to . (decimal point).

The st_ctime and st_mtime fields of the file are marked for update between the successful execution of the printf() or fprintf() functions and the next successful completion of a call to the fflush() or fclose() functions on the same stream, or a call to the exit() or abort() functions.

AES Support Level: Full use

Return Values

Upon successful completion, each of these functions returns the number of display characters in the output string rather than the number of bytes in the string. Otherwise, a negative value is returned.

The value returned by the sprintf() function does not include the final '\0' character.
Errors

The `printf()` or `fprintf()` functions fail if either the `stream` is unbuffered, or the `stream`'s buffer needed to be flushed and the function call caused an underlying `write()` or `lseek()` function to be invoked. In addition, if the `printf()` or `fprintf()` function fails, `errno` may be set to one of the following values:

- **[EAGAIN]** The O_NONBLOCK flag is set for the file descriptor underlying `stream` and the process would be delayed in the write operation.
- **[EBADF]** The file descriptor underlying `stream` is not a valid file descriptor open for writing.
- **[EFBIG]** An attempt was made to write to a file that exceeds the process’ file size limit or the maximum file size.
- **[EINTR]** The read operation was interrupted by a signal which was caught, and no data was transferred.
- **[EIO]** The implementation supports job control, the process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
- **[ENOSPC]** There was no free space remaining on the device containing the file.
- **[EPIPE]** An attempt was made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

Related Information

Functions: `conv(3)`, `ecvt(3)`, `putc(3)`, `scanf(3)`, `wsprintf(3)`
profil

Purpose  Starts and stops execution profiling

Synopsis  void profil(
           short *short_buffer,
           unsigned int buffer_size,
           unsigned int offset,
           unsigned int scale);

Parameters

  short_buffer  Points to an area of memory in the user address space. Its length (in bytes) is given by the buffer_size parameter.
  buffer_size  Specifies the length (in bytes) of the buffer.
  offset  Specifies the delta of program counter start and buffer; for example, an offset of 0 (zero) implies that text begins at 0.
  scale  Specifies the mapping factor between the program counter and short_buffer.

Description

The profil() function controls execution profiling.

The short_buffer parameter points to an area of memory whose length (in bytes) is given by the buffer_size parameter. After this call, the process’ program counters are examined at regular intervals (10 ms. in most implementations). The value of the offset parameter is subtracted from the program counter, and the result multiplied by the scale parameter. The corresponding location in the short_buffer parameter is incremented if the resulting number is less than the buffer_size parameter.

The scale parameter is interpreted as an unsigned, fixed point fraction with 16 bits of mantissa: 0x10000 gives a 1-1 mapping of program counter values to words in the short_buffer parameter; 0x8000 maps each pair of program counter values together.

Profiling is turned off by giving a scale parameter of 1. Profiling is turned off when an execve() is executed. Profiling remains on in both the parent and child processes after a fork. Profiling is turned off if an update in the short_buffer parameter would cause a memory fault.
If the process contains multiple kernel threads, each will be independently sampled and the counts will reflect the sum of the samples for all of the threads.

Related Information

Functions: `exec(2), fork(2)`
Commands: `prof(1), gprof(1)`
pthread_attr_create

Purpose
Creates a thread attributes object

Library
Threads Library (libpthreads.a)

Synopsis
#include <pthread.h>

int pthread_attr_create(
    pthread_attr_t *attr);

Parameters
attr Specifies the address in which the ID for the new thread attributes object will be stored.

Description
The pthread_attr_create( ) function creates a thread attributes object specified by the attr parameter, initialized with the default values for the thread attributes. When you create a new thread (using the pthread_create( ) function), you use an attributes object to specify the attributes to be used for that thread.

The only thread attribute that is currently modifiable is stack size. Use the pthread_attr_setstacksize( ) function to change the value of the stack size attribute. The default stack size is the maximum stack size allowed on your system.

You can apply the same attributes object to more than one thread.

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values
Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.
Errors

If the `pthread_attr_create()` function fails, `errno` may be set to one of the following values:

- [ENOMEM] There is not enough memory to create the thread attributes object. This is not a temporary condition.
- [EINVAL] The value specified by the `attr` parameter is invalid.

Related Information

Functions: `pthread_create(3), pthread_attr_delete(3)`
pthread_attr_delete

Purpose
Deletes a thread attributes object

Library
Threads Library (libpthread.a)

Synopsis
#include <pthread.h>
int pthread_attr_delete(
    pthread_attr_t *attr);

Parameters
attr Specifies the address of the thread attributes object to be deleted.

Description
The pthread_attr_delete() function deletes a thread attributes object, which
allows the resources for the attr parameter to be reclaimed.

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be
changed to conform to the final version.

Return Values
Upon successful completion, the attr parameter is set to an illegal value, and a
value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate
the error.

Errors
If the pthread_attr_delete() function fails, errno may be set to the following
value:
[EINVAL] The value specified by the attr parameter is invalid.

Related Information
Functions: pthread_create(3), pthread_attr_create(3)
**pthread_attr_getstacksize**

**Purpose**
Returns the value of the stack size attribute of a thread attributes object.

**Library**
Threads Library (libpthreads.a)

**Synopsis**
```
#include <pthread.h>
int pthread_attr_getstacksize(
    pthread_attr_t *attr);
```

**Parameters**
- *attr* Specifies the address of the thread attributes object to be examined.

**Description**
The `pthread_attr_getstacksize()` function returns the value of the stack size attribute of the specified attributes object in bytes.

**Notes**
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

**Return Values**
Upon successful completion, the stack size is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

**Errors**
If the `pthread_attr_getstacksize()` function fails, `errno` may be set to the following value:

- [EINVAL] The value specified by the `attr` parameter is invalid.
Functions

pthread_attr_getstacksize(3)

Related Information

Functions: pthread_attr_create(3), pthread_attr_setstacksize(3)
pthread_attr_setstacksize

**Purpose**
Sets the value of the stack size attribute of a thread attributes object

**Library**
Threads Library (libpthreads.a)

**Synopsis**
#include <pthread.h>
int pthread_attr_setstacksize(
    pthread_attr_t *attr,
    long stacksize);

**Parameters**
- *attr* Specifies the address of the thread attributes object to be modified.
- *stacksize* Specifies the new value for the stack size attribute (in bytes).

**Description**
The pthread_attr_setstacksize() function sets the thread stack size attribute. The stack size attribute specifies the minimum number of bytes allocated to the thread when it is created. The default stack size is the maximum stack size allowed on your system.

**Notes**
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

**Return Values**
Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.
Errors

If the `pthread_attr_setstacksize()` function fails, `errno` may be set to the following value:

[EINVAL] The value specified by the `attr` or `stacksize` parameter is invalid.

Related Information

Functions: `pthread_attr_create(3)`, `pthread_attr_getstacksize(3)`
pthread_cancel

**Purpose**

Initiates termination of a thread

**Library**

Threads Library (*libpthreads.a*)

**Synopsis**

```c
#include <pthread.h>

int pthread_cancel(
    pthread_t thread);
```

**Parameters**

- `thread` Specifies the ID of the thread to be canceled.

**Description**

The `pthread_cancel()` function initiates termination processing of the specified thread. If the target thread has already been canceled, the termination request is ignored.

If the general cancelability of the target thread has been disabled, the termination of the thread is held pending until general cancelability is reenabled. If general cancelability is enabled and asynchronous cancelability is enabled, the termination of the target thread begins immediately. If general cancelability is enabled and asynchronous cancelability is disabled, termination is held pending until the next cancellation point.

During termination processing, any outstanding cleanup routines are executed in the context of the target thread and a status of `((void *)-1)` is made available to any threads joining with the target thread.

**Notes**

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

**Return Values**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.
Errors

If the `pthread_cancel()` function fails, `errno` may be set to one of the following values:

- [EINVAL] The value specified by `thread` is invalid.
- [ESRCH] The value specified by `thread` does not refer to an existing thread.

Related Information

Functions: `pthread_exit(3)`, `pthread_setasynccancel(3)`, `pthread_setcancel(3)`, `pthread_join(3)`
pthread_cleanup_pop

Purpose       Removes a routine from the top of the cleanup stack of the calling thread and optionally executes it

Library       Threads Library (libpthreads.a)

Synopsis      #include <pthread.h>
              void pthread_cleanup_pop(
                                  int execute);

Parameters    

execute       Specifies whether or not to execute the cleanup routine.

Description   

The pthread_cleanup_pop() function removes the routine at the top of a thread's cleanup stack. If the execute parameter is nonzero, pthread_cleanup_pop() also executes the routine. If execute is 0 (zero), the routine is not executed.

Every call to the pthread_cleanup_push() function must be matched by exactly one call to the pthread_cleanup_pop() function at the same lexical level as the push.

The effect of calling longjmp() or executing a return or goto after a call to the pthread_cleanup_push() function but before the matching call to the pthread_cleanup_pop() function is unspecified. The effect of calling longjmp() from a cleanup routine is also unspecified.

Notes         

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.
Return Values

No value is returned.

Related Information

Functions: `pthread_cleanup_push(3), pthread_cancel(3), pthread_setcancel(3)`
pthread_cleanup_push

Purpose
Pushes a routine onto the cleanup stack of the calling thread

Library
Threads Library (libpthreads.a)

Synopsis
```
#include <pthread.h>
void pthread_cleanup_push(
    void (*routine)(void *arg),
    void *arg);
```

Parameters
- **routine**: Specifies the routine to push on the calling thread’s cleanup stack.
- **arg**: Specifies the single parameter to be passed to the cleanup routine.

Description
The `pthread_cleanup_push()` function pushes the specified routine onto the calling thread’s cleanup stack.

Each thread maintains a list of cleanup routines. The `pthread_cleanup_push()` function is used to place routines on the list, and the `pthread_cleanup_pop()` function is used to remove routines from the list.

A cleanup routine will be popped from the stack and executed with the `arg` parameter when one of the following occurs:

- The thread exits.
- The thread is canceled.
- The thread calls the `pthread_cleanup_pop()` function with a nonzero `execute` parameter.

Every call to the `pthread_cleanup_push()` function must be matched by exactly one call to the `pthread_cleanup_pop()` function at the same lexical level as the push.
The effect of calling the `longjmp()` parameter or executing a `return` or `goto` after a call to the `pthread_cleanup_push()` function but before the matching call to the `pthread_cleanup_pop()` function is unspecified. The effect of calling the `longjmp()` parameter from a cleanup routine is also unspecified.

**Notes**

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

**Return Values**

No value is returned.

**Related Information**

Functions: `pthread_cancel(3)`, `pthread_setcancel(3)`
# pthread_cond_broadcast

**Purpose**  Wakes up all threads that are waiting on a condition variable

**Library**  Threads Library (libpthreads.a)

**Synopsis**  
```
#include <pthread.h>
int pthread_cond_broadcast(
    pthread_cond_t cond);
```

**Parameters**

- `cond`  Specifies the condition variable being waited on.

**Description**

The `pthread_cond_broadcast()` function wakes up all of the threads that are waiting for the specified condition to be satisfied.

The call has no effect if no threads are waiting on the condition.

**Notes**

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

**Return Values**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `pthread_cond_broadcast()` function fails, `errno` may be set to the following value:

- [EINVAL]  The value specified by the `cond` parameter is invalid.
Functions

pthread_cond_broadcast(3)

Related Information

Functions: pthread_cond_wait(3), pthread_cond_timedwait(3)
pthread_cond_destroy

Purpose
Destroys a condition variable

Library
Threads Library (libpthreads.a)

Synopsis
#include <pthread.h>
int pthread_cond_destroy(
    pthread_cond_t *cond);

Parameters
cond Specifies the address of the ID of the condition variable to be deleted.

Description
The pthread_cond_destroy() function deletes the specified condition variable, which allows the resources for the cond parameter to be reclaimed.

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values
Upon successful completion, the cond parameter is set to an illegal value, and a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.
Errors

If the `pthread_cond_destroy( )` function fails, `errno` may be set to one of the following values:

- [EBUSY] A thread is currently executing a `pthread_cond_wait( )` or `pthread_cond_timedwait( )` function on the specified condition variable.
- [EINVAL] The value specified by the `cond` parameter is invalid.

Related Information

Functions: `pthread_cond_init(3)`, `pthread_cond_signal(3)`
`pthread_cond_broadcast(3)`, `pthread_cond_wait(3)`
`pthread_cond_timedwait(3)`
pthread_cond_init

Purpose
Creates a condition variable

Library
Threads Library (libpthreads.a)

Synopsis
#include <pthread.h>
int pthread_cond_init(
    pthread_cond_t *cond,
    pthread_condattr_t attr);

Parameters
cond Specifies the address in which the ID for the new condition variable will be stored.
attr Specifies the attributes object to use in creating the new condition variable.

Description
The pthread_cond_init() function creates a condition variable with attributes specified by the attr parameter. If the attr parameter is pthread_condattr_default, the default attributes are used.

To have a thread block until some condition is true, use a condition variable with a mutex. Use the pthread_cond_wait() or pthread_cond_timedwait() function to cause the calling thread to wait until the condition is satisfied, and use the pthread_cond_signal() or pthread_cond_broadcast() function to indicate that the condition has been satisfied and to wake up the waiting thread(s).

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values
Upon successful completion, the ID of the new condition variable is stored at *cond, and a value of 0 (zero) is returned. Otherwise, no condition variable is created, -1 is returned, and errno is set to indicate the error.
Errors

If the `pthread_cond_init()` function fails, `errno` may be set to one of the following values:

- **[EAGAIN]**: The system lacks the resources necessary for creating another condition variable.
- **[EAGAIN]**: The new condition variable cannot be created without exceeding the system-imposed limit on the total number of condition variables allowed for each user.
- **[ENOMEM]**: There is not enough memory to create the condition variable. This is not a temporary condition.
- **[EINVAL]**: The value specified by the `cond` or `attr` parameter is invalid.

Related Information

Functions: `pthread_cond_destroy(3)`, `pthread_cond_signal(3)`, `pthread_cond_broadcast(3)`, `pthread_cond_wait(3)`, `pthread_cond_timedwait(3)`
The `pthread_cond_signal()` function wakes up a thread, if one exists, that is waiting for the specified condition to be satisfied.

If more than one thread is waiting on the condition, the thread to be awakened will be determined by the scheduler.

This call has no effect if no threads are waiting on the condition.

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.
Errors

If the `pthread_cond_signal()` function fails, `errno` may be set to the following value:

[EINVAL] The value specified by the `cond` parameter is invalid.

Related Information

Functions: `pthread_cond_wait(3)`, `pthread_cond_timedwait(3)`
**pthread_cond_timedwait**

**Purpose**
Waits on a condition variable for a specified period of time

**Library**
Threads Library (libpthreads.a)

**Synopsis**
```
#include <pthread.h>
int pthread_cond_timedwait(
    pthread_cond_t cond,
    pthread_mutex_t mutex,
    struct timespec *abstime);
```

**Parameters**
- `cond` Specifies the condition variable to wait on.
- `mutex` Specifies the mutex in which the condition variable is located; the mutex must be locked by the calling thread.
- `abstime` Specifies the time in nanoseconds to wait for the condition variable to be satisfied.

**Description**
The `pthread_cond_timedwait()` function unlocks the mutex specified by the mutex parameter and causes the calling thread to wait on the specified condition variable. If the condition is satisfied within the time specified by the `abstime` parameter, the mutex is relocked and the function returns. If the absolute time specified by `abstime` elapses before the condition is signaled, an error is returned with mutex relocked.

**Notes**
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.
Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.

Errors

If the pthread_cond_timedwait( ) function fails, errno may be set to one of the following values:

[EINVAL] The value specified by the mutex, cond, or abstime parameter is invalid.

[EAGAIN] The time specified by abstime has elapsed.

[EDEADLK] The calling thread does not own the mutex.

Related Information

Functions: pthread_cond_signal(3), pthread_cond_broadcast(3)
**pthread_cond_wait**

**Purpose**
Waits on a condition variable

**Library**
Threads Library (libpthreads.a)

**Synopsis**
```
#include <pthread.h>
int pthread_cond_wait(
    pthread_cond_t cond,
    pthread_mutex_t mutex);
```

**Parameters**
- `cond`: Specifies the condition variable to wait on.
- `mutex`: Specifies the mutex in which the condition variable is located; the mutex must be locked by the calling thread.

**Description**
The `pthread_cond_wait()` function unlocks the mutex specified by the `mutex` parameter and causes the calling thread to wait on the specified condition variable. When the condition is satisfied, the mutex is relocked and the function returns. The condition should be retested after the return to ensure the thread has not been erroneously awakened.

Use the `pthread_cond_signal()` or `pthread_cond_broadcast()` function to indicate that the condition has been satisfied.

**Notes**
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.
Example

```c
pthread_mutex_lock(&mutex);
while (!condition_true)
    pthread_cond_wait(&cond,&mutex);
/*
  * condition is valid here
  */
pthread_mutex_unlock(&mutex);
```

Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors

If the `pthread_cond_wait()` function fails, `errno` may be set to one of the following values:

- [EINVAL] The value specified by the `mutex` or `cond` parameter is invalid.
- [EDEADLK] The calling thread is not the owner of `mutex`.

Related Information

Functions: `pthread_cond_signal(3), pthread_cond_broadcast(3)`
pthread_condattr_create

**Purpose**

Creates a condition variable attributes object

**Library**

Threads Library (libpthreads.a)

**Synopsis**

```c
#include <pthread.h>
int pthread_condattr_create(
    pthread_condattr_t *attr);
```

**Parameters**

- `attr` Specifies the address in which the ID for the new condition variable attributes object will be stored.

**Description**

The `pthread_condattr_create()` function creates a condition variable attributes object initialized with the default values for the defined attributes and stores its ID in `attr`. When you create a new condition variable (with the `pthread_cond_init()` function), you use an attributes object to specify the attributes to be used for that condition variable.

No condition variable attributes are currently defined.

**Notes**

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

**Return Values**

Upon successful completion, the ID of the created condition variable attributes object is stored in `*attr`, and a value of 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.
Errors

If the `pthread_condattr_create()` function fails, `errno` may be set to one of the following values:

[ENOMEM] There is not enough memory to create the condition attributes object. This is not a temporary condition.

[EINVAL] The value specified by the `attr` parameter is invalid.

Related Information

Functions: `pthread_cond_init(3)`
pthread_condattr_delete

Purpose
Deletes a condition variable attributes object

Library
Threads Library (libpthreads.a)

Synopsis
#include <pthread.h>

int pthread_condattr_delete(
    pthread_condattr_t *attr);

Parameters
attr Specifies the address of the ID of the condition variable attributes
        object to be deleted.

Description
The pthread_condattr_delete( ) function deletes a condition variable attributes
object, which allows the resources for attr to be reclaimed.

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be
changed to conform to the final version.

Return Values
Upon successful completion, the attr parameter is set to an illegal value, and a
value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate
the error.

Errors
If the pthread_condattr_delete( ) function fails, errno may be set to the
following value:

[EINVAL] The value specified by the attr parameter is invalid.
Functions

pthread_condattr_delete(3)

Related Information

Functions: pthread_cond_init(3), pthread_condattr_create(3)
pthread_create

**Purpose**
Creates a thread

**Library**
Threads Library (libpthread.a)

**Synopsis**
```c
#include <pthread.h>
int pthread_create(
    pthread_t *thread,
    pthread_attr_t attr,
    void *( *start_routine) (void *arg),
    void *arg);
```

**Parameters**
- **thread**: Specifies the address in which the ID for the new thread will be stored.
- **attr**: Specifies the address of the attributes object to use in creating the new thread.
- **start_routine**: Specifies the address of the routine to be executed by the new thread.
- **arg**: Specifies the single argument to be passed to the `start_routine` parameter.

**Description**
The `pthread_create()` function creates a new thread, with attributes specified by the `attr` parameter. If `attr` is `pthread_attr_default`, the default attributes are used.

The thread is created executing `start_routine`, with `arg` as its sole argument. If `start_routine` returns, an implicit call to the `pthread_exit()` function is made using the return value of `start_routine` as the exit status.

Variables accessible to one thread in a process are available to all other threads in that process. Use the `pthread_mutex_init()` function to create a mutex for controlling access to shared data. Use the `pthread_keycreate()` function to create a key for accessing thread-specific data.

Each thread has its own **cancelability state**, which determines the thread’s response to a cancellation request (that is, whether or not the thread can be canceled, and when it can be canceled). There are two types of cancelability:
general cancelability (which is set with pthread_setcancel() function), and asynchronous cancelability (which is set with pthread_setasynccancel() function). They work together to determine a thread’s cancelability state. When a thread is created, general cancelability is enabled and asynchronous cancelability is disabled, which means that the thread can only be canceled at cancellation points.

Notes

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values

Upon successful completion, the ID of the created thread is stored at *thread, and a value of 0 (zero) is returned. Otherwise, no thread is created, -1 is returned, and errno is set to indicate the error.

Errors

If the pthread_create() function fails, errno may be set to one of the following values:

[EAGAIN] The system lacks the resources necessary to create another thread.

[EAGAIN] The new thread cannot be created without exceeding the system-imposed limit on the total number of threads allowed for each user.

[ENOMEM] There is not enough memory to create the thread. This is not a temporary condition.

[EINVAL] The value specified by the thread or attr parameter is invalid.

Related Information

Functions: fork(2), pthread_exit(3), pthread_join(3)
pthread_detach

Purpose
Detaches a thread

Library
Threads Library (libpthreads.a)

Synopsis
#include <pthread.h>

int pthread_detach(
    pthread_t *thread);

Parameters
thread Specifies the address of the ID of the thread to detach.

Description
The pthread_detach( ) function indicates that all resources for thread may be reclaimed when thread terminates. This may include storage for thread’s return value. If thread has not terminated, pthread_detach( ) will not cause it to terminate, but will cause the storage to be reclaimed after thread terminates.

Once a thread has been detached, any subsequent calls to the pthread_join( ) function will fail.

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values
Upon successful completion, the thread parameter is set to an illegal value, and a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.
Errors

If the `pthread_detach()` function fails, `errno` may be set to one of the following values:

- **[EINVAL]** The value specified by the `thread` parameter is invalid.
- **[ESRCH]** The value specified by the `thread` parameter does not refer to an existing thread.

Related Information

Functions: `pthread_join(3)`
pthread_equal

**Purpose**
Compares two thread identifiers

**Library**
Threads Library (libpthreads.a)

**Synopsis**
```
#include <pthread.h>
int pthread_equal(
    pthread_t t1,
    pthread_t t2);
```

**Parameters**
- `t1`: Specifies a thread to be compared with the thread represented by `t2`.
- `t2`: Specifies a thread to be compared with the thread represented by `t1`.

**Description**
The `pthread_equal()` function determines whether two thread identifiers are equivalent.

**Notes**
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

**Return Values**
If `t1` is equal to `t2`, a nonzero value is returned. Otherwise, 0 (zero) is returned.

**Related Information**
Functions: `pthread_create(3)`
**Functions**

**pthread_exit(3)**

---

**pthread_exit**

**Purpose**
Terminates the calling thread

**Library**
Threads Library (libpthreads.a)

**Synopsis**
```c
#include <pthread.h>

void pthread_exit(
    void *status);
```

**Parameters**
- `status` Specifies the exit status of the thread.

**Description**

The `pthread_exit()` function terminates the calling thread and saves the exit status. This status is thereby made available to any thread that joins with this thread (using the `pthread_join()` function).

The `pthread_exit()` function is called implicitly when a thread returns from the start routine that was used to create the thread; the routine’s return value serves as the thread’s exit status. The process itself exits when the last thread calls `pthread_exit()`. If the last thread to call `pthread_exit()` has been detached, the process exit status will be 0 (zero). Otherwise, it will be -1.

**Notes**

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

**Return Values**

The `pthread_exit()` function cannot return to its caller.

**Related Information**

Functions: `pthread_create(3), pthread_join(3)`
pthread_getspecific

Purpose
Returns the value bound to a key

Library
Threads Library (libpthreads.a)

Synopsis
#include <pthread.h>
int pthread_getspecific(
    pthread_key_t key,
    void **value);

Parameters
key
    Specifies the address of the key that the value parameter is bound to.
value
    Specifies the address in which the thread-specific data is stored.

Description
The pthread_getspecific() function stores the value that is bound to the specified key for the calling thread in the value parameter. If no data has been bound, then a value of null will be stored.

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values
Upon successful completion, the value bound to the key parameter is stored at the value parameter, and a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.
Errors

If the `pthread_getspecific()` function fails, `errno` may be set to the following value:

- [EINVAL] The value specified by the `key` parameter is invalid.

Related Information

Functions: `pthread_keycreate(3)`, `pthread_setspecific(3)`
pthread_join

Purpose  Waits for a thread to terminate

Library  Threads Library (libpthreads.a)

Synopsis  
```
#include <pthread.h>
int pthread_join(
    pthread_t thread,
    void **status);
```

Parameters

- `thread`  Specifies the ID of the thread to wait for.
- `status`  Specifies the location in which the exit status of the joined thread will be stored.

Description

The `pthread_join()` function blocks execution of the calling thread until the target thread, specified by the `thread` parameter, terminates. If the target thread has already terminated, `pthread_join()` returns without blocking.

When the target thread exits, the exit status of the thread is stored in the `status` parameter unless `status` is a null pointer.

A thread may be joined by many threads until it is detached.

Notes

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.
Errors

If the `pthread_join()` function fails, `errno` may be set to one of the following values:

[EINVAL]      The value specified by the `thread` parameter is invalid.
[ESRCH]       The value specified by the `thread` parameter does not refer to an existing thread.
[EDEADLK]     A deadlock condition was detected: the target thread is the calling thread.

Related Information

Functions: `pthread_create(3), wait(2)`
pthread_keycreate

**Purpose**

Creates a key to be used with thread-specific data

**Library**

Threads Library (libpthreads.a)

**Synopsis**

```
#include <pthread.h>

int pthread_keycreate(
    pthread_key_t *key,
    void (*destructor)(void *value);
```

**Parameters**

- `key` Specifies the address in which the new key will be stored.
- `value` Specifies the value associated with the key.
- `destructor` Specifies the address of an optional destructor function.

**Description**

The `pthread_keycreate()` function creates a key. A key is an opaque object that can be seen by all of the threads in a process. Each thread can bind its own value to that key using the `pthread_setspecific()` function; the value is maintained by the thread until the thread exits.

Ordinarily, the value that a thread binds to a key will be a pointer to storage allocated dynamically on behalf of that thread. To have this storage freed when the thread exits, use the `destructor` function. If the old value needs to be destroyed before the new value is bound, then the calling thread must use the `pthread_getspecific()` function and call the destructor explicitly itself. The destructor function is also called when the thread exits if the value bound is not null. If you do not specify `destructor`, no destructor function is called.

**Notes**

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.
Return Values

Upon successful completion, the newly created key is stored at *key, and a value of 0 (zero) is returned. Otherwise, no key is created, -1 is returned, and errno is set to indicate the error.

Errors

If the pthread_keycreate( ) function fails, errno may be set to one of the following values:

[EAGAIN]  There is not enough memory to create the key.

[ENOMEM]  The key name space is exhausted, so the key cannot be allocated. This is not a temporary condition.

[EINVAL]  The value specified by the destructor, value, or key parameter is invalid.

Related Information

Functions: pthread_getspecific(3), pthread_setspecific(3)
pthread_mutex_destroy

Purpose
Deletes a mutex

Library
Threads Library (libpthreads.a)

Synopsis
#include <pthread.h>

int pthread_mutex_destroy(
   pthread_mutex_t *mutex);

Parameters
mutex Specifies the address of the ID of the mutex to be deleted.

Description
The pthread_mutex_destroy( ) function deletes the specified mutex, which allows the resources of the mutex to be reclaimed.

Attempting to lock or unlock a mutex that has been deleted will result in undefined behavior.

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values
Upon successful completion, the mutex parameter is set to an illegal value, and a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.
Errors

If the `pthread_mutex_destroy()` function fails, `errno` may be set to one of the following values:

- [EBUSY] The mutex is locked.
- [EINVAL] The value specified by the `mutex` parameter is invalid.

Related Information

Functions: `pthread_mutex_init(3)`, `pthread_mutex_lock(3)`, `pthread_mutex_unlock(3)`, `pthread_mutex_trylock(3)`
pthread_mutex_init

**Purpose**
Creates a mutex

**Library**
Threads Library (libpthreads.a)

**Synopsis**
```c
#include <pthread.h>

int pthread_mutex_init(
    pthread_mutex_t *mutex,
    pthread_mutexattr_t attr);
```

**Parameters**
- `mutex`: Specifies the address in which the ID for the new mutex will be stored.
- `attr`: Specifies the attributes object to use in creating the new mutex.

**Description**
The `pthread_mutex_init()` function creates a new mutex with attributes specified by the `attr` parameter. If `attr` is `pthread_mutexattr_default`, the default attributes are used.

A mutex (from "mutual exclusion") is used to serialize the access of multiple threads to shared data. Mutexes should only be used for synchronizing threads within a single process; using mutexes outside of a single process results in undefined behavior.

Because a mutex lock is not a cancellation point, use mutexes to protect resources that will be held only for short fixed periods of time, where the absence of cancelability will not cause problems. Use a condition variable to protect resources that need to be held exclusively for long periods of time.

**Notes**
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.
Return Values

Upon successful completion, the ID of the created mutex is stored at *mutex, and a value of 0 (zero) is returned. Otherwise, no mutex is created, -1 is returned, and errno is set to indicate the error.

Errors

If the pthread_mutex_init( ) function fails, errno may be set to one of the following values:

[EAGAIN] The system lacks the resources necessary to create another mutex.

[EAGAIN] The new mutex cannot be created without exceeding the system-imposed limit on the total number of mutexes allowed for each user.

[ENOMEM] There is not enough memory to create the mutex object. This is not a temporary condition.

[EINVAL] The value specified by the mutex or attr parameter is invalid.

Related Information

Functions: pthread_mutex_destroy(3), pthread_mutex_lock(3), pthread_mutex_unlock(3), pthread_mutex_trylock(3)
pthread_mutex_lock

Purpose

Locks a mutex

Library

Threads Library (libpthreads.a)

Synopsis

```c
#include <pthread.h>

int pthread_mutex_lock(
    pthread_mutex_t *mutex);
```

Parameters

`mutex` Specifies the ID of the mutex to be locked.

Description

The `pthread_mutex_lock()` function locks the specified mutex and makes the calling thread the owner of the mutex. If mutex is already locked, the `pthread_mutex_lock()` function blocks the calling thread until the mutex is available.

Because the `pthread_mutex_lock()` function is not a cancellation point, you can safely call the `pthread_mutex_lock()` function during a cleanup routine. During cleanup, it is often necessary to lock a mutex so that you can change the state that says that a resource is owned. However, care must be taken to ensure that the thread does not already have the mutex locked.

Notes

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, no mutex is created, -1 is returned, and `errno` is set to indicate the error.
Errors

If the `pthread_mutex_lock()` function fails, `errno` may be set to one of the following values:

- `[EINVAL]` The value specified by the `mutex` parameter is invalid.
- `[EDEADLK]` The mutex was already locked by the calling thread.

Related Information

Functions: `pthread_mutex_init(3)`, `pthread_mutex_destroy(3)`, `pthread_mutex_trylock(3)`, `pthread_mutex_unlock(3)`
pthread_mutex_trylock

Purpose  Tries once to lock a mutex

Library  Threads Library (libpthreads.a)

Synopsis  
```
#include <pthread.h>

int pthread_mutex_trylock(
    pthread_mutex_t *mutex);
```

Parameters  

mutex  Specifies the ID of the mutex to lock.

Description  
The `pthread_mutex_trylock( )` function attempts to lock the specified mutex. If the mutex is already locked, `pthread_mutex_trylock( )` returns immediately indicating the lock failed.

Notes  
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values  
If the mutex is owned by a thread (which may be the calling thread), a value of 0 (zero) is returned. If a lock on the mutex is acquired, a value of 1 is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.
Errors

If the `pthread_mutex_trylock()` function fails, `errno` may be set to the following value:

[EINVAL] The value specified by the `mutex` parameter is invalid.

Related Information

Functions: `pthread_mutex_init(3)`, `pthread_mutex_destroy(3)`
pthread_mutex_unlock

Purpose
Unlocks a mutex

Library
Threads Library (libpthreads.a)

Synopsis
#include <pthread.h>
int pthread_mutex_unlock(
    pthread_mutex_t *mutex);

Parameters
mutex Specifies the ID of the mutex to unlock.

Description
The pthread_mutex_unlock() function unlocks the specified mutex. When the mutex is unlocked, if more than one thread is waiting for the mutex, the next thread to lock the mutex is determined by the scheduler.

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values
Upon successful completion, a value of 0 (zero) is returned. Otherwise, no mutex is created, -1 is returned, and errno is set to indicate the error.
Errors

If the `pthread_mutex_unlock()` function fails, `errno` may be set to the following values:

- [EINVAL] The value specified by the `mutex` parameter is invalid.
- [EPERM] The mutex is not locked by the calling thread.

Related Information

Functions: `pthread_mutex_init(3)`, `pthread_mutex_destroy(3)`, `pthread_mutex_lock(3)`
pthread_mutexattr_create

**Purpose**
Creates a mutex attributes object

**Library**
Threads Library (libpthreads.a)

**Synopsis**
```
#include <pthread.h>
int pthread_mutexattr_create(
    pthread_mutexattr_t *attr);
```

**Parameters**
- *attr*: Specifies the address in which the ID for the new mutex attributes object will be stored.

**Description**
The `pthread_mutexattr_create()` function creates a mutex attributes object initialized with the default values for the defined attributes, and stores its ID in the `attr` parameter. When you create a new mutex (with the `pthread_mutex_init()` function), you use an attributes object to specify the attributes to be used for that mutex.

No mutex attributes are currently defined.

**Notes**
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

**Return Values**
Upon successful completion, the ID of the created mutex attributes object is stored in `*attr`, and a value of 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.
Errors

If the `pthread_mutexattr_create()` function fails, `errno` may be set to one of the following values:

- **[ENOMEM]** There is not enough memory to create the mutex attributes object. This is not a temporary condition.
- **[EINVAL]** The value specified by the `attr` parameter is invalid.

Related Information

Functions: `pthread_create(3), pthread_mutex_init(3), pthread_cond_init(3)`
pthread_mutexattr_delete

**Purpose**  
Deletes a mutex attributes object

**Library**  
Threads Library (libpthreads.a)

**Synopsis**  
#include <pthread.h>

int pthread_mutexattr_delete(
    pthread_mutexattr_t *attr);

**Parameters**

*attr  
Specifies the address of the ID of the mutex attributes object to be deleted.

**Description**  
The `pthread_mutexattr_delete()` function deletes a mutex attributes object, which allows the storage for `attr` to be reclaimed.

**Notes**  
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

**Return Values**  
Upon successful completion, *attr is set to an illegal value, and a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.

**Errors**  
If the `pthread_mutexattr_delete()` function fails, errno may be set to the following value:

[EINVAL]  
The value specified by the attr parameter is invalid.

**Related Information**  
Functions: `pthread_mutexattr_create(3)`
pthread_once

Purpose  Calls an initialization routine

Library  Threads Library (libpthreads.a)

Synopsis  

```c
#include <pthread.h>
static pthread_once_t once_block = pthread_once_init;
int pthread_once(
    pthread_once_t *once_block,
    void(*init_routine)());
```

Parameters

- `once_block` Specifies a name to use for the routine that is used to check whether the initialization routine has already been executed.
- `init_routine` Specifies the name of the initialization routine.

Description

The pthread_once() function determines whether or not init_routine has been called by a previous pthread_once() call; if init_routine has not been called, pthread_once() calls it.

The pthread_once() function does not return to any calling thread until the init_routine has been completed. You should declare a single once_block for each initialization routine.

Undefined behavior results if once_block is not initialized or is not declared as static.

Notes

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.
Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.

Errors

If the pthread_once() function fails, errno may be set to one of the following values:

[EINVAL] The value specified by the once_block or init_routine parameter is invalid.

Related Information

Functions: pthread_mutex_init(3), pthread_cond_init(3)
Functions

pthread_self(3)

pthread_self

Purpose
Returns the ID of the calling thread

Library
Threads Library (libpthreads.a)

Synopsis
#include <pthread.h>

pthread_t pthread_self(void);

Description
The pthread_self() function returns the thread ID of the calling thread. You can use the returned thread ID to pass as the thread argument to other thread calls.

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values
Upon successful completion, the pthread_self() function returns the thread ID of the calling thread.

Related Information
Functions: pthread_create(3)
Purpose
Enables or disables the asynchronous cancelability of the calling thread

Library
Threads Library (libpthreads.a)

Synopsis
```
#include <pthread.h>
int pthread_setasynccancel(
    int state);
```

Parameters

```
state
Specifies the new cancelability state; legal values are:

CANCEL_ON
   Enables asynchronous cancellation

CANCEL_OFF
   Disables asynchronous cancellation
```

Description

The `pthread_setasynccancel()` function sets the calling thread’s asynchronous cancelability state to that indicated by the `state` parameter and returns the previous asynchronous cancelability state.

By default, asynchronous cancelability is disabled, and general cancelability is enabled (see the `pthread_setcancel()` function), which means that the thread can only be canceled at cancellation points.

If you enable both asynchronous cancelability and general cancelability, the thread can be canceled at any time.

If you disable general cancelability, the thread cannot be canceled; the state of asynchronous cancelability is ignored.

You should not enable asynchronous cancelability if the thread is executing in a critical section, or is in another state that would be difficult or impossible to recover from (for example, if the thread is contending for a shared resource).
The C standard functions that you can safely call with asynchronous cancelability enabled are the character handling functions, the mathematical functions, the string handling functions, and the abs() function. The effect of calling any other C standard function with asynchronous cancelability enabled is unspecified.

The character handling functions that you can safely call with asynchronous cancelability enabled are:

- isalnum
- isalpha
- iscntrl
- isdigit
- isgraph
- islower
- isprint
- ispunct
- isspace
- isupper
- isxdigit
- tolower
- toupper

The mathematical functions that you can safely call with asynchronous cancelability enabled are:

- acos
- asin
- atan
- atan2
- cos
- sin
- tan
- cosh
- sinh
- tanh
- exp
- frexp
- ldexp
- log
- log10
- modf
- pow
- sqrt
- ceil
- fabs
- floor
- fmod

The string handling functions that you can safely call with asynchronous cancelability enabled are:

- memcpy
- memmove
- strcpy
- strncpy
- memset
- strerror
- strlen
- strcat
- strncat
- memcmp
- strcmp
- strcoll
- strncmp
- strxfrm
- memchr
- strchr
- strcspn
- strpbrk
- strrrchr
- strspn
- strstr
- strtok
- memset
- strlen

Notes

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values

Upon successful completion, the previous value of the cancelability state is returned. Otherwise, -1 is returned and errno is set to indicate the error.
Errors

If the `pthread_setasynccancel()` function fails, `errno` may be set to the following value:

[EINVAIL] The specified state is not CANCEL_ON or CANCEL_OFF.

Related Information

Functions: `pthread_cancel(3), pthread_setcancel(3)`
pthread_setcancel

**Purpose**
Enables or disables the general cancelability of the calling thread

**Library**
Threads Library (libpthreads.a)

**Synopsis**
```c
#include <pthread.h>
int pthread_setcancel(
    int state);
```

**Parameters**
- `state` Specifies the new cancelability state; legal values are:
  - CANCEL_ON
    Enables general cancellation
  - CANCEL_OFF
    Disables general cancellation

**Description**
The `pthread_setcancel()` function sets the calling thread’s general cancelability to that indicated by the `state` parameter and returns the previous cancelability state.

By default, general cancelability is enabled and asynchronous cancelability (see the `pthread_setasynccancel()` function) is disabled, which means that the thread can only be canceled at cancellation points. Cancellation points include the following:

- While waiting on a condition variable (within a call to the `pthread_cond_wait()` or `pthread_cond_timedwait()` function)
- While waiting for the termination of another thread (within a call to the `pthread_join()` function)
- Where the `pthread_testcancel()` function has been called
- Where the `pthread_setcancel()` function has been called with the parameter CANCEL_ON

If the general cancelability of the target thread has been disabled, the termination of the thread is held pending until general cancelability is reenabled. If general cancelability is enabled and asynchronous cancelability is enabled, the termination
of the target thread begins immediately. If general cancelability is enabled and asynchronous cancelability is disabled, termination is held pending until the next cancellation point.

Notes

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values

Upon successful completion, the previous value of the cancelability state is returned. Otherwise, -1 is returned and errno is set to indicate the error.

Errors

If the pthread_setcancel() function fails, errno may be set to the following value:

[EINVAL] The specified state is not CANCEL_ON or CANCEL_OFF.

Related Information

Functions: pthread_cancel(3), pthread_setasynccancel(3)
Functions

pthread_setspecific(3)

pthread_setspecific

Purpose
Binds a thread-specific value to a key

Library
Threads Library (libpthreads.a)

Synopsis
#include <pthread.h>

int pthread_setspecific(
    pthread_key_t key,
    void *value);

Parameters
key Specifies the key that the value parameter will be bound to.
value Specifies the value of the thread-specific data to be bound to the key.

Description
The pthread_setspecific() function binds a thread-specific value with a key created with a previous call to the pthread_keycreate() parameter. Different threads may bind different values to the same key. The values are typically pointers to blocks of dynamically allocated memory that will be used only by the calling thread.

The calling thread must explicitly destroy the old value itself, if required, before binding the new value using this call.

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Return Values
Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.
Errors

If the `pthread_setspecific()` function fails, `errno` may be set to the following value:

```
[EINVAL]  The value specified by the `key` parameter is invalid.
```

Related Information

Functions: `pthread_keycreate(3)`, `pthread_getspecific(3)`
Functions

pthread_testcancel(3)

pthread_testcancel

Purpose

Creates a cancellation point in the calling thread

Library

Threads Library (libpthreads.a)

Synopsis

#include <pthread.h>

void pthread_testcancel (void);

Description

The `pthread_testcancel()` function creates a cancellation point in the calling thread. A cancellation point is a place where it is permissible for the thread to be canceled. A common place for a cancellation point is right before an operation that may block or before or after a long critical section.

If general cancelability is disabled, cancellation points, including `pthread_testcancel()`, are ignored.

Before any cancellation point, you should always set up a cleanup handler that will restore invariants if the thread is canceled at that point, if necessary.

Notes

This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.

Related Information

Functions: pthread_cancel(3)
pthread_yield

Purpose
Allows the scheduler to run another thread instead of the current one

Library
Threads Library (libpthreads.a)

Synopsis
#include <pthread.h>
void pthread_yield(void);

Description
The pthread_yield() function allows the scheduler to determine if another thread could be run in preference to the calling thread. If no other thread is suitable, the scheduler continues to run the calling thread.

Notes
This interface is based on draft 4 of the IEEE P1003.4a standard, and will be changed to conform to the final version.
 ptrace

Purpose
Traces the execution of a child process

Synopsis
#include <sys/signal.h>
#include <sys/ptrace.h>
int ptrace(
    int request,
    int process,
    int *address,
    int data);

Parameters
request Determines the action to be taken by the ptrace() function.
process Specifies the process ID.
address Determined by the value of the request parameter.
data Determined by the value of the request parameter.

Description
The ptrace() function permits a parent process to control execution of a child process. It is primarily used by utility programs to enable breakpoint debugging.

The child process behaves normally until it receives a signal. When a signal is delivered, the child process is stopped, and a SIGCHLD signal is sent to its parent. The parent process can wait for the child process to stop using the wait() function.

When the child process is stopped, its parent process may use the ptrace() function to examine and modify the image memory of the child process, to either terminate the child process or permit it to continue.

As a security measure, the ptrace() function inhibits the set-user ID facility when any subsequent exec function is issued by the child process. When a traced process calls one of the exec functions, it stops before executing the first instruction of the new image as if it had received the SIGTRAP signal.

The request parameter is set to one of the following values. Only the PT_TRACE_ME request may be issued by child processes; the remaining requests can only be used by the parent process. For each request, the process parameter is the process ID of the child process. The child process must be in a stopped state before these requests are made.
PT_TRACE_ME
This request sets the child process trace flag. It must be issued by the child process that is to be traced by its parent process. When the trace flag is set, the child process is left in a stopped state on receipt of a signal, and the action specified by the `sigaction()` function is ignored. The `process`, `address`, and `data` parameters are ignored, and the return value is not defined for this request. Do not issue this request when the parent process does not expect to trace the child process.

PT_READ_I or PT_READ_D
These requests return the address space data of the child process at the location pointed to by the `address` parameter. The PT_READ_I and PT_READ_D requests can be used with equal results. The `data` parameter is ignored. These requests fail when the value of the `address` parameter is not in the address space of the child process or on some machines, when the `address` parameter is not properly aligned. These errors return a value of -1, and the parent process `errno` is set to [EIO].

PT_READ_U
This request returns the variable of the system's per-process data area for the child, specified by the `address` parameter. This area contains the register values and other information about the process. On some machines, the `address` parameter is subject to alignment constraints. The `data` parameter is ignored. This request fails when the value of the `address` parameter is outside of the system's per-process data area for the child. On failure, a value of -1 is returned and the parent process `errno` is set to [EIO].

PT_WRITE_I, PT_WRITE_D
These requests write the value of the `data` parameter into the address space variable of the child process at the location pointed to by the `address` parameter. On some machines, where necessary, the PT_WRITE_I request synchronizes any hardware caches, if present. In all other respects, the PT_WRITE_I and PT_WRITE_D requests can be used with equal results. On some machines, these requests return the previous contents of the address space variable of the child process, while on other machines no useful value is returned. These requests also fail when the value of the `address` parameter is out of range and on some machines, when the `address` parameter is not properly aligned. On failure a value of -1 is returned and the parent process `errno` is set to [EIO].
Functions

ptrace(2)

PT_WRITE_U
This request writes the value of the data parameter into the variable of the system's per-process data area for the child, specified by the address parameter. This area contains the register values and other information about the process. On some machines, the address parameter is subject to alignment constraints. Not all locations within the system's per-process data area for the child may be written. This request fails when the value of the address parameter is outside of the system's per-process data area for the child. On failure, a value of -1 is returned and the parent process errno is set to indicate the error.

PT_CONTINUE
This request permits the child process to resume execution. When the data parameter is 0 (zero), the signal that caused the child process to stop is canceled before the child process resumes execution.

When the data parameter has a valid signal value, the child process resumes execution as though that signal had been received. When the address parameter is equal to 1, execution continues from where it stopped. When the address parameter is not 1, it is assumed to be the address at which the process should resume execution.

This request fails when the data parameter is not 0 (zero) or a valid signal value. On failure, a value of -1 is returned to the parent process and the parent process errno is set to [EIO].

PT_KILL
This request terminates a child process as if the child process called the exit() function.

PT_STEP
This request permits execution to continue in the same manner as PT_CONTINUE; however, as soon as possible after the execution of at least one instruction, execution stops again as if the child process had received the SIGTRAP signal.

Errors

If the ptrace() function fails, errno may be set to one of the following values:

[EIO] The request parameter does not have one of the listed values, or is not valid for the machine type on which the process is executing.

[EIO] The given signal number is invalid.

[EIO] The specified address is either out of bounds or improperly aligned.
ptrace(2)

[ESRCH] The process parameter identifies a child process that does not exist or that has not executed this function with the request parameter PT_TRACE_ME.

[EPERM] The specified process cannot be traced.

[EINVAL] An invalid location was specified for the system’s per-process data area.

[EACCES] The location within the system’s per-process data area could not be modified.

Related Information

Functions: exec(2), sigaction(2), wait(2)
putc, putchar, fputc, putw

**Purpose**
Write a character or a word to a stream

**Library**
Standard I/O Package (libc.a)

**Synopsis**
```
#include <stdio.h>

int putc(
    int c,
    FILE *stream );

int putchar(
    int c );

int fputc(
    int c,
    FILE *stream );

int putw(
    int w,
    FILE *stream );
```

**Parameters**
- `stream` Points to the file structure of an open file.
- `c` Specifies the character to be written.
- `w` Specifies the word to be written.

**Description**
The `putc()` macro writes the character `c` to the output specified by the `stream` parameter. The character is written at the position at which the file pointer is currently pointing, if defined.

The `putchar()` macro is the same as the `putc()` macro except that `putchar()` writes to the standard output.

The `fputc()` function works the same as the `putc()` macro, but `fputc()` is a true function rather than a macro. It runs more slowly than `putc()`, but takes less space per invocation.
Because `putc()` is implemented as a macro, it incorrectly treats a `stream` parameter with side effects, such as `putc(c, *f++)`. For such cases, use the `fputc()` function. Also, use `fputc()` when you need to pass a pointer to this function as a parameter to another function.

The `putw()` function writes the word (int) specified by the `w` parameter to the output specified by the `stream` parameter. The word is written at the position at which the file pointer, if defined, is pointing. The size of a word is the size of an integer and varies from machine to machine. The `putw()` function does not assume or cause special alignment of the data in the file.

Because of possible differences in word length and byte ordering, files written using the `putw()` function are machine-dependent, and may not be readable using the `getw()` function on a different processor.

With the exception of `stderr`, output streams are, by default, buffered if they refer to files, or line-buffered if they refer to terminals. The standard error output stream, `stderr`, is unbuffered by default, but using the `freopen()` function causes it to become buffered or line-buffered. Use the `setbuf()` function to change the stream buffering strategy.

When an output stream is unbuffered, information is queued for writing on the destination file or terminal as soon as it is written. When an output stream is buffered, many characters are saved and written as a block. When an output stream is line-buffered, each line of output is queued for writing on the destination terminal as soon as the line is completed (that is, as soon as a newline character is written or terminal input is requested).

The `st_ctime` and `st_mtime` fields of the file are marked for update between the successful execution of the `putc()`, `putw()`, `putchar()`, or `fputc()` function, and the next successful completion of a call to the `fflush()` or `fclose()` function on the same stream, or a call to the `exit()` or `abort()` function.

**Notes**

The reentrant versions of these functions are locked against multiple threads calling them simultaneously. This will incur an overhead to ensure integrity of the stream. The unlocked versions of these calls may be used safely, providing that the stream is locked when the calls are used with the `flockfile()` and `funlockfile()` functions.

**AES Support Level:** Full use (`putc()`, `fputc()`, `putchar()`)

Trial use (`putw()`)

1–556
Functions

putc(3)

Return Values

Upon successful completion, these functions each return the value written. If these functions fail, they return the constant EOF. They fail if the stream parameter is not open for writing, or if the output file size cannot be increased. Because the EOF value is a valid integer, you should use the ferror() function to detect the putw() parameter errors.

Errors

The putc(), putw(), putchar(), and fputc() functions fail if either the stream is unbuffered, or the stream's buffer needed to be flushed and the function call caused an underlying write() or lseek() to be invoked. In addition, if the putc(), putw(), putchar(), or fputc() function fails, errno may be set to one of the following values:

[EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying stream and the process would be delayed in the write operation.

[EBADF] The file descriptor underlying stream is not a valid file descriptor open for writing.

[EFBIG] An attempt was made to write to a file that exceeds the process’ file size limit or the maximum file size.

[EINTR] The read operation was interrupted by a signal which was caught, and no data was transferred.

[EIO] The implementation supports job control, the process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

[ENOSPC] There was no free space remaining on the device containing the file.

[EPipe] An attempt was made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

Related Information

Functions: getc(3), getwc(3), printf(3), puts(3), putwc(3), unlocked_putc(3), unlocked_putchar(3)
putenv

**Purpose**
Sets an environment variable

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <stdlib.h>

int putenv (char *string);
```

**Parameters**
- **string**: Points to a name=value string.

**Description**
The `putenv()` function sets the value of an environment variable by altering an existing variable or by creating a new one. The `string` parameter points to a string of the form name=value, where `name` is the environment variable and `value` is the new value for it.

**Notes**
The `putenv()` function manipulates the `environ` external variable, and it can be used in conjunction with the `getenv()` function. However, the third parameter to the main function (the environment pointer), is not changed.

The `putenv()` function uses the `malloc()` function to enlarge the environment.

**AES Support Level**: Trial use

**Return Values**
Upon successful completion, a value of 0 (zero) is returned. If the `malloc()` function is unable to obtain sufficient space to expand the environment, then the `putenv()` function returns a nonzero value.

**Related Information**
Functions: `clearenv(3)`, `exec(2)`, `getenv(3)`, `malloc(3)`
putlong

**Purpose**
Places long byte quantities into the byte stream

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

void putlong (  
    unsigned long long,  
    u_char *message_ptr );
```

**Parameters**
- `long` Represents a 32-bit integer.
- `message_ptr` Represents a pointer into the byte stream.

**Description**
The `putlong()` function places long byte quantities into the byte stream or arbitrary byte boundaries.

The `putlong()` function is one of a set of subroutines that form the resolver, a set of functions that resolve domain names. Global information that is used by the resolver functions is kept in the `_res` data structure. The `/include/resolv.h` file contains the `_res` data structure definition.

**Files**
- `/etc/resolv.conf` Lists the name server and domain name.

**Related Information**
Functions: `dn_comp(3)`, `dn_expand(3)`, `dn_find(3)`, `dn_skipname(3)`, `getlong(3)`, `getshort(3)`, `putshort(3)`, `res_init(3)`, `res_mkquery(3)`, `res_send(3)`
puts, fputs

Purpose
Writes a string to a stream

Library
Standard I/O Library (libc.a)

Synopsis
```c
#include <stdio.h>
int puts (const char *string);
int fputs (const char *string, FILE *stream);
```

Parameters
- `string`: Points to a string to be written to output.
- `stream`: Points to the `FILE` structure of an open file.

Description
The `puts()` function writes the null-terminated string pointed to by the `string` parameter, followed by a newline character, to the standard output stream, `stdout`.

The `fputs()` function writes the null-terminated string pointed to by the `string` parameter to the output stream specified by the `stream` parameter. The `fputs()` function does not append a newline character.

Neither function writes the terminating null character.

The `st_ctime` and `st_mtime` fields of the file are marked for update between the successful execution of the `puts()` or `fputs()` function, and the next successful completion of a call to the `fflush()` or `fclose()` function on the same stream, or a call to the `exit()` or `abort()` function.

Notes

AES Support Level: Full use
Return Values

Upon successful completion, the `puts()` and `fputs()` functions return the number of characters written. Both subroutines return EOF on an error. This happens if the routines try to write on a file that has not been opened for writing.

Errors

The `puts()` and `fputs()` functions fail if either the stream is unbuffered, or the stream’s buffer needed to be flushed and the function call caused an underlying the `write()` or `lseek()` function to be invoked. In addition, if the `puts()` or `fputs()` function fails, `errno` may be set to one of the following values:

- **[EAGAIN]** The O_NONBLOCK flag is set for the file descriptor underlying stream and the process would be delayed in the write operation.
- **[EBADF]** The file descriptor underlying stream is not a valid file descriptor open for writing.
- **[EFBIG]** An attempt was made to write to a file that exceeds the process’ file size limit or the maximum file size.
- **[EINTR]** The read operation was interrupted by a signal which was caught, and no data was transferred.
- **[EIO]** The implementation supports job control, the process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
- **[ENOSPC]** There was no free space remaining on the device containing the file.
- **[EPIPE]** An attempt was made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

Related Information

Functions: `gets(3), getws(3), printf(3), putc(3), putwc(3), putws(3)`
putshort

**Purpose**
Places short byte quantities into the byte stream

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

void putshort (
    unsigned short short,
    u_char *message_ptr);
```

**Parameters**
- `short` Represents a 16-bit integer.
- `message_ptr` Represents a pointer into the byte stream.

**Description**
The `putshort()` function puts short byte quantities into the byte stream or arbitrary byte boundaries.

The `putshort()` function is one of a set of subroutines that form the resolver, a set of functions that resolve domain names. Global information that is used by the resolver functions is kept in the `_res` data structure. The `/include/resolv.h` file contains the `_res` structure definition.
Files

/etc/resolv.conf

Lists the name server and domain name.

Related Information

Functions: dn_comp(3), dn_expand(3), dn_find(3), dn_skipname(3),
getlong(3), getshort(3), putlong(3), res_init(3), res_mkquery(3), res_send(3)
putwc, putwchar, fputwc

Purpose
Writes a character or a word to a stream

Library
Standard I/O Library (libc.a)

Synopsis
#include <stdio.h>

int putwc(
    int c,
    FILE *stream );

int putwchar(
    int c );

int fputwc(
    int c;
    FILE *stream );

Parameters

- $c$: Specifies the NLchar to be written.
- $stream$: Points to the output data.

Description

The putwc(), putwchar(), and fputwc() functions are provided when Japanese Language Support is installed on your system. They parallel the putc(), putchar(), and fputc() functions.

The putwc() function writes the NLchar specified by the $c$ parameter to the $stream$ parameter as 1 or 2 bytes.

The putwchar() macro works like the putwc() function, except that putwchar() writes the specified NLchar to the standard output.

The fputwc() function works the same as putwc().
With the exception of stderr, output streams are, by default, buffered if they refer to files, or line-buffered if they refer to terminals. The standard error output stream, stderr, is unbuffered by default, but using the freopen() function causes it to become buffered or line-buffered. Use the setbuf() function to change the stream's buffering strategy.

Return Values

Upon successful completion, these functions each return the value written. If these functions fail, they return the constant EOF. They fail if the stream parameter is not open for writing, or if the output file size cannot be increased.

Related Information

Functions: getc(3), getwc(3), printf(3), putc(3), puts(3), wsprintf(3)
putws(3)

putws, fputws

Purpose

Writes a string to a stream

Library

Standard I/O Library (libc.a)

Synopsis

#include <stdio.h>
#include <NLchar.h>

int putws (NLchar *string);

int fputws (NLchar *string,
FILE *stream);

Parameters

string Points to a string to be written to output.

stream Points to the FILE structure of an open file.

Description

The putws() and fputws() functions are provided when Japanese Language Support is installed on your system. They parallel the puts() and fputs() functions.

The putws() function writes the NLchar string pointed to by the string parameter to the standard output stream, stdout. In this case, each element of the string parameter produces either 1 or 2 bytes of output, according to the size required for its encoding. In all other respects, putws() functions like puts().

The fputws() function writes the NLchar string pointed to by the string parameter to the output stream. Again, each element of the string parameter produces either 1 or 2 bytes of output, according to the size required for its encoding. In all other respects, fputws() functions like fputs().
Return Values

Upon successful completion, the `putws()` and `fputws()` functions return the number of characters written. Both subroutines return EOF on an error. This happens if the routines try to write on a file that has not been opened for writing.

Related Information

Functions: `gets(3), getws(3), printf(3), putc(3), puts(3), putwc(3)`
qsort

Purpose
Sorts a table in place

Library
Standard C Library (libc.a)

Synopsis
```c
#include <stdlib.h>

void qsort(
    void *base,
    size_t nmemb,
    size_t size,
    int (*compar)(const void*, const void*));
```

Parameters
- `base`: Points to the first entry in the table.
- `nmemb`: Specifies the number of entries in the table.
- `size`: Specifies the size in bytes of each table entry.
- `compar`: Points to the user-specified function to be used to compare pairs of table elements. The comparison function will be called with two parameters that point to the two elements to be compared. The comparison function must return an integer less than, equal to, or greater than zero, depending on whether the first element in the comparison is considered less than, equal to, or greater than the second element.

Description
The `qsort()` function sorts a table having a specified number of entries. The contents of the table are sorted in ascending order according to a user-specified comparison function (the `strcmp()` function, for example).
Notes

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

When two members compare equal, their order in the sorted array is indeterminate.

AES Support Level: Full use

Related Information

Functions: bsearch(), lsearch()
The `quotactl()` function is used to enable and disable quotas and to manipulate disk quotas for file systems on which quotas have been enabled.

Currently quotas are supported only for the UFS file system. For UFS, a command is composed of a primary command (see below) and a command type that is used to interpret the `id` parameter. Types are supported for interpretation of user identifiers and group identifiers. The UFS specific commands are:

**Q_QUOTAON**

Enable disk quotas for the file system specified by the `path` parameter. The command type specifies the type of the quotas being enabled. The `addr` parameter specifies a file from which to take the quotas. The quota file must exist; it is normally created with the `quotacheck` program. The `id` parameter is unused. Only users with superuser privilege can turn quotas on.
Q_QUOTAOFF
Disable disk quotas for the file system specified by the path parameter. The command type specifies the type of the quotas being disabled. The addr and id parameters are unused. Only users with superuser privilege can turn quotas off.

Q_GETQUOTA
Get disk quota limits and current usage for the user or group (as determined by the command type) with identifier id. The addr parameter points to a struct dqblk structure, defined in the ufs/quota.h header file.

Q_SETQUOTA
Set disk quota limits for the user or group (as determined by the command type) with identifier id. The addr parameter points to a struct dqblk structure, defined in the ufs/quota.h header file. The usage fields of the dqblk structure are ignored. This function is restricted to processes with superuser privilege.

Q_SETUSE
Set disk usage limits for the user or group (as determined by the command type) with identifier id. The addr parameter points to a struct dqblk structure, defined in the ufs/quota.h header file. Only the usage fields are used. This function is restricted to processes with superuser privilege.

Q_SYNC
Update the on-disk copy of quota usages. The command type specifies which type of quotas are to be updated. The id and addr parameters are ignored.

Return Value
Upon successful completion, 0 (zero) is returned. Otherwise, -1 is returned and errno is set to indicate the error.

Errors
If the quotactl() function fails, errno may be set to one of the following values:

[EOPNOTSUPP] The kernel has not been compiled with the QUOTA option.

[EUSERS] The quota table cannot be expanded.

[EINVAL] The cmd parameter or the command type is invalid.

[EINVAL] A pathname contains a character with the high-order bit set.

[EACCES] In Q_QUOTAON, the quota file is not a plain file.

[EACCES] Search permission is denied for a component of a path prefix.
[ENOTDIR]  A component of a path prefix is not a directory.

[ENAMETOOLONG]  A component of the pathname exceeded NAME_MAX, or the entire length of the pathname exceeded PATH_MAX.

[ENOENT]  A filename does not exist.

[ELOOP]  Too many symbolic links were encountered in translating a pathname.

[EROFS]  In Q_QUOTAON, the quota file resides on a read-only file system.

[EIO]  An I/O error occurred while reading from or writing to a file containing quotas.

[EFAULT]  An invalid addr is supplied; the associated structure could not be copied in or out of the kernel.

[EFAULT]  The path parameter points outside the process's allocated address space.

[EPERM]  The call is privileged and the caller does not have appropriate privilege.

Related Information

Commands: quota(1), edquota(8), quotacheck(8), quotaon(8), repquota(8)
raise

Purpose Sends a signal to the executing program

Library Standard C Library (libc.a)

Synopsis
#include <sys/signal.h>

    int raise(
            int signal);

Parameters

    signal Specifies a signal number.

Description

The raise() function sends the signal specified by the signal parameter to the executing program. It is equivalent to the following:

    error = kill(getpid(), signal);

Notes

AES Support Level: Full use

Return Values

Upon successful completion of the raise() function, a value of 0 (zero) is returned. Otherwise, a nonzero value is returned and errno is set to indicate the error.

Errors

If the raise() function fails, errno may be set to the following value:

[EINVAL] The value of the signal parameter is an invalid or unsupported signal number.

Related Information

Functions: kill(2), sigaction(2)
rand, rand_r, srand

Purpose
Generates pseudo-random numbers

Library
Standard C Library (libc.a),
Berkeley Compatibility Library (libbsd.a)
Reentrant Library (libc_r.a)

Synopsis
#include <stdlib.h>

int rand (void);
int rand_r(
    unsigned int *seedptr,
    int *randval);
void srand(
    unsigned int seed);

Parameters
seed Specifies an initial seed value.
seedptr Points to a seed value, updated at each call.
randval Points to a place to store the random number.

Description
The rand() function returns successive pseudo-random numbers in the range from 0 (zero) to RAND_MAX. The sequence of values returned depends on the seed value set with the srand() function. If rand() is called before any calls to srand() have been made, the same sequence will be generated as when srand() is first called with a seed value of 1.

The rand_r() function is the reentrant version of the rand() function, for use with multi-threaded applications. The rand_r() function places the seed value at the address pointed to by seedptr, and places the random number at the address pointed to by randval.

The srand() function resets the random-number generator to a random starting point. The generator is initially seeded with a value of 1.
The `rand()` function is a very simple random-number generator. Its spectral properties, the mathematical measurement of how random the number sequence is, are somewhat weak.

See the `drand48()` and `random()` functions for more elaborate random-number generators that have better spectral properties.

### Notes

The `rand()` function is not supported for multi-threaded applications. Instead, its reentrant equivalent `rand_r()` should be used with multiple threads.

The BSD version of the `rand()` function returns a number in the range 0 to $2^{31} - 1$, rather than 0 to $2^{15} - 1$, and can be used by compiling with the Berkeley Compatibility Library (`libbsd.a`).

There are better random number generators, as noted above; however, the `rand()` and `srand()` functions are the interfaces defined for the ANSI C library.

The following functions define the semantics of the `rand()` and `srand()` functions, and are included here to facilitate porting applications from different implementations:

```c
static unsigned int next = 1;

int rand()
{
    next = next * 1103515245 + 12345;
    return ((next >> 16) & RAND_MAX);
}

void srand (seed)
int seed;
{
    next = seed
}
```

**AES Support Level:** Full use (`rand()`, `srand()`)  

### Return Values

The `rand()` function returns the next pseudo-random number in the sequence.

Upon successful completion, the `rand_r()` function returns a value of 0 (zero). Otherwise, -1 is returned and `errno` is set to indicate the error.

The `srand()` function returns no value.
Errors

If the `rand_r()` function fails, `errno` may be set to the following value:

[EINVAL] Either `seedptr` or `randval` is a null pointer.

Related Information

Functions: `drand48(3), random(3)`
random, srandom, initstate, setstate

Purpose
Generates "better" pseudo-random numbers

Library
Standard C Library (libc.a)

Synopsis
long random ( void );
srandom (      int seed );

char *initstate (      unsigned seed,      char *state,      int size );

char *setstate (      char *state );

Parameters

seed Specifies an initial seed value.
state Points to the array of state information.
size Specifies the size of the state information array.

Description
The random() and srandom() functions are random number generators that have virtually the same calling sequence and initialization properties as the rand() and srand() functions, but produce sequences that are more random. The low dozen bits generated by the rand() function go through a cyclic pattern, and all the bits generated by the random() function are usable. For example, "random()&01" produces a random binary value.

The random() function uses a nonlinear additive feedback random number generator employing a default state array size of 31 long integers to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$. The period of this random number generator is approximately $16 \times (2^{31}-1)$. The size of the state array determines the period of the random number generator. Increasing the state array size increases the period.
With a full 256 bytes of state information, the period of the random-number generator is greater than $2^{69}$, which should be sufficient for most purposes.

Like the `rand()` function, the `random()` function produces by default a sequence of numbers that can be duplicated by calling the `srandom()` function with 1 as the seed. The `srandom()` function, unlike the `srand()` function, does not return the old seed because the amount of state information used is more than a single word.

The `initstate()` and `setstate()` functions handle restarting and changing random-number generators. The `initstate()` function allows a state array, passed in as an argument, to be initialized for future use. The size in bytes of the state array is used by the `initstate()` function to decide how sophisticated a random-number generator to use; the larger the state array, the more random the numbers. Values for the amount of state information are 8, 32, 64, 128, and 256 bytes. Amounts less than 8 bytes generate an error, while other amounts are rounded down to the nearest known value. The `seed` parameter specifies a starting point for the random-number sequence and provides for restarting at the same point. The `initstate()` function returns a pointer to the previous state information array.

Once a state has been initialized, the `setstate()` function allows rapid switching between states. The array defined by the `state` parameter is used for further random-number generation until the `initstate()` function is called or the `setstate()` function is called again. The `setstate()` function returns a pointer to the previous state array.

After initialization, a state array can be restarted at a different point in one of two ways:

1. The `initstate()` function can be used, with the desired seed, state array, and size of the array.
2. The `setstate()` function, with the desired state, can be used, followed by the `srandom()` function with the desired seed. The advantage of using both of these functions is that the size of the state array does not have to be saved once it is initialized.

**Return Values**

The `random()` and `srandom()` functions return a random number. The `initstate()` and `setstate()` functions return a pointer to the previous state information array.
Errors

If the `initstate()` function is called with less than 8 bytes of state information, or if the `setstate()` function detects that the state information has been damaged, error messages are sent to the standard output.

Related Information

Functions: `drand48(3)`, `rand(3)`
Purpose
Allows execution of commands on a remote host

Library
Standard C Library (libe.a)

Synopsis
```c
int rcmd(
    char **host,
    u_short port,
    char *local_user,
    char *remote_user,
    char *command,
    int *err_file_desc);
```

Parameters

- **host**: Specifies the name of a remote host that is listed in the `/etc/hosts` file. If the specified name of the host is not found in this file, the `rcmd()` function fails.

- **port**: Specifies the well-known port to use for the connection. The `/etc/services` file contains the DARPA Internet services, their ports, and socket types.

- **local_user**: Points to user names that are valid at the local host. Any valid user name can be given.

- **remote_user**: Points to user names that are valid at the remote host. Any valid user name can be given.

- **command**: Specifies the name of the command to be executed at the remote host.

- **err_file_desc**: Specifies an integer that controls the set up of communications channels. Integer options are as follows:
  - If a nonzero integer is specified, an auxiliary channel to a control process is set up, and the `error_file_desc` parameter points to the file descriptor for the channel. The control process provides diagnostic output from the remote command on this channel and also accepts bytes as signal numbers to be forwarded to the process group of the command.
Functions

rcmd(3)

- If 0 (zero) is specified, the standard error (stderr) of the remote command is the same as the standard output (stdout). No provision is made for sending arbitrary signals to the remote process. However, it is possible to send out-of-band data to the remote command.

Description

The rcmd( ) (remote command) function allows execution of certain commands on a remote host that supports the rshd(), rlogin(), and rpc() functions, among others.

The rcmd( ) function looks up a host via the name server or, if the local name server is not running, via the /etc/hosts file. If the connection succeeds, a socket in the Internet domain of type SOCK_STREAM is returned to the calling process and given to the remote command as standard input (stdin) and standard output (stdout).

Always specify the host name. If the local domain and remote domain are the same, specifying the domain parts is optional.

Only processes with an effective user ID of root user can use the rcmd() function. An authentication scheme based on remote port numbers is used to verify permissions. Ports in the range from 0 to 1023 can only be used by a root user.

Return Values

Upon successful completion, the rcmd( ) function returns a valid socket descriptor. The function returns -1 if the effective user ID of the calling process is not root user or if the function fails to resolve the host.

Files

/etc/services Contains the service names, ports, and socket types.
/etc/hosts Contains hostnames and their addresses for hosts in a network.
/etc/resolv.conf Contains the name server and domain name.

Related Information

Functions: gethostname(2), rresvport(3), ruserok(3), sethostname(2)
Commands: rlogind(8), rshd(8), named(8)
re_comp, re_exec

Purpose Handles regular expressions

Library
Standard C Library (libc.a)
Berkeley Compatibility Library (libbsd.a)

Synopsis

```c
char *re_comp(
    char *string);

int re_exec(
    char *string);
```

Parameters

string Points to the string that is to be matched or converted.

Description

The `re_comp()` function converts a string into an internal form suitable for pattern matching. The `re_exec()` function compares the `string` parameter with the last string passed to the `re_comp()` function.

When the `re_comp()` function is passed a value of 0 (zero) or null, the regular expression currently being converted remains unchanged.

Strings passed to both the `re_comp()` and `re_exec()` functions may have trailing or embedded newline characters; however, these strings are terminated by a null. Recognized regular expressions are described in the reference page for the `regexp` functions (`advance()`, `compile()` and `step()`). Refer to that reference page when the differences described above are noted.

Return Values

The `re_comp()` function returns 0 (zero) when the string pointed to by the `string` parameter is successfully converted; otherwise an error message string is returned. The `re_exec()` function returns 0 (zero) when the string is recognized by the last
compiled regular expression and a value of +1 when the string pointed to by the
string parameter fails to match the last converted regular expression; the value -1 is
returned when the converted regular expression is invalid (indicating an internal
error).

Errors

Upon error, the re_exec( ) function returns a value of -1, and the re_comp( )
function returns a string indicating the nature of the error.

Related Information

Function: regexp(3)
read, readv

Purpose
Reads from a file

Synopsis
int read(
    int filedes,
    char *buffer,
    unsigned int nbytes);
#include <sys/types.h>
#include <sys/uio.h>
int readv(
    int filedes,
    struct iovec *iov,
    int iovcount);

Parameters

  filedes     Specifies a file descriptor identifying the object to be read.
  buffer     Points to the buffer to receive data read.
  nbytes     Specifies the number of bytes to read from the file associated with
            the filedes parameter.
  iov         Points to an array of iovec structures that identifies the buffers into
              which the data is to be placed.
  iovcount   Specifies the number of iovec structures pointed to by the iov
            parameter.

Description

The read() function attempts to read nbytes of data from the file associated with
the filedes parameter into the buffer pointed to by the buffer parameter. The
readv() function performs the same action as the read() function, but scatters the
input data into the buffers specified by the array of iovec structures pointed to by
the iov parameter.

On regular files and devices capable of seeking, the read() function starts at a
position in the file given by the file pointer associated with the filedes parameter.
Upon return from the read() function, the file pointer is incremented by the
number of bytes actually read.

Devices that are incapable of seeking always read from the current position. The
value of a file pointer associated with such a file is undefined.
The `read()` and `readv()` functions, which suspend the calling process until the request is completed, are redefined so that only the calling thread is suspended.

Upon successful completion, the `read()` function returns the number of bytes actually read and placed in the buffer. This number will never be greater than `nbytes`. The value returned may be less than `nbytes` if the number of bytes left in the file is less than `nbytes`, if the `read()` request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than `nbytes` bytes immediately available for reading. For example, a `read()` from a file associated with a terminal may return one typed line of data.

No data transfer will occur past the current End-of-File. If the starting position is at or after the End-of-File, 0 (zero) is returned.

If the value of `nbytes` is 0 (zero), the `read()` function will return 0 and have no other results.

When attempting to read from an empty pipe (or FIFO):

- If no process has the pipe open for writing, the `read()` function returns 0 (zero) to indicate End-of-File.
- If some process has the pipe open for writing:
  - If neither O_NONBLOCK nor O_NDELAY is set, the `read()` function will block until some data is written or the pipe is closed by all processes that had opened the pipe for writing.
  - If O_NONBLOCK or O_NDELAY is set, the `read()` function returns a value of -1 and sets `errno` to [EAGAIN].

When attempting to read from a character special file that supports nonblocking reads, such as a terminal, and no data is currently available:

- If neither O_NONBLOCK nor O_NDELAY is set, the `read()` function will block until data becomes available.
- If O_NONBLOCK or O_NDELAY is set, the `read()` functions return -1 and sets `errno` to [EAGAIN] if no data is available. The use of the O_NONBLOCK flag has no effect if there is some data available.

When attempting to read from a regular file with enforcement mode record locking enabled, and all or part of the region to be read is currently locked by another process (a write lock or exclusive lock):

- If O_NDELAY and O_NONBLOCK are clear, the `read()` function blocks the calling process until the lock is released, or `read()` is terminated by a signal.
- If O_NDELAY or O_NONBLOCK is set, the `read()` function returns -1 and sets `errno` to [EAGAIN].
If a `read()` function is interrupted by a signal before it reads any data, it will return -1 with `errno` set to [EINTR]. If a `read()` function is interrupted by a signal after it has successfully read some data, the behavior depends on how the handler for the arriving signal was installed.

If the handler was installed with an indication that functions should not be restarted, the `read()` function returns a value of -1 and `errno` is set to [EINTR] (even if some data was already consumed). If the handler was installed with an indication that functions should be restarted, and data had been read when the interrupt was handled, the `read()` function returns the amount of data consumed.

A `read()` from a pipe or FIFO will never return with `errno` set to [EINTR] if it has transferred any data.

For any portion of an ordinary file prior to the End-of-File that has not been written, the `read()` function returns bytes with value 0 (zero).

Upon successful completion, the `read()` function marks the `st_atime` field of the file for update.

The `readv()` function performs the same action as the `read()` function, but scatters the input data into the buffers specified by the array of `iovec` structures pointed to by the `iov` parameter. The `iovcount` parameter specifies the number of buffers pointed to by the `iov` parameter. Each `iovec` entry specifies the base address and length of an area in memory where data should be placed. The `readv()` function always fills an area completely before proceeding to the next.

The `iovec` structure is defined in the `sys/uio.h` header file and contains the following members:

```c
  caddr_t iov_base;
  int iov_len;
```

**Notes**

**AES Support Level:** Full use (`read()`)  

**Return Values**

Upon successful completion, the `read()` and `readv()` functions return the number of bytes actually read and placed into buffers. The system guarantees to read the number of bytes requested only if the descriptor references a normal file that has the same number of bytes left before the End-of-File. Otherwise, a value of -1 is returned, `errno` is set to indicate the error, and the content of the buffer pointed to by the `buffer` parameter is indeterminate.
Errors

If the `read()` or `readv()` function fails, `errno` may be set to one of the following values.

- **[EBADF]** The `filedes` parameter is not a valid file descriptor open for reading.
- **[EINVAL]** The file position pointer associated with the `filedes` parameter was negative.
- **[EINVAL]** The sum of the `iov_len` values in the `iov` array was negative or overflowed a 32-bit integer.
- **[EINVAL]** The value of the `iovcount` parameter was not between 1 and 16, inclusive.
- **[EAGAIN]** The O_NONBLOCK flag is set for the file descriptor and the process would be delayed in the read operation.
- **[EFAULT]** The `buffer` or part of the `iov` points to a location outside of the allocated address space of the process.
- **[EINTR]** A `read()` was interrupted by a signal before any data arrived, and the signal handler was installed with an indication that functions are not to be restarted.
- **[EIO]** The process is a member of a background process attempting to read from its controlling terminal, the process is ignoring or blocking the SIGTTIN signal, or the process group is orphaned.
- **[EAGAIN]** Enforced record locking is enabled, O_NDELAY or O_NONBLOCK is set, and there is a write lock owned by another process.
- **[ENOLCK]** The file has mandatory enforcement mode file locking set and LOCK_MAX regions are already locked in the system.
- **[EDEADLK]** Enforcement mode file locking is enabled, O_NDELAY and O_NONBLOCK are clear, and a deadlock condition is detected.

Related Information

Functions: `fcntl(2)`, `lockf(3)`, `lseek(2)`, `open(2)`, `pipe(2)`, `poll(2)`, `socket(2)`, `socketpair(2)`, `opendir(3)`
readlink

**Purpose**  Reads the value of a symbolic link

**Synopsis**

```c
#include <symlink.h>

int readlink (char *path, char *buffer, int buf_size);
```

**Parameters**

- `path` Specifies the pathname of the destination file or directory.
- `buffer` Points to the user's buffer. The buffer should be at least as large as the `buf_size` parameter.
- `buf_size` Specifies the size of the buffer.

**Description**

The `readlink()` function places the contents of the symbolic link named by the `path` parameter in `buffer`, which has size `buf_size`. If the actual length of the symbolic link is less than `buf_size`, the string copied into the buffer will be null-terminated. If the actual length of the symbolic link is greater than `buf_size`, an error will be returned. The length of a symbolic link will not exceed PATH_MAX.

For a `readlink()` function to complete successfully, the calling process must have search access to the directory containing the link.

**Notes**

- **AES Support Level:** Trial use
Return Values

Upon successful completion, the `readlink()` function returns a count of the number of characters placed in the buffer (not including any terminating null). If the `readlink()` function fails, the buffer is not modified, a value of -1 is returned, and `errno` is set to indicate the error.

Errors

If the `readlink()` function fails, `errno` may be set to one of the following values:

- [ENOENT] The file named by the `path` parameter does not exist or the `path` parameter points to an empty string.
- [EINVAL] The file named by the `path` parameter is not a symbolic link.
- [ERANGE] The pathname in the symbolic link is longer than `buf_size`.
- [ENOTDIR] A component of the path prefix of the `path` parameter is not a directory.
- [EACCES] Search permission is denied on a component of the path prefix of the `path` parameter, or read permission is denied on the final component of the path prefix of the `path` parameter.
- [ENAMETOOLONG] The length of the `path` parameter exceeds PATH_MAX, or a pathname component is longer than NAME_MAX.

Related Information

Functions: `link(2)`, `stat(2)`, `symlink(2)`, `unlink(2)`
reboot

Purpose
Reboots system or halts processor

Synopsis
#include <sys/reboot.h>

void reboot(
    int howto);

Parameters
howto Specifies a mask of options.

Description
The reboot() function restarts the system. The startup is automatic and brings up /vmunix in the normal, nonmaintenance mode. The calling process must have superuser privilege to run this function successfully. However, a reboot is invoked automatically in the event of unrecoverable system failures.

The following options, defined in the sys/reboot.h include file are passed to the new kernel or the new bootstrap and init programs. They are supplied as values to the howto parameter.

RB_AUTOBOOT
The default, causing the system to reboot in its usual fashion.

RB_ASKNAME
Interpreted by the bootstrap program itself, causing it to prompt on the console as to what file should be booted.

RB_DFLTROOT
Use the compiled-in root device. If possible, the system uses the device from which it was booted as the root device. (The default behavior is dependent on the ability of the bootstrap program to determine the drive from which it was loaded, which is not possible on all systems.)

RB_DUMP
Dump kernel memory before rebooting; see the savecore command for more information.

RB_HALT
The processor is simply halted; no reboot takes place. This option should be used with caution.
RB_INITNAME
Allows the specification of an init program (see the init program) other than /sbin/init to be run when the system reboots. This switch is not currently available.

RB_KDB
Load the symbol table and enable a built-in debugger in the system. This option has no useful function if the kernel is not configured for debugging. Several other options have different meanings if combined with this option, although their use may not be possible via the reboot() function.

RB_NOSYNC
Normally, the disks are sync’d (see the sync() command) before the processor is halted or rebooted.

RB_RDONLY
Initially mount the root file system read-only. This is currently the default, and this option has been deprecated as a no-op.

RB_SINGLE
Normally, the reboot procedure involves an automatic disk consistency check and then multiuser operations. RB_SINGLE prevents this, booting the system with a single-user shell on the console. RB_SINGLE is actually interpreted by the init program in the newly booted system.

RB_UNIPROC
Restart the system in uniprocessor mode.

When no options are given (that is, RB_AUTOBOOT is used), the system is rebooted from file vmunix in the root file system of unit 0 (zero) of a disk chosen in a processor-specific way. An automatic consistency check of the disks is then normally performed (see the fsck command).

Some options may not be supported on all machines.

Return Values
If successful, this call does not return. Otherwise, a -1 is returned and errno is set to indicate the error.
OSF/1 Programmer's Reference

reboot(2)

Errors

If the reboot() function fails, errno may be set to the following value:

[EPERM] The calling process does not have appropriate privilege.

Related Information

Commands: crash(8), halt(8), init(8), reboot(8), savecore(8)
recv

Purpose Receives messages from connected sockets

Synopsis
```
#include <sys/types.h>
#include <sys/socket.h>

int recv (  
    int socket,  
    char *buffer,  
    int length,  
    int flags);
```

Parameters
- socket Specifies the socket descriptor.
- buffer Points to an address where the message should be placed.
- length Specifies the size of the address pointed to by the buffer parameter.
- flags Points to a value controlling the message reception. The flags parameter is formed by logically ORing one or more of the following values, defined in the sys/socket.h file:
  - MSG_PEEK Peek at incoming message. The data is treated as unread and the next recv() function (or similar function) will still return this data.
  - MSG_OOB Process out-of-band data.

Description
The recv() function receives messages from a connected socket. The recvfrom() and recvmsg() functions receive messages from both connected and unconnected sockets; however, they are usually used for unconnected sockets only.

The recv() function returns the length of the message. If a message is too long to fit in the supplied buffer, excess bytes may be truncated depending on the type of socket that issued the message.
If no messages are available at the socket, the `recv()` function waits for a message to arrive, unless the socket is nonblocking. If a socket is nonblocking, `errno` is set to `[EWOULDBLOCK]`. Use the `select()` function to determine when more data arrives.

**Notes**

The `recv()` function is identical to the `recvfrom()` function with a zero-valued `address_len` parameter, and to the `read()` function if no flags are used. For that reason, the `recv()` function is disabled when 4.4BSD behavior is enabled (that is, when the `_SOCKADDR_LEN` compile-time option is defined).

**Return Values**

Upon successful completion, the `recv()` function returns the length of the message in bytes. Otherwise, a value of -1 is returned, and `errno` is set to indicate the error.

**Errors**

If the `recv()` function fails, `errno` may be set to one of the following values:

- `[EBADF]` The `socket` parameter is not valid.
- `[ENOTSOCK]` The `socket` parameter refers to a file, not a socket.
- `[EWOULDBLOCK]` The socket is marked nonblocking, and no data is waiting to be received.
- `[EINTR]` A signal interrupted the `recv()` function before any data was available.
- `[EFAULT]` The data was directed to be received into a nonexistent or protected part of the process address space. The `buffer` parameter is invalid.

**Related Information**

Functions: `recvfrom(2)`, `recvmsg(2)`, `send(2)`, `sendmsg(2)`, `sendto(2)`, `select(2)`, `shutdown(2)`, `socket(2)`, `read(2)`, `write(2)`
recvfrom

Purpose
Receives messages from sockets

Synopsis
#include <sys/types.h>
#include <sys/socket.h>

int recvfrom(
    int socket,
    char *buffer,
    int length,
    int flags,
    struct sockaddr *address,
    int *address_len);

Parameters

socket
Specifies the socket file descriptor.

buffer
Specifies a pointer to the buffer to which the message should be written.

length
Specifies the length in bytes of the buffer pointed to by the buffer parameter.

flags
Points to a value that controls message reception. The parameter to control message reception is formed by the logical OR of one or more of the following values:

MSG_PEEK
Peeks at the incoming message.

MSG_OOB
Processes out-of-band data.

address
Points to a sockaddr structure, the format of which is determined by the domain and by the behavior requested for the socket. The sockaddr structure is an overlay for a sockaddr_in, sockaddr_un, or sockaddr_ns structure, depending on which of the supported address families is active. If the compile-time option _SOCKADDR_LEN is defined before the sys/socket.h header file is
recvfrom(2)

included, the sockaddr structure takes 4.4BSD behavior, with a field for specifying the length of the socket address. Otherwise, the default 4.3BSD sockaddr structure is used, with the length of the socket address assumed to be 14 bytes or less.

If _SOCKADDR_LEN is defined, the 4.3BSD sockaddr structure is defined with the name osockaddr.

address_len
 specifies the length of the sockaddr structure pointed to by the address parameter.

Description

The recvfrom( ) function permits an application program to receive messages from unconnected sockets. It is normally applied to unconnected sockets because it includes parameters that permit a calling program to retrieve the source endpoint of received data.

To obtain the source address of the message, specify a nonzero value for the address parameter. The recvfrom() function is called with the address_len parameter set to the size of the buffer specified by the address parameter. On return, this function modifies the address_len parameter to the actual size in bytes of the address specified by the address parameter. The recvfrom() function returns the length of the message written to the buffer pointed to by the buffer parameter.

When a message is too long for the specified buffer, excess bytes may be truncated depending on the type of socket that issued the message, and depending on which flags are set with the flags parameter.

When no message is available at the socket specified by the socket parameter, the recvfrom() function waits for a message to arrive, unless the socket is nonblocking. When the socket is nonblocking, errno is set to [EWOULDBLOCK].

Return Values

Upon successful completion, the byte length of the written message is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the recvfrom() function fails, errno may be set to one of the following values:

[EBADF] The socket parameter is not a valid file descriptor.
[ENOTSOCK] The socket parameter refers to a file, not a socket.
Functions
recvfrom(2)

[EWOULDBLOCK]
The socket is nonblocking; no data is ready to be received.

[EFAULT]  A valid message buffer was not specified. Nonexistent or protected
address space is specified for the message buffer.

Related Information
Functions: recv(2), recvmsg(2), send(2), sendmsg(2), sendto(2), select(2),
shutdown(2), socket(2), read(2), write(2)
recvmsg

Purpose
Receives a message from a socket

Synopsis
#include <sys/types.h>
#include <sys/socket.h>

int recvmsg(
    int socket,
    struct msghdr *message,
    int flags );

Parameters

socket
Specifies a unique name of the socket.

message
Points to a msghdr structure, containing both the address for the incoming message and the buffers for the source address. The format of the address is determined by the behavior requested for the socket. If the compile-time option _SOCKADDR_LEN is defined before the sys/socket.h header file is included, the msghdr structure takes 4.4BSD behavior. Otherwise, the default 4.3BSD msghdr structure is used.

In 4.4BSD, the msghdr structure has a separate msg_flags field for holding flags from the received message. In addition, the msg_accrights field is generalized into a msg_control field. See the DESCRIPTION section for more information.

If _SOCKADDR_LEN is defined, the 4.3BSD msghdr structure is defined with the name omsghdr.

flags
Permits the caller of this function to exercise control over the reception of messages. The value for this parameter is formed by a logical OR of one or both of the following values:

MSG_PEEK
Peeks at the incoming message.

MSG_OOB
Processes out-of-band data.
Description

The `recvmsg()` function receives messages from unconnected or connected sockets and returns the length of the message. When a message is too long for the buffer, the message may be truncated depending on the type of socket from which the message is written.

When no messages are available at the socket specified by the `socket` parameter, the `recvmsg()` function waits for a message to arrive. When the socket is nonblocking and no message is available, the `recvmsg()` function fails and sets `errno` to [EWOULDBLOCK].

Use the `select()` function to determine when more data arrives.

In the `msghdr` structure, the `msg_name` and `msg_name_len` fields specify the destination address if the socket is unconnected. The `msg_name` field may be given as a null pointer if no names are desired or required. The `msg_iov` and `msg_iovlen` fields describe the scatter gather locations.

In 4.3BSD, the `msg_accrights` field is a buffer for passing access rights. In 4.4BSD, the `msg_accrights` field has been expanded into a `msg_control` field, to include other protocol control messages or other miscellaneous ancillary data.

In the 4.4BSD `msghdr` structure, the `msg_flags` field holds flags from the received message. In addition to `MSG_PEEK` and `MSG_OOB`, the incoming flags reported in the `msg_flags` field can be any of the following values:

- `MSG_EOR` Data includes the end-of-record marker.
- `MSG_TRUNC` Data was truncated before delivery.
- `MSG_CTRUNC` Control data was truncated before delivery.

Return Values

Upon successful completion, the `recvmsg()` function returns the length of the message in bytes, and fills in the fields of the `msghdr` structure pointed to by the `message` parameter as appropriate. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `recvmsg()` function fails, `errno` may be set to one of the following values:

- `[EBADF]` The `socket` parameter is not valid.
- `[ENOTSOCK]` The `socket` parameter refers to a file, not a socket.
recvmsg(2)

[EWOULDBLOCK]  The socket is marked nonblocking and no data is ready to be received.

[EINTR]  This function was interrupted by a signal before any data was available.

[EFAULT]  The message parameter is not in a readable or writable part of user address-space.

Related Information
Functions: recv(2), recvfrom(2), send(2), sendmsg(2), sendto(2), select(2), shutdown(2), socket(2)
Functions
regexp(3)

advance, compile, step

Purpose
Regular-expression compile and match routines

Synopsis
#define INIT declarations
#define GETC getc code
#define PEEK peek code
#define UNGETC(c) ungetc code
#define RETURN(ptr) return code
#define ERROR(val) error code

#include <regexp.h>

char *compile(
    char *instring,
    char *expbuf,
    char *endbuf,
    int_eof);

int step(
    char *string,
    char *expbuf);

int advance(
    char *string,
    char *expbuf);

extern char *loc1, *loc2, *locs;

Parameters

instring Specifies a string to be passed to the compile() function. The instring parameter is never used explicitly by the compile() function, but you can use it in your macros. For example, you may want to pass the string containing a pattern as the instring parameter to the compile() function and use the INIT() macro to set a pointer to the beginning of this string. When your macros do not use instring, call the compile() function with a value of ((char *) 0) for this parameter.

expbuf Points to a character array where the compiled regular expression is stored.
**regexp(3)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>endbuf</code></td>
<td>Points to the location that immediately follows the character array where the compiled regular expression is stored. When the compiled expression cannot be contained in ( \text{endbuf-expbuf} ) number of bytes, a call to the \textbf{ERROR(50)} macro is made.</td>
</tr>
<tr>
<td><code>eof</code></td>
<td>Specifies the character that marks the end of the regular expression. For example, in ed this character is usually / (slash).</td>
</tr>
<tr>
<td><code>string</code></td>
<td>Points to a null-terminated string of characters in the \textbf{step()} function, to be searched for a match.</td>
</tr>
</tbody>
</table>

**Description**

The \textbf{compile()}, \textbf{advance()}, and \textbf{step()} functions are used for general-purpose expression-matching.

The \textbf{compile()} function takes a simple regular expression as input and produces a compiled expression that can be used with the \textbf{step()} and \textbf{advance()} functions.

The following six macros, used in the \textbf{compile()} function, must be defined before the \#include `<regexp.h>` statement in programs. The \textbf{GETC()}, \textbf{PEEKC()}, and \textbf{UNGETC()} macros operate on the regular expression provided as input for the \textbf{compile()} function.

**INIT()**

The \textbf{INIT()} macro is used for dependent \textit{declarations} and initializations. In the \texttt{regexp.h} header file this macro is located right after the \textbf{compile()} function declarations and opening \{ (left brace). Your \textbf{INIT()} \textit{declarations} must end with a ; (semicolon).

The \textbf{INIT()} macro is frequently used to set a register variable to point to the beginning of the regular expression so that this pointer can be used in declarations for \textbf{GETC()}, \textbf{PEEKC()}, and \textbf{UNGETC()}. Alternatively, you can use \textbf{INIT()} to declare external variables that \textbf{GETC()}, \textbf{PEEKC()}, and \textbf{UNGETC()} need.

**GETC()**

The \textbf{GETC()} macro returns the value of the next character (byte) in the regular-expression pattern. Successive calls to \textbf{GETC()} return successive characters of the regular expression.

**PEEKC()**

The \textbf{PEEKC()} macro returns the next character (byte) in the regular expression. Immediate subsequent calls to this macro return the same byte, which is also the next character returned by the \textbf{GETC()} macro.
UNGETC(c) The UNGETC() macro causes the c parameter to be returned by the next call to the GETC() and PEEKC() macros. No more than one character of pushback is ever needed because this character is guaranteed to be the last character read by the GETC() macro. The value of the UNGETC() macro is always ignored.

RETURN(ptr)
The RETURN() macro is used for normal exit of the compile() function. The value of the ptr parameter is a pointer to the character following the last character of the compiled regular expression. This is useful in programs that manage memory allocation.

ERROR(val) The ERROR() macro is the abnormal return from the compile() function. A call to this macro should never return a value. In this macro, val is an error number, which is described in the ERRORS section of this reference page.

The step() function finds the first substring of the string parameter that matches the compiled expression pointed to by the expbuf parameter. When there is no match, the step() function returns 0 (zero). When there is a match, the step() function returns a nonzero value and sets two global character pointers: loc1, which points to the first character of the substring that matches the pattern, and loc2, which points to the character immediately following the substring that matches the pattern. When the regular expression matches the entire expression, loc1 points to the first character of the string parameter and loc2 points to the null character at the end of the expression specified by the string parameter.

The step() function uses the integer variable circf, which is set by the compile() function when the regular expression begins with a ^ (circumflex). When this variable is set, the step() function only tries to match the regular expression to the beginning of the string. When you compile more than one regular expression before executing the first one, save the value of circf for each compiled expression and set circf to the saved value before each call to step().

The advance() function tests whether an initial substring of the string parameter matches the expression pointed to by the expbuf parameter. Using the same parameters that were passed to it, the step() function calls the advance() function. The step() function increments a pointer through the string parameter characters and calls advance() until a nonzero value, which indicates a match, is returned, or until the end of the expression pointed to by the string parameter is reached. To unconditionally constrain string to point to the beginning of the expression, call the advance() function directly instead of calling step().
When the `advance()` function encounters an * (asterisk) or a \{\} sequence in the regular expression, it advances its pointer to the string to be matched as far as possible and recursively calls itself, trying to match the remainder of the regular expression. As long as there is no match, the `advance()` function backs up along the string until it finds a match or reaches the point in the string where the initial match with the * or \{\} character occurred.

It is sometimes desirable to stop this backing-up before the initial pointer position in the string is reached. When the `locs` global character pointer is matched with the character at the pointer position in the string during the backing-up process, the `advance()` function breaks out of the recursive loop that backs up and returns the value 0 (zero).

Example

The following is an example of the regular expression macros and calls from the `grep` command:

```c
#define INIT register char *sp=instring;
#define GETC() (*sp++)
#define PEEKC() (*sp)
#define UNGETC(c) (--sp)
#define RETURN(c) return;
#define ERROR(c) return;
#include <regexp.h>

... compile (patstr, expbuf, &expbuf[ESIZE], ' ');
... if (step (linebuf, expbuf))
    succeed ( );
... 
```

Notes

Two versions of these functions are available. The first, for XPG3 applications, supports simple internationalized expressions. The second, for System V applications, supports simple (non-internationalized) regular expressions.

BSD applications use different functions for regular expression handling. See the `re_comp()` and `re_exec()` functions.

AES Support Level: Trial use
Return Values

Upon successful completion, the compile() function calls the RETURN() macro. Upon failure, this function calls the ERROR() macro.

Whenever a successful match occurs, the step() and advance() functions return a nonzero value. Upon failure, these functions return 0 (zero).

Errors

If the compile() function fails, the ERROR() macro is called with an error number as its argument. The possible error numbers are:

<table>
<thead>
<tr>
<th>Error</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Range endpoint too large</td>
</tr>
<tr>
<td>16</td>
<td>Bad number</td>
</tr>
<tr>
<td>25</td>
<td>\digit out of range</td>
</tr>
<tr>
<td>36</td>
<td>Illegal or missing delimiter</td>
</tr>
<tr>
<td>41</td>
<td>No remembered search string</td>
</tr>
<tr>
<td>42</td>
<td>There is a ( ) pair imbalance</td>
</tr>
<tr>
<td>43</td>
<td>Too many ( ) pairs (maximum is 9)</td>
</tr>
<tr>
<td>44</td>
<td>More than two numbers given in the {} pair</td>
</tr>
<tr>
<td>45</td>
<td>A } character expected after \</td>
</tr>
<tr>
<td>46</td>
<td>First number exceeds second in the {} pair</td>
</tr>
<tr>
<td>49</td>
<td>There is a [ ] pair imbalance</td>
</tr>
<tr>
<td>50</td>
<td>Regular expression overflow</td>
</tr>
<tr>
<td>70</td>
<td>Invalid endpoint in range expression</td>
</tr>
</tbody>
</table>

Related Information

Functions: ctype(3), re_comp(3)
Commands: ed(1), sed(1), grep(1)
reltimer

Purpose  Establishes timeout intervals of a per-process timer

Library  Standard C Library (libc.a)

Synopsis  

```c
#include <sys/timers.h>

int reltimer(
    timer_t tmrid,
    struct itimerspec *val,
    struct itimerspec *oval);
```

Parameters

- **tmrid**  Specifies the per-process timer to access.
- **val**  Points to a type `itimerspec` structure containing the values of the initial and offset timeout intervals.
- **oval**  Points to a type `itimerspec` structure where the current value of the timer timeout interval and the time-to-go are to be stored.

Description

The `reltimer()` function establishes initial and offset timeout intervals of a per-process timer specified by the `tmrid` parameter. Initial and offset timeout interval information is stored in an `itimerspec` structure pointed to by the `val` parameter. When the per-process timer specified by the `tmrid` parameter is active, after timeout of the initial time interval, all subsequent timeouts are controlled by the offset timeout value; as long as `tmrid` continues to operate, the offset values pointed to by the `val` parameter are used as the per-process timeout interval. The current timeout interval and the time-to-go are returned to the location pointed to by the `oval` parameter.

Initial and offset time information for the per-process timer is stored in space reserved by a type `itimerspec` structure pointed to by the `val` parameter. A type `itimerspec` structure is also used to store returned time information specified by the `oval` parameter. The `itimerspec` structure is defined in the `sys/timers.h` include file.
Notes

Time values smaller than the resolution of the specified timer are rounded up to the
resolution value. Time values larger than the maximum timeout value of the
specified per-process timer are rounded down to that maximum value.

AES Support Level: Trial use

Return Values

Upon successful completion, the `reltimer()` function returns 0 (zero). Otherwise,
-1 is returned and `errno` is set to indicate the error.

Errors

If the `reltimer()` function fails, `errno` may be set to the following value:

[EINVAL] The `timerid` parameter does not specify an allocated per-process
timer, or the `val` parameter points to a nanosecond value less than
zero or greater than or equal to 1000 million.

Related Information

Functions: `alarm(3), getclock(3), gettimer(3), mktimer(3)`
remove

Purpose
Removes a file

Library
Standard C Library (libc.a)

Synopsis
#include <stdio.h>

int remove(
    const char *file_name);

Parameters

file_name  Points to the file to be removed.

Description
The remove( ) function causes a file named by the string pointed to by file_name to be no longer accessible by that name. A subsequent attempt to open that file using that name will fail unless it is created anew.

If the file_name parameter is called on a directory, it is equivalent to calling the rmdir() function on that directory.

Notes
If the file operated upon by the remove() function has multiple links, the link count in the file is decremented.

AES Support Level: Full use

Return Values
Upon successful completion, the remove() function returns 0 (zero). Otherwise, a nonzero value is returned.
Errors

Refer to the `unlink()` function and the `rmdir()` function for information on error conditions.

Related Information

Functions: `link(2)`, `rename(2)`

Commands: `link(1)`, `unlink(1)`
rename

Purpose
Renames a directory or a file within a file system

Synopsis
#include <stdio.h>

int rename
  (char *from,
   char *to);

Parameters
from Identifies the file or directory to be renamed.
to Identifies the new pathname of the file or directory to be renamed. If the to parameter is an existing file or empty directory, it is replaced by the from parameter. If the to parameter is a nonempty directory, the rename() function exits with an error.

Description
The rename() function renames a directory or a file within a file system.

For rename() to complete successfully, the calling process must have write and search permission to the parent directories of both the from and to parameters. If the from parameter is a directory and the parent directories of from and to are different, then the calling process must have write and search permission to the from parameter as well.

If the from and to parameters both refer to the same existing file, the rename() function returns successfully and performs no other action.

Both the from and to parameters must be of the same type (that is, both directories or both nondirectories) and must reside on the same file system. If the to parameter already exists, it is first removed. In this case it is guaranteed that a link named the to parameter will exist throughout the operation. This link refers to the file named by either the to or from parameter before the operation began.

If the final component of the from parameter is a symbolic link, the symbolic link (not the file or directory to which it points) is renamed. If the final component of the to parameter is a symbolic link, the symbolic link is destroyed.
If the `from` and `to` parameters name directories, the following must be true:

- The `from` parameter is not an ancestor of the `to` parameter. For example, the `to` pathname must not contain a path prefix that names `from`.
- The `from` parameter is well-formed. For example, the `. (dot) entry in `from`, if it exists, refers to the same directory as `from`, exactly one directory has a link to `from` (excluding the self-referential `. `), and the `.. (dot-dot) entry in `from`, if it exists, refers to the directory that contains an entry for `from`.
- The `to` parameter, if it exists, must be well-formed (as defined previously).

Upon successful completion, the `rename()` function marks the `st_ctime` and `st_mtime` fields of the parent directory of each file for update.

**Notes**

**AES Support Level:** Full use

**Return Values**

Upon successful completion, the `rename()` function returns a value of 0 (zero). Otherwise, a value of -1 is returned, and `errno` is set to indicate the error.

**Errors**

If the `rename()` function fails, the file or directory name remains unchanged and `errno` may be set to one of the following values:

- `[ENOTDIR]` The `from` parameter names a directory and the `to` parameter names a nondirectory.
- `[EISDIR]` The `to` parameter names a directory and the `from` parameter names a nondirectory.
- `[ENOENT]` A component of either path does not exist, or either path is the empty string, or the file named by the `from` parameter does not exist.
- `[EACCES]` Creating the requested link requires writing in a directory with a mode that denies write permission, or a component of either pathname denies search permission.
- `[EXDEV]` The link named by the `to` parameter and the file named by the `from` parameter are on different file systems.
- `[EBUSY]` The directory named by the `from` or `to` parameter is currently in use by the system or by another process.
rename(2)

- **[EINVAL]** Either the *from* or *to* parameter is not a well-formed directory, an attempt is made to rename . (dot) or .. (dot-dot), or the *from* parameter is an ancestor of the *to* parameter.
- **[EROFS]** The requested operation requires writing in a directory on a read-only file system.
- **[EEXIST]** The *to* parameter is an existing nonempty directory.
- **[ENOSPC]** The directory that would contain *to* cannot be extended because the file system is out of space.
- **[EDQUOT]** The directory that would contain *to* cannot be extended because the user’s quota of disk blocks on the file system containing the directory is exhausted.
- **[EFAULT]** Either the *to* or *from* parameter is an invalid address.
- **[ELOOP]** Too many links were encountered in translating either *to* or *from*.
- **[ENAMETOOLONG]** The length of the *to* or *from* parameters exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- **[EPERM]** The S_ISVTX flag is set on the directory containing the file to be renamed, and the caller is not the file owner.

### Related Information

**Functions:** chmod(2), link(2), mkdir(2), rmdir(2), unlink(2)

**Commands:** chmod(1), mkdir(1), mv(1), mvdir(1)
res_init

Purpose  Searches for a default domain name and Internet address

Library  Standard C Library (libc.a)

Synopsis  
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
void res_init ( void );

Description  
The res_init() function reads the /etc/resolv.conf file for the default domain name and the Internet address of the initial hosts running the name server, even if the name server is not functioning.

The res_init() function is one of a set of subroutines that form the resolver, a set of functions that resolve domain names. All resolver functions use the /usr/include/resolv.h header file, which defines the _res data structure. The res_init() function stores domain name information in the _res data structure.

Notes  
If the /etc/resolv.conf file does not exist, the res_init() function attempts name resolution using the local /etc/hosts file. If the system is not using a domain name server, the /etc/resolv.conf file should not exist. The /etc/host file should be present on the system even if the system is using a name server. In this instance, the file should contain the host IDs that the system requires to function even if the name server is not functioning.

Files  
/etc/resolv.conf  
Contains the name server and domain name.

/etc/hosts  
Contains hostnames and their addresses for hosts in a network. This file is used to resolve a hostname into an Internet address.
res_init(3)

Related Information

Functions: dn_comp(3), dn_expand(3), dn_find(3), dn_skipname(3),
getlong(3), getshort(3), putlong(3), putshort(3), res_mkquery(3), res_send(3)
res_mkquery

Purpose  Makes query messages for name servers

Library  Standard C Library (libc.a)

Synopsis  
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

int res_mkquery(
    int query_type,
    char *domain_name,
    int class,
    int type,
    char *data,
    int data_length,
    struct rrec *reserved,
    char *buffer,
    int buf_length);

Parameters

query_type  Specifies a query type. The usual type is QUERY, but the parameter can be set to any of the query types defined in the arpa/nameser.h file.

domain_name  Points to the name of the domain. If the domain_name parameter points to a single label and the RES_DEFNAMES bit is set, as it is by default, the function appends domain_name to the current domain name. The current domain name is defined by the name server in use or in the /etc/resolv.conf file.

class  Specifies one of the following parameters:
    C_IN  Specifies the ARPA Internet.
    C_CHAOS  Specifies the Chaos network at MIT.
**res_mkquery(3)**

**type**  
Requires one of the following values:

- **T_A**  
  Host address
- **T_NS**  
  Authoritative server
- **T_MD**  
  Mail destination
- **T_MF**  
  Mail forwarder
- **T_CNAME**  
  Canonical name
- **T_SOA**  
  Start of authority zone
- **T_MB**  
  Mailbox domain name
- **T_MG**  
  Mail group member
- **T_MR**  
  Mail rename name
- **T_NULL**  
  NULL resource record
- **T_WKS**  
  Well known service
- **T_PTR**  
  Domain name pointer
- **T_HINFO**  
  Host information
- **T_MINFO**  
  Mailbox information
- **T_MX**  
  Mail routing information
- **T_UINFO**  
  User (**finger**) information
- **T_UID**  
  User ID
- **T_GID**  
  Group ID

**data**  
Points to the data that is sent to the name server as a search key. The data is stored as a character array.

**data_length**  
Defines the size of the array pointed to by the **data** parameter.

**reserved**  
Specifies a reserved and currently unused parameter.

**buffer**  
Points to a location containing the query message.

**buf_length**  
Specifies the length of the message pointed to by the **buffer** parameter.
Description

The res_mkquery() function makes packets for name servers in the Internet domain. The res_mkquery() function makes a standard query message and places it in the location pointed to by the buffer parameter.

The res_mkquery() function is one of a set of subroutines that form the resolver, a set of functions that resolve domain names. Global information that is used by the resolver functions is kept in the _res data structure. The /include/resolv.h file contains the _res data structure definition.

Return Values

Upon successful completion, the res_mkquery() function returns the size of the query. If the query is larger than the value of the buf_length parameter, the function fails and returns a value of -1.

Files

/etc/resolv.conf
Contains the name server and domain name.

Related Information

res_send

Purpose
Sends a query to a name server and retrieves a response

Library
Standard C Library (libc.a)

Synopsis
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

int res_send (char *msg_ptr, int msg_len, char *answer, int ans_len);

Parameters

msg_ptr Points to the beginning of a message.
msg_len Specifies the length of the message.
answer Points to an address where the response is stored.
ans_len Specifies the size of the answer area.

Description
The res_send() function sends a query to name servers and calls the res_init() function if the RES_INIT option of the _res data structure is not set. This function sends the query to the local name server and handles timeouts and retries.

The res_send() function is one of a set of subroutines that form the resolver, a set of functions that resolve domain names. Global information that is used by the resolver functions is kept in the _res data structure. The /include/resolv.h file contains the _res data structure definition.
Return Values

Upon successful completion, the res_send() function returns the length of the message. Otherwise, -1 is returned.

Files

/etc/resolv.conf
Contains general name server and domain name information.

Related Information

Purpose
Allows command execution on a remote host

Library
Standard C Library (libc.a)

Synopsis
```
int rexec(  
  char **host,  
  int port,  
  char *user,  
  char *passwd,  
  char *command,  
  int *err_file_desc );
```

Parameters

- **host**
  Contains the name of a remote host that is listed in the /etc/hosts file or /etc/resolv.conf file. If the name of the host is not found in either file, the reexec( ) fails.

- **port**
  Specifies the well-known DARPA Internet port to use for the connection. A pointer to the structure that contains the necessary port can be obtained by issuing the following library call: 
  `getservbyname( ) (exec, tcp)`

- **user**
  Points to a user ID valid at the host.

- **passwd**
  Points to the password of the specified user ID on the host.

- **command**
  Points to the name of the command to be executed at the remote host.

- **err_file_desc**
  Specifies the file to which standard error from the remote command is sent.

If the **err_file_desc** parameter is 0 (zero), the standard error of the remote command is the same as standard output. No provision is made for sending arbitrary signals to the remote process. In this case, however, it may be possible to send out-of-band data to the remote command.
If the `err_file_desc` parameter is nonzero, an auxiliary channel to a control process is set up, and a descriptor for it is placed in the `err_file_desc` parameter. The control process provides diagnostic output from the remote command on this channel and also accepts bytes as signal numbers to be forwarded to the process group of the command. This diagnostic information does not include remote authorization failure, since this connection is set up after authorization has been verified.

**Description**

The `rexec()` (remote execution) function allows the calling process to execute commands on a remote host.

If the `rexec()` connection succeeds, a socket in the Internet domain of type SOCK_STREAM is returned to the calling process and is given to the remote command as standard input and standard output.

The `user` and `passwd` parameters specify a valid user ID and the associated password for that user on the remote host. If the `user` and `passwd` parameters are not supplied, the `rexec()` function takes the following actions until finding a user ID and password to send to the remote host:

1. Searches the current environment for the user ID and password on the remote host.
2. Searches the tser’s home directory for a file called `~/.netrc` that contains a user ID and password.
3. Prompts the user for a user ID and password.

**Return Values**

Upon successful completion, the system returns a socket to the remote command. Otherwise, `-1` is returned, indicating that the specified hostname does not exist.
Files

/etc/hosts  Contains hostnames and their addresses for hosts in a network. This file is used to resolve a hostname into an Internet address.

/etc/resolv.conf  Contains the name server and domain name.

$HOME/.netrc  Contains automatic login information.

Related Information

Functions: `getservbyname(3)`, `rcmd(3)`, `rresvport(3)`, `ruserok(3)`
Commands: `rexecd(8)`
rmdir

Purpose
Removes a directory file

Synopsis
```c
int rmdir( 
    const char *path);
```

Parameters
- `path`: Specifies the directory pathname. The final component of the `path` parameter cannot be a symbolic link.

Description
The `rmdir()` function removes the directory specified by the `path` parameter. The directory is removed only if it is an empty directory.

For the `rmdir()` function to execute successfully, the calling process must have write access to the parent directory of the `path` parameter.

If the directory's link count becomes 0 (zero) and no process has the directory open, the space occupied by the directory is freed and the directory is no longer accessible. If one or more processes have the directory open when the last link is removed, the `. (dot) and .. (dot-dot) entries, if present, are removed before the `rmdir()` function returns, and no new entries may be created in the directory. However, the directory is not removed until all references to the directory have been closed.

Upon successful completion, the `rmdir()` function marks the `st_ctime` and `st_mtime` fields of the parent directory for update.

Notes

 AES Support Level: Full use

Return Values
Upon successful completion, the `rmdir()` function returns a value of 0 (zero). If the `rmdir()` function fails, a value of -1 is returned and `errno` is set to indicate the error.
Errors

If the `rmdir()` function fails, the directory is not deleted and `errno` may be set to one of the following values:

- **[EBUSY]** The directory is in use as either the mount point for a file system or the current directory of the process that issued the `rmdir()` function.
- **[EEXIST]** The directory named by the `path` parameter is not empty.
- **[ENOENT]** The directory named by the `path` parameter does not exist or is an empty string.
- **[EROFS]** The directory named by the `path` parameter resides on a read-only file system.
- **[EACCES]** Search permission is denied on a component of the `path` parameter, or write permission is denied on the parent directory of the directory to be removed.
- **[EFAULT]** The `path` parameter is an invalid address.
- **[ELOOP]** Too many links were encountered in translating `path`.
- **[ENAMETOOLONG]** The length of the `path` parameter exceeds PATH_MAX, or a pathname component is longer than NAME_MAX.
- **[EPERM]** The S_ISVTX flag is set on the parent directory of the directory to be removed, and the caller is not the file owner.
- **[ENOTDIR]** A component of the `path` parameter is not a directory.

Related Information

Functions: `chmod(2), mkdir(2), mknod(2), mkfifo(3), remove(3), rename(2), umask(2), unlink(2)`
rmtimer

Purpose Frees a per-process timer

Library Standard C Library (libc.a)

Synopsis #include <sys/timers.h>

int rmtimer(
    timer_t timer_id);

Parameters

   timer_id Specifies the per-process timer to deallocate.

Description

The rmtimer() function is used to free a previously allocated per-process timer
(previously returned by the mktimer() function). Any pending per-process timer
event generated by the timer specified by the timer_id parameter is cancelled when
this function successfully executes.

Notes

AES Support Level: Trial use

Return Values

Upon successful completion, the value 0 (zero) is returned. Otherwise, the value -1
is returned and errno is set to indicate the error.

Errors

If the rmtimer() function fails, errno may be set to the following value:

[EINVAL] The timerid parameter does not specify an allocated per-process
timer.

Related Information

Functions: gettimer(3), mktimer(3), reltimer(3)
rresvport

Purpose
Retrieves a socket with a privileged address

Library
Standard C Library (libc.a)

Synopsis
int rresvport(
    int *port);

Parameters
port Specifies the port to use for the connection.

Description
The rresvport() function obtains a socket with a privileged address bound to the socket. A privileged Internet port is one that falls in the range of 0 to 1023.

Only processes with an effective user ID of root can use the rresvport() function. An authentication scheme based on remote port numbers is used to verify permissions.

If the connection succeeds, a socket in the Internet domain of type SOCK_STREAM is returned to the calling process.

Return Values
Upon successful completion, the rresvport() function returns a valid, bound socket descriptor. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors
If the rresvport() function fails, errno may be set to one of the following values:
[EAGAIN] All network ports are in use.
[EAFNOSUPPORT]
The addresses in the specified address family cannot be used with this socket.
Functions

**rresvport(3)**

[EMFILE] Two hundred (200) file descriptors are currently open.

[ENFILE] The system file table is full.

ENOBUFS Insufficient buffers are available in the system to complete the function.

**Files**

/etc/services Contains the service names.

**Related Information**

Functions: **rcmd(3)**, **ruserok(3)**
ruserok

Purpose
Allows servers to authenticate clients

Library
Standard C Library (libc.a)

Synopsis
```c
int ruserok ( 
    char *host,
    int root_user,
    char *remote_user,
    char *local_user);
```

Parameters
- **host**
  Specifies the name of a remote host.

- **root_user**
  Specifies a value to indicate whether the effective user ID of the calling process is that of a root user. A value of 0 (zero) indicates the process does not have a root user ID. A value of 1 indicates that the process has local root user privileges, and the `/etc/host.equiv` file is not checked.

- **remote_user**
  Points to a username that is valid at the remote host. Any valid username can be specified.

- **local_user**
  Points to a username that is valid at the local host. Any valid username can be specified.

Description
The **ruserok()** (remote command user OK) function allows servers to authenticate clients requesting services.
The hostname must be specified. If the local domain and remote domain are the same, specifying the domain parts is optional. To determine the domain of the host, use the `gethostname()` function. The `ruserok()` function checks for this host in the `/etc/host.equiv` file. Then, if necessary, the subroutine checks a file in the user's home directory at the server called `$HOME/.rhosts` for a host and remote user ID.

**Return Values**

The `ruserok()` function returns 0 (zero) if the subroutine successfully locates the name specified by the `host` parameter in the `/etc/hosts.equiv` file or if the IDs specified by the `host` and `remote_user` parameters are found in the `$HOME/.rhosts` file.

If the name specified by the `host` parameter was not found, the `ruserok()` function returns a value of -1.

**Files**

`/etc/services` Contains service names.

`/etc/host.equiv` Specifies foreign hostnames.

`$HOME/.rhosts` Specifies the remote users of a local user account.

**Related Information**

Functions: `gethostname(2), rcmd(3), rresvport(3), sethostname(2)`

Commands: `rlogind(8), rshd(8)`
scandir, alphasort

**Purpose**
Scans or sorts directory contents

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <sys/types.h>
#include <sys/dir.h>

int scandir (char *dir_name,
             struct direct *(*name_list[]),
             int (*select) (void),
             int (*compare) (void));

int alphasort (struct direct **dirl,
               struct direct **dir2);
```

**Parameters**
- `dir_name` Points to the directory name.
- `name_list` Points to the array of pointers to directory entries.
- `select` Points to a user-supplied function that is called by the `scandir()` function to select which entries to include in the array.
- `compare` Points to a user-supplied function that sorts the completed array.
- `dirl` Points to a `direct` structure.
- `dir2` Points to a `direct` structure.

**Description**
The `scandir()` function reads the directory pointed to by the `dir_name` parameter. It then uses the `malloc()` function to create an array of pointers to directory entries. The `scandir()` function returns the number of entries in the array and, through the `name_list` parameter, a pointer to the array.
The `select` parameter points to a user-supplied function that the `scandir()` function calls to select which entries to include in the array. The selection routine is passed a pointer to a directory entry and returns a nonzero value for a directory entry that is included in the array. If the `select` parameter is a null value, all directory entries are included.

The `compare` parameter points to a user-supplied function that is passed to the `qsort()` function to sort the completed array. If the `compare` parameter is a null value, the array is not sorted.

The memory allocated to the array can be deallocated by freeing each pointer in the array, and the array itself, with the `free()` function.

The `alphasort()` function alphabetically compares the two `direct` structures pointed to by the `dir1` and `dir2` parameters. This function can be passed as the `compare` parameter to either the `scandir()` function or the `qsort()` function. A user-supplied subroutine may also be used.

### Return Values

The `scandir()` function returns -1 if the directory cannot be opened for reading or if the `malloc()` function cannot allocate enough memory to hold all the data structures. If successful, the `scandir()` function returns the number of entries found.

The `alphasort()` function returns the following values:

- Less than 0 (zero): The `direct` structure pointed to by the `dir1` parameter is lexically less than the `direct` structure pointed to by the `dir2` parameter.
- 0 (zero): The `direct` structures pointed to by the `dir1` parameter and the `dir2` parameter are equal.
- Greater than 0 (zero): The `direct` structure pointed to by the `dir1` parameter is lexically greater than the `direct` structure pointed to by the `dir2` parameter.

### Related Information

Functions: `malloc(3)`, `opendir(3)`, `qsort(3)`
scanf, fscanf, sscanf

**Purpose**  
Converts formatted input

**Library**  
Standard I/O package (libc.a)

**Synopsis**  
```c
#include <stdio.h>

int scanf (const char *format [, pointer, ... ] );
int fscanf (FILE *stream, const char *format [, pointer, ... ] );
int sscanf (const char *string, const char *format [, pointer, ... ] );
```

**Parameters**

- **format**  
  Specifies the format conversion.

- **stream**  
  Specifies the input stream.

- **string**  
  Specifies input to be read.

- **pointer**  
  Points to location to store interpreted data.

**Description**

The `scanf()`, `fscanf()`, and `sscanf()` functions read character data, interpret it according to a format, and store the converted results into specified memory locations. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored.

These functions read their input from the following sources:

- **scanf()**  
  Reads from standard input (stdin).

- **fscanf()**  
  Reads from the `stream` parameter.

- **sscanf()**  
  Reads from the character string specified by the `string` parameter.
The *format* parameter contains conversion specifications used to interpret the input. The *pointer* parameters specify where to store the interpreted data. If there are insufficient arguments for the *format*, the behavior is undefined. If the *format* is exhausted while arguments remain, the excess arguments are evaluated as always but are otherwise ignored.

The *format* parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.

- Any character except % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field.

### Conversion Specifications

Each conversion specification in the *format* parameter contains the following elements:

- The character % (percent sign)
- The optional assignment suppression character * (asterisk)
- An optional numeric maximum field width
- An optional character that sets the size of the receiving variable as for some flags, as follows:

  - **L** Signed long integer rather than an int when preceding the d, u, o, or x conversion codes.
  - **h** Signed short integer (half int) rather than an int when preceding the d, u, o, or x conversion codes.
  - A conversion code

The conversion specification has the following syntax:

```
%[*][width][size]convcode
```

The results from the conversion are placed in *pointer* unless you specify assignment suppression with * (asterisk). Assignment suppression provides a way to describe an input field that is to be skipped. The input field is a string of nonwhite-space characters. It extends to the next inappropriate character or until the field width, if specified, is exhausted.
The conversion code indicates how to interpret the input field. The corresponding pointer must usually be of a restricted type. You should not specify the pointer parameter for a suppressed field. You can use the following conversion codes:

- `%` Accepts a single % (percent sign) input at this point; no assignment is done.
- `d, i` Accepts a decimal integer; the pointer parameter should be an integer pointer.
- `u` Accepts an unsigned decimal integer; the pointer parameter should be an unsigned integer pointer.
- `o` Accepts an octal integer; the pointer parameter should be an integer pointer.
- `x` Accepts a hexadecimal integer; the pointer parameter should be an integer pointer.
- `e, f, g` Accepts a floating-point number. The next field is converted accordingly and stored through the corresponding parameter, which should be a pointer to a float. The input format for floating-point numbers is a string of digits, with the following optional characteristics:
  - It can be a signed value.
  - It can be an exponential value, containing a decimal point followed by an exponent field, which consists of an E or an e followed by an (optionally signed) integer.
  - It can be one of the special values INF, NaQ, or NaNQ. This value is translated into the ANSI/IEEE value for infinity, quiet NaN, or signaling NaN, respectively.

For Japanese Language Support, the conversion codes recognize double-width versions of digits as equivalent to the single-width versions of those digits.

- `p` Matches an unsigned hexadecimal integer, the same as the &p conversion of the printf() function. The corresponding argument will be a pointer to a pointer to void.

- `n` No input is consumed. The corresponding argument is a pointer to an integer into which is written the number of characters read from the input stream so far by this function. The assignment count returned at the completion of this function is not incremented.
s  Accepts a string of characters. The pointer parameter should be a character pointer that points to an array of characters large enough to accept the string and ending with \0. The \0 is added automatically. The input field ends with a white-space character. A string of char values is output.

c  A char value is expected. The pointer parameter should be a char pointer. The normal skip over white space is suppressed. Use %ls to read the next nonwhite-space character. If a field width is given, pointer refers to a character array, and the indicated number of char values is read.

c  For Japanese Language Support, a char value is expected and the pointer parameter should be a char pointer. The normal skip over white space is again suppressed, and %ls is used to read the next nonwhite-space char value. If a field width is given, pointer refers to a character array, and the indicated number of char values is read.

[scanset]
Accepts as input the characters included in the scanset. The scanset explicitly defines the characters that are accepted in the string data as those enclosed within square brackets. The leading white space that is normally skipped over is suppressed. A scanset in the form of [^scanset] is an exclusive scanset: the ^ (circumflex) serves as a complement operator and the following characters in the scanset are not accepted as input. Conventions used in the construction of the scanset follow:

- You can represent a range of characters by the construct First-Last. Thus, you can express [0123456789] as [0-9]. The First parameter must be lexically less than or equal to Last, or else the - (dash) stands for itself. The - also stands for itself whenever it is the first or the last character in the scanset.

- You can include the ] (right bracket) as an element of the scanset if it is the first character of the scanset. In this case it is not interpreted as the bracket that closes the scanset. If the scanset is an exclusive scanset, the ] is preceded by the ^ (circumflex) to make the ] an element of the scanset. The corresponding pointer parameter must point to a character array large enough to hold the data field and that ends with 0 (zero). The 0 is added automatically.
A `scanf()` ends at the end of the file, the end of the control string, or when an input character conflicts with the control string. If it ends with an input character conflict, the conflicting character is not read from the input stream.

Unless there is a match in the control string, trailing white space (including a newline character) is not read.

The success of literal matches and suppressed assignments is not directly determinable.

Japanese Language Support

The NLS extensions to the `scanf()` functions can handle a format string that enables the system to process elements of the argument list in variable order. The normal conversion character `%` (percent sign) is replaced by `%digit$`, where `digit` is a decimal number. Conversions are then applied to arguments in the list with ordinal digits, rather than to the next unused argument.

The following restrictions apply:

- The format passed to the NLS extensions can contain one of the following forms, but not both:
  - The format of the conversion.
  - The explicit or implicit argument number.

  These forms cannot be mixed within a single format string.

- The `*` (asterisk) specification for field width or precision is not permitted with the variable order `%digit$` format.

Notes

AES Support Level: Full use

Return Values

The `scanf()`, `fscanf()`, or `sscanf()` function returns the display length of the string it outputs, which is the number of the display characters in the string, rather than the number of bytes. These functions return EOF on the end of input and on a short count for missing or invalid data items.

The `scanf()` functions return the number of successfully matched and assigned input items. This number can be 0 (zero) if there was an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, EOF is returned and `errno` is set to indicate the error.
Errors

The `fscanf()` function fails if either the `stream` is unbuffered, or the `stream`'s buffer needed to be flushed and the function call caused an underlying `write()` or `lseek()` to be invoked. In addition, if the `scanf()`, `fscanf()`, `sscanf()`, function fails, `errno` may be set to one of the following values:

- `[EAGAIN]` The O_NONBLOCK flag is set for the file descriptor underlying `stream` and the process would be delayed in the write operation.
- `[EBADF]` The file descriptor underlying `stream` is not a valid file descriptor open for writing.
- `[EFBIG]` An attempt was made to write to a file that exceeds the process' file size limit or the maximum file size.
- `[EINVAL]` The read operation was interrupted by a signal which was caught, and no data was transferred.
- `[EIO]` The implementation supports job control, the process is a member of a background process group attempting to write to its controlling terminal, `TOSTOP` is set, the process is neither ignoring nor blocking `SIGTTOU` and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
- `[ENOSPC]` There was no free space remaining on the device containing the file.
- `[EPIPE]` An attempt was made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

Related Information

Functions: `atof(3)`, `atoi(3)`, `getc(3)`, `getwc(3)`, `printf(3)`, `wscanf(3)`
select

Purpose  
Synchronous I/O multiplexing

Synopsis
#include<sys/types.h>
#include<sys/time.h>

int select(
    int nfds,
    fd_set *readfds,
    fd_set *writefds,
    fd_set *exceptfds,
    struct timeval *timeout);

FD_SET(
    int fd,
    fd_set *fdset);

FD_CLR(
    int fd,
    fd_set *fdset);

FD_ISSET(
    int fd,
    fd_set *fdset);

FD_ZERO(
    fd_set *fdset);

Parameters

nfds  
Specifies the number of open objects that may be ready for reading or writing or that have exceptions pending. The nfds parameter cannot be greater than FD_SETSIZE.

readfds  
Points to an I/O descriptor set consisting of file descriptors of objects opened for reading. When the readfds parameter is a null pointer, the read I/O descriptor set is ignored by the select() function.

writefds  
Points to an I/O descriptor set consisting of file descriptors for objects opened for writing. When the writefds parameter is a null pointer, the write I/O descriptor set is ignored.
**Description**

The `select()` function checks the status of objects identified by bit masks called I/O descriptor sets. Each I/O descriptor set consists of an array of bits whose relative position and state represent a file descriptor and the status of its corresponding object. There is an I/O descriptor set for reading, writing, and for pending exceptions. These I/O descriptor sets are pointed to by the `readfds`, `writefds`, and `exceptfds` parameters, respectively. The I/O descriptor sets provide a means of monitoring the read, write, and exception status of objects represented by file descriptors.

The status of `nfds - 1` file descriptors in each referenced I/O descriptor set is checked when the `select()` function is called. The `select()` function returns a modified I/O descriptor set, which has the following characteristics: for any selected I/O descriptor set pointed to by the `readfds`, `writefds`, and `exceptfds` parameters, if the state of any bit corresponding with an active file descriptor is set on entry, when the object represented by the set bit is ready for reading, writing, or its exception condition has been satisfied, a corresponding bit position is also set in the returned I/O descriptor set pointed to by the `readfds`, `writefds`, or `exceptfds` parameters.

On return, the `select()` function replaces the original I/O descriptor sets with the corresponding I/O descriptor sets that have a set bit for each file descriptor representing those objects that are ready for the requested operation. The total number of ready objects represented by set bits in all the I/O descriptor sets is returned by the `select()` function.
After an I/O descriptor set is created, it may be modified with the following macros:

FD_ZERO(&fdset)
Initializes the I/O descriptor set addressed by fdset to a null value.

FD_SET(fd, &fdset)
Includes the particular I/O descriptor bit specified by fd in the I/O descriptor set addressed by fdset.

FD_CLR(fd, &fdset)
Clears the I/O descriptor bit specified by file descriptor fd in the I/O descriptor set addressed by fdset.

FD_ISSET(fd, &fdset)
Returns a nonzero value when the I/O descriptor bit for fd is included in the I/O descriptor set addressed by fdset. Otherwise 0 (zero) is returned.

The behavior of these macros is undefined when parameter fd has a value less than 0 (zero) or greater than or equal to FD_SETSIZE, which is normally at least equal to the maximum number of file descriptors supported by the system.

Notes

Although the getdtablesize() function is intended to allow users to write programs independently of the kernel limit on the number of open files, the dimensioning of a sufficiently large bit field for select() remains a problem. The default size FD_SETSIZE (currently 256) is larger than the current kernel limit on the permitted number of open files. To accommodate programs that specify more open files with the select() function, it is possible to specify an alternate value for FD_SETSIZE before including the sys/types.h header file.

Return Values

Upon successful completion, the select() function returns the number of ready objects represented by corresponding file descriptor bits in the I/O descriptor sets. When an error occurs, -1 is returned. When the time limit specified by values pointed to by the timeout parameter expires, this function returns the value 0 (zero).

When select() returns an error, including a process interrupt, the I/O descriptor sets pointed to by the readfds, writefds, and exceptfds parameters remain unmodified.
Errors

If the `select()` function fails, `errno` may be set to one of the following values:

- **[EBADF]** One of the I/O descriptor sets you specified is invalid.
- **[EINTR]** A signal was delivered before the time limit specified by the `timeout` parameter expired and before any of the selected events occurred.
- **[EINVAL]** The time limit specified by the `timeout` parameter is invalid. One of its components is negative or too large.

Related Information

Functions: `accept(2)`, `connect(2)`, `send(2)`, `getdtablesize(2)`, `poll(2)` `read(2)`, `recv(2)`, `write(2)`
Purpose

Performs semaphore control operations

Synopsis

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl(
    int semid,         // Specifies the ID of the semaphore set.
    int semnum,        // Specifies the number of the semaphore to be processed.
    int cmd,           // Specifies the type of command. See DESCRIPTION.
    union semun {      // The address of a user data structure to be used either to set or to return semaphore values. If the structure is specified, the calling process must allocate it before making the call. The members of this structure are described as follows:
        int val, // Contains the semaphore value to which semval is set when the SETVAL command is performed.
        struct semid_ds *buf, // Points to the structure containing the contents of the requested semid_ds. When the IPC_STAT command is performed, the contents of the requested semid_ds structure are copied into arg.buf. When the IPC_SET command is performed, the contents of arg.buf are copied into the requested the semid_ds structure.
        u_short *array, // Points to an array of semval values. These semval values are returned by the GETALL command and set by the SETALL command.
    } arg);
```

Parameters

- **semid**: Specifies the ID of the semaphore set.
- **semnum**: Specifies the number of the semaphore to be processed.
- **cmd**: Specifies the type of command. See DESCRIPTION.
- **arg**: The address of a user data structure to be used either to set or to return semaphore values. If the structure is specified, the calling process must allocate it before making the call. The members of this structure are described as follows:
  - **val**: Contains the semaphore value to which semval is set when the SETVAL command is performed.
  - **buf**: Points to the structure containing the contents of the requested semid_ds. When the IPC_STAT command is performed, the contents of the requested semid_ds structure are copied into arg.buf. When the IPC_SET command is performed, the contents of arg.buf are copied into the requested the semid_ds structure.
  - **array**: Points to an array of semval values. These semval values are returned by the GETALL command and set by the SETALL command.
Description

The `semctl()` function allows a process to perform various operations on an individual semaphore within a semaphore set, on all semaphores within a semaphore set, and on the `semid_ds` structure associated with the semaphore set. It also allows a process to remove the semaphore set’s ID and its associated `semid_ds` structure.

The `cmd` value determines which operation is performed. The following commands operate on the specified semaphore (that is, `semnum`) within the specified semaphore set:

- **GETVAL** Returns the value of `semval`. This command requires read permission.
- **SETVAL** Sets the value of `semval` to `arg.val`. When this command successfully executes, the kernel clears the semaphore’s adjust-on-exit value in all processes. This command requires modify permission.
- **GETPID** Returns the value of `sempid`. This command requires read permission.
- **GETNCNT** Returns the value of `semncnt`. This command requires read permission.
- **GETZCNT** Returns the value of `semzcnt`. This command requires read permission.

The following commands operate on all the semaphores in the semaphore set:

- **GETALL** Returns all the `semval` values and places them in the array pointed to by `arg.array`. This command requires read permission.
- **SETALL** Sets all the `semval` values according to the array pointed to by `arg.array`. When this command successfully executes, the kernel clears the semaphore’s adjust-on-exit value in all processes. This command requires modify permission.

The following IPC commands can also be used:

- **IPC_STAT** Queries the semaphore ID by copying the contents of its associated `semid_ds` structure into the structure pointed to by `arg.buf`. This command requires read permission.
- **IPC_SET** Sets the semaphore set by copying the user-supplied values found in the `arg.buf` structure into corresponding fields in the `semid_ds` structure associated with the semaphore ID. This is a restricted operation. The effective user ID of the calling process must have
superuser privilege or must be equal to the value of `sem_perm.cuid` or `sem_perm.uid` in the structure associated with the semaphore ID. The fields are set as follows:
- The `sem_perm.uid` field is set to the owner’s user ID.
- The `sem_perm.gid` field is set to the owner’s group ID.
- The `sem_perm.mode` field is set to the access modes for the semaphore set. Only the low-order nine bits are set.

**IPC_RMID**
Removes the semaphore ID and destroys the set of semaphores and the `semid_ds` data structure associated with it. This is a restricted operation. The effective user ID of the calling process must have superuser privilege or equal to the value of `sem_perm.cuid` or `sem_perm.uid` in the associated `semid_ds` structure.

### Return Value
Upon successful completion, the value returned depends on the `cmd` parameter as follows:

- **GETVAL** Returns the value of `semval`.
- **GETPID** Returns the value of `sempid`.
- **GETNCNT** Returns the value of `semncnt`.
- **GETZCNT** Returns the value of `semzcnt`.

All other commands return a value of 0 (zero).

If the `semctl()` function fails, a value of -1 is returned and `errno` is set to indicate the error.

### Errors
If the `semctl()` function fails, `errno` may be set to one of the following values:

- **[EINVAL]** The `semid` parameter is not a valid semaphore ID; the value of `sem_num` is less than 0 (zero) or greater than `sem_nsems`; or `cmd` is not a valid command.

- **[EACCES]** The calling process does not have the required permission.

- **[ERANGE]** The `cmd` parameter is SETVAL or SETALL and the value to which `semval` is to be set is greater than the system-defined maximum.
Either the *cmd* parameter is equal to IPC_RMID and the effective user ID of the calling process does not have appropriate privilege, or the *cmd* parameter is equal to IPC_SET and the effective user ID of the calling process is not equal to the value of *sem_perm.cuid* or *sem_perm.uid* in the *semid_ds* structure associated with the semaphore ID.

The *cmd* parameter is IPC_STAT or IPC_SET and an error occurred in accessing the *arg* structure.

The system does not have enough memory to complete the function.

**Related Information**

Functions: *semget(2), semop(2)*

Data structures: *semid_ds(4)*
semget

Purpose

Returns (and possibly creates) a semaphore ID

Synopsis

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(
    key_t key,
    int nsems,
    int semflg):
```

Parameters

- **key**: Specifies the key that identifies the semaphore set. The IPC_PRIVATE key can be used to assure the return of a new, unused, entry in the semaphore table.
- **nsems**: Specifies the number of semaphores to create in the semaphore set.
- **semflg**: Specifies the creation flags. Possible values are:
  - **IPC_CREAT**: If the key does not exist, the `semget()` function creates a semaphore set using the given key. If the key does exist, forces an error notification.
  - **IPC_EXCL**: If the key already exists, the `semget()` function fails and returns an error notification.

The low-order nine bits of `sem_perm.mode` are set equal to the low-order nine bits of `semflg`.

Description

The `semget()` function returns (and possibly creates) the ID for a semaphore set identified by the `key` parameter. Semaphores are used primarily for synchronization between processes.

The sets of semaphores are implemented collectively as a system-wide table, with each set being an entry in the table. The returned ID identifies the semaphore set’s entry in the table. Each set of semaphores is implemented using the `semid_ds` data structure. This structure defines an array whose members are the individual semaphores in the set.
Functions

semget(2)

Each individual semaphore within a set is implemented using the `sem` structure. The `semget()` function creates a semaphore ID, its associated `semid_ds` data structure, and `nsems` individual semaphores if one of the following is true:

- The `key` parameter is IPC_PRIVATE.
- The `key` parameter does not already exist as an entry in the semaphore table and the IPC_CREAT flag is set.

After creating a semaphore ID, the `semget()` function initializes the `semid_ds` structure associated with the ID as follows:

- The `sem_perm.cuid` and `sem_perm.uid` fields are set equal to the effective user ID of the calling process.
- The `sem_perm.cgid` and `sem_perm.gid` fields are set equal to the effective group ID of the calling process.
- The low-order nine bits of `sem_perm.mode` are set equal to the low-order nine bits of `semflg`.
- The `sem_nsems` field is set equal to the value of `nsems`.
- The `sem_otime` field is set equal to 0 (zero) and the `sem_ctime` field is set equal to the current time.

The `semget()` function does not initialize the `sem` structure associated with each semaphore in the set. The individual semaphores are initialized by using the `semctl()` function with the SETVAL or SETALL command.

Return Values

Upon successful completion, a semaphore identifier is returned. If the `semget()` function fails, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `semget()` function fails, `errno` may be set to one of the following values:

- **[EACCES]** A semaphore ID already exists for the `key` parameter, but operation permission as specified by the low-order nine bits of the `semflg` parameter was not granted.
- **[EINVAL]** The value of the `nsems` parameter is less than or equal to 0 (zero) or greater than the system-defined limit. Or, a semaphore ID already exists for the `key` parameter, but the number of semaphores in the set is less than the `nsems` parameter, and the `nsems` parameter is not equal to 0 (zero).
- **[ENOENT]** A semaphore ID does not exist for the `key` parameter and IPC_CREAT was not set.
semget(2)

[ENOSPC] An attempt to create a new semaphore ID exceeded the system-wide limit on the size of the semaphore table.

[EEXIST] A semaphore ID already exists for the key parameter, but IPC_CREAT and IPC_EXCL were used for the semflg parameter.

Related Information

Functions: semctl(2), semop(2)

Data structures: semid_ds(4)
semop

Purpose
Performs semaphore operations

Synopsis
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(
    int semid,
    struct sembuf *sops,
    u_int nsops);

Parameters

semid Specifies the ID of the semaphore set.
sops Points to the user-defined array of sembuf structures that contain
        the semaphore operations.
nsops The number of sembuf structures in the array.

Description

The semop() function performs operations on the semaphores in the specified
semaphore set. The semaphore operations are defined in the sops array. The sops
array contains nsops elements, each of which is represented by a sembuf structure.

The sembuf structure (from sys/sem.h) is shown here:

struct sembuf {
    u_short  sem_num;
    short    sem_op;
    short    sem_flg;
};

The fields in the sembuf structure are defined as follows:

sem_num Specifies an individual semaphore within the semaphore set.
sem_op Specifies the operation to perform on the semaphore.
semop() function does one of the following:

- If the semaphore’s current value (in semval) is equal to or greater than the absolute value of sem_op, the absolute value of sem_op is subtracted from semval. If SEM_UNDO is set, the absolute value of sem_op is added to the calling process’ adjust-on-exit value for the semaphore.

- If semval is less than the absolute value of sem_op and IPC_NOWAIT is set, semop() returns immediately with an [EAGAIN] error.

- If semval is less than the absolute value of sem_op and IPC_NOWAIT is not set, semop() increments the semaphore’s semncnt value and suspends the calling process.

If the process is suspended, it sleeps until one of the following occurs:

- The semval value becomes equal to or greater than the absolute value of sem_op. In this case, the semaphore’s semncnt value is decremented; the absolute value of sem_op is subtracted from semval; and, if SEM_UNDO is set, the absolute value of sem_op is added to the calling process’s adjust-on-exit value for the semaphore.

- The semaphore set (specified by semid) is removed from the system. In this case, errno is set equal to [EIDRM] and a value of -1 is returned to the calling process.

- The calling process catches a signal. In this case, the semaphore’s semncnt value is decremented, and the calling process resumes execution as directed by the signal() function.
If `sem_op` is a positive integer and the calling process has modify permission, `semop()` adds the `sem_op` value to the semaphore's current `semval` value. If SEM_UNDO is set, the `sem_op` value is subtracted from the calling process's adjust-on-exit value for the semaphore.

If `sem_op` is 0 (zero) and the calling process has read permission, `semop()` does one of the following:

- If `semval` is 0, `semop()` returns immediately.
- If `semval` is not equal to 0 and IPC_NOWAIT is set, `semop()` returns immediately.
- If `semval` is not equal to 0 and IPC_NOWAIT is not set, `semop()` increments the semaphore's `semzcnt` value and suspends the calling process.

If the process is suspended, it sleeps until one of the following occurs:

- The `semval` value becomes 0 (zero). In this case, the semaphore's `semncnt` value is decremented.
- The semaphore set (specified by `semid`) is removed from the system. In this case, `errno` is set equal to [EIDRM] and a value of -1 is returned to the calling process.
- The calling process catches a signal. In this case, the semaphore's `semncnt` value is decremented, and the calling process resumes execution as directed by the `signal()` function.

**Notes**

Semaphore operations are performed atomically; that is, either all of the requested operations are performed, or none are. If the kernel goes to sleep while doing the operations, it restores all of the semaphores in the set to their previous values, at the start of the `semop()` function.

**Return Values**

Upon successful completion, the `semop()` function returns a value of 0 (zero) and the `semid` value for each semaphore that is operated upon is set to the process ID of the calling process.

If the `semop()` function fails, a value of -1 is returned and `errno` is set to indicate the error.
Errors

If the `semop()` function fails, `errno` may be set to one of the following values:

- **[EINVAL]** The `semid` parameter is not a valid semaphore ID, or the number of semaphores for which SEM_UNDO is requested exceeds the system-defined limit.
- **[EFBIG]** The `sem_num` parameter is less than 0 (zero) or greater than or equal to the number of semaphores in `semid`.
- **[E2BIG]** The `nsops` parameter is greater than the system-defined maximum.
- **[EACCES]** The calling process does not have the required permission.
- **[ENOSPC]** The system-defined limit on the number of processes using SEM_UNDO was exceeded.
- **[ERANGE]** An operation caused a `semval` to overflow the system-defined limit, or an operation caused an adjust-on-exit value to exceed the system-defined limit.
- **[EINTR]** The `semop()` function was interrupted by a signal.
- **[EIDRM]** The semaphore ID specified by the `semid` parameter has been removed from the system.

Related Information

Functions: `exec(2), exit(2), fork(2), semctl(2), semget(2)`

Data Structures: `semid_ds(4)`
send

Purpose Sends messages on a socket

Synopsis

```c
#include <sys/types.h>
#include <sys/socket.h>

int send (
    int socket,
    char *message,
    int length,
    int flags);
```

Parameters

- **socket**: Specifies the unique name for the socket.
- **message**: Points to the address of the message to send.
- **length**: Specifies the length of the message in bytes.
- **flags**: Allows the sender to control the transmission of the message. The `flags` parameter to send a call is formed by logically ORing the values shown in the following list, defined in the `sys/socket.h` header file:
  - **MSG_OOB**: Sends out-of-band data on sockets that support out-of-band communication.
  - **MSG_DONTROUTE**: Sends without using routing tables. (Not recommended, for debugging purposes only.)

Description

The `send()` function sends a message only when the socket is connected. The `sendto()` and `sendmsg()` functions can be used with unconnected or connected sockets.

Specify the length of the message with the `length` parameter. If the message is too long to pass through the underlying protocol, the system returns an error and does not transmit the message.

No indication of failure to deliver is implied in a `send()` function. A return value of -1 indicates only locally detected errors.
If no space for messages is available at the sending socket to hold the message to be transmitted, the `send()` function blocks unless the socket is in a nonblocking I/O mode. Use the `select()` function to determine when it is possible to send more data.

**Notes**

The `send()` function is identical to the `sendto()` function with a zero-valued `dest_len` parameter, and to the `write()` function if no flags are used. For that reason, the `send()` function is disabled when 4.4BSD behavior is enabled (that is, when the `_SOCKADDR_LEN` compile-time option is defined).

**Return Values**

Upon successful completion, the `send()` function returns the number of characters sent. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `send()` function fails, `errno` may be set to one of the following values:

- [EBADF] The `socket` parameter is not valid.
- [ENOTSOCK] The `socket` parameter refers to a file, not a socket.
- [EFAULT] The `message` parameter is not in a readable or writable part of the user address space.
- [EMSGSIZE] The message is too large be sent all at once, as the socket requires.
- [EWOULDBLOCK] The socket is marked nonblocking, and no space is available for the `send()` function.

**Related Information**

Functions: `recv(2)`, `recvfrom(2)`, `recvmsg(2)`, `sendmsg(2)`, `sendto(2)`, `shutdown(2)`, `connect(2)`, `socket(2)`, `getsockopt(2)`, `select(2)`, `setsockopt(2)`
sendmsg

**Purpose** Sends a message from a socket using a message structure

**Synopsis**
```
#include <sys/types.h>
#include <sys/socket.h>
int sendmsg (  
    int socket,  
    struct msghdr *message,  
    int flags );
```

**Parameters**

- `socket` Specifies the socket descriptor.

- `message` Points to a `msghdr` structure, containing both the address for the incoming message and the buffers for the source address. The format of the address is determined by the behavior requested for the socket. If the compile-time option `_SOCKADDR_LEN` is defined before the `sys/socket.h` header file is included, the `msghdr` structure takes 4.4BSD behavior. Otherwise, the default 4.3BSD `msghdr` structure is used.

  In 4.4BSD, the `msghdr` structure has a separate `msg_flags` field for holding flags from the received message. In addition, the `msg_accrights` field is generalized into a `msg_control` field. See the `recvmsg()` function for more information.

  If `_SOCKADDR_LEN` is defined, the 4.3BSD `msghdr` structure is defined with the name `omsghdr`.

- `flags` Allows the sender to control the message transmission. The `sys/socket.h` file contains the `flags` values. The `flags` value to send a call is formed by logically ORing the following values:

  **MSG_OOB** Processes out-of-band data on sockets that support out-of-bound data.

  **MSG_DONTROUTE** Sends without using routing tables. (Not recommended, for debugging purposes only.)
sendmsg(2)

Description

The sendmsg() function sends messages through connected or unconnected sockets using the msghdr message structure. The sys/socket.h file contains the msghdr structure and defines the structure members.

To broadcast on a socket, the application program must first issue a setsockopt() function using the SO_BROADCAST option to gain broadcast permissions.

Return Values

Upon successful completion, the sendmsg() function returns the number of characters sent. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the sendmsg() function fails, errno may be set to one of the following values:

- [EBADF] The socket parameter is not valid.
- [ENOTSOCK] The socket parameter refers to a file, not a socket.
- [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.
- [EWOULDBLOCK] The socket is marked nonblocking, and no space is available for the sendmsg() function.

Related Information

Functions: recv(2), recvfrom(2), recvmsg(2), send(2), sendto(2), shutdown(2), socket(2), select(2), getsockopt(2), setsockopt(2)
sendto

**Purpose** Sends messages through a socket

**Synopsis**
```
#include <sys/types.h>
#include <sys/socket.h>

int sendto (int socket, char *message_addr, int length, int flags, struct sockaddr *dest_addr, int dest_len);
```

**Parameters**

- **socket** Specifies the unique name for the socket.
- **message_addr** Points to the address containing the message to be sent.
- **length** Specifies the size of the message in bytes.
- **flags** Allows the sender to control the message transmission. The flags value to send a call is formed by logically ORing the following values, defined in the `sys/socket.h` file:
  - **MSG_OOB** Processes out-of-band data on sockets that support out-of-band data.
  - **MSG_DONTROUTE** Sends without using routing tables. (Not recommended, for debugging purposes only.)
- **dest_addr** Points to a `sockaddr` structure, the format of which is determined by the domain and by the behavior requested for the socket. The `sockaddr` structure is an overlay for a `sockaddr_in`, `sockaddr_un`, or `sockaddr_ns` structure, depending on which of the supported address families is active.
If the compile-time option \_SOCKADDR\_LEN is defined before the \texttt{sys/socket.h} header file is included, the \texttt{sockaddr} structure takes 4.4BSD behavior, with a field for specifying the length of the socket address. Otherwise, the default 4.3BSD \texttt{sockaddr} structure is used, with the length of the socket address assumed to be 14 bytes or less.

If \_SOCKADDR\_LEN is defined, the 4.3BSD \texttt{sockaddr} structure is defined with the name \texttt{osockaddr}.

\begin{description}
\item[\texttt{dest\_len}] Specifies the length of the \texttt{sockaddr} structure pointed to by the \texttt{dest\_addr} parameter.
\end{description}

\section*{Description}

The \texttt{sendto(\() \) function allows an application program to send messages through an unconnected socket by specifying a destination address.

To broadcast on a socket, issue a \texttt{setsockopt(\() \) function using the SO\_BROADCAST option to gain broadcast permissions.

Use the \texttt{dest\_addr} parameter to provide the address of the target. Specify the length of the message with the \texttt{length} parameter.

If the \texttt{sending} socket has no space to hold the message to be transmitted, the \texttt{sendto(\() \) function blocks unless the socket is in a nonblocking I/O mode.

Use the \texttt{select(\() \) function to determine when it is possible to send more data.

\section*{Return Values}

Upon successful completion, the \texttt{sendto(\() \) function returns the number of characters sent. Otherwise, the a value of -1 is returned, and \texttt{errno} is set to indicate the error.

\section*{Errors}

If the \texttt{sendto(\() \) function fails, \texttt{errno} may be set to one of the following values:

\begin{itemize}
\item \texttt{[EBADF]} The \texttt{socket} parameter is not valid.
\item \texttt{[ENOTSOCK]} The \texttt{socket} parameter refers to a file, not a socket.
\item \texttt{[EFAULT]} The \texttt{dest\_addr} parameter is not in a writable part of the user address space.
\end{itemize}
[EMSGSIZE] The message is too large to be sent all at once, as the socket requires.

[EWOULDBLOCK]  
The socket is marked nonblocking, and no space is available for the sendto() function.

Related Information

Functions:  recv(2), recvfrom(2), recvmsg(2), send(2), sendmsg(2), shutdown(2), socket(2), select(2), getsockopt(2), setsockopt(2)
setbuf, setvbuf, setbuffer, setlinebuf

**Purpose**
Assigns buffering to a stream

**Library**
Standard I/O Package (libc.a)

**Synopsis**
```c
#include <stdio.h>

void setbuf (FILE *stream, char *buffer);

int setvbuf (FILE *stream, char *buffer, int mode, size_t size);

void setbuffer (FILE *stream, char *buffer, char *size);

void setlinebuf (FILE *stream);
```

**Parameters**
- **stream**: Specifies the input/output stream.
- **buffer**: Points to a character array.
- **mode**: Determines how the `stream` parameter is buffered.
- **size**: Specifies the size of the buffer to be used.

**Description**
The `setbuf()` function causes the character array pointed to by the `buffer` parameter to be used instead of an automatically allocated buffer. Use the `setbuf()` function after a stream has been opened, but before it is read or written.

If the `buffer` parameter is a null character pointer, input/output is completely unbuffered.
A constant, BUFSIZ, defined in the **stdio.h** header file, tells how large an array is needed:

```c
char buf[BUFSIZ];
```

For the **setvbuf()** function, the *mode* parameter determines how the *stream* parameter is buffered:

- _IOFBF Causes input/output to be fully buffered.
- _IOLBF Causes output to be line-buffered. The buffer is flushed when a new line is written, the buffer is full, or input is requested.
- _IONBUF Causes input/output to be completely unbuffered.

If the *buffer* parameter is not a null character pointer, the array it points to is used for buffering instead of an automatically allocated buffer. The *size* parameter specifies the size of the buffer to be used. The constant BUFSIZ in the **stdio.h** header file is one buffer size. If input/output is unbuffered, the *buffer* and *size* parameters are ignored. The **setbuffer()** function, an alternate form of the **setbuf()** function, is used after *stream* has been opened, but before it is read or written. The character array *buffer*, whose size is determined by the *size* parameter, is used instead of an automatically allocated buffer. If the *buffer* parameter is a null character pointer, input/output is completely unbuffered.

The **setbuffer()** function is not needed under normal circumstances since the default file I/O buffer size is optimal.

The **setlinebuf()** function is used to change stdout or stderr from block-buffered or unbuffered to line-buffered. Unlike the **setbuf()** and **setbuffer()** functions, the **setlinebuf()** function can be used any time the file descriptor is active.

A buffer is normally obtained from the **malloc()** function at the time of the first **getc()** or **putc()** function on the file, except that the standard error stream, **stderr**, is normally not buffered.

Output streams directed to terminals are always either line-buffered or unbuffered.

**Notes**

A common source of error is allocating buffer space as an automatic variable in a code block, and then failing to close the stream in the same block.

**AES Support Level:** Full use (**setbuf()**, **setvbuf()**)

**Related Information**

Functions: **fopen(3)**, **fread(3)**, **getc(3)**, **getwc(3)**, **malloc(3)**, **putc(3)**, **putwc(3)**
setclock

Purpose
Sets value of system-wide clock

Library
Standard C Library (libc.a)

Synopsis
#include <sys/timers.h>

int setclock(
    int clktyp,
    struct timespec *val);

Parameters

clktyp
Specifies a system-wide clock whose symbolic name must be
TIMEOFDAY.

val
Points to the location where the value of the time to set into
the clock specified by the clktyp parameter is stored.

Description

The setclock() function sets a time value into the system-wide clock whose
symbolic name is specified by the clktyp parameter, which must be TIMEOFDAY,
declared in the sys/timers.h header file.

The source of the current value of time set into the system-wide time-of-day clock
by this function is stored in space reserved by a type timespec structure pointed to
by the val parameter. This time information is the amount of time since the epoch.
The epoch is referenced to 00:00:00 GMT (Greenwich Mean Time) 1 Jan 1970.
The timespec structure, which is also defined in the sys/timers.h header file has
the following members:

unsigned long tv_sec Time of day since the epoch in seconds.
long tv_nsec Time of day fraction of a second (expressed in
nanoseconds).

Notes

AES Support Level: Trial use
Return Values

Upon successful completion, the `setclock()` function returns a value of 0 (zero). Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `setclock()` function fails, `errno` may be set to one of the following values:

- **[EINVAL]** The `clktyp` parameter does not specify a known system-wide clock, the information pointed to by the `val` parameter is outside the permissible range for the clock specified by the `clktyp` parameter, or a nanosecond value less than zero or greater than or equal to 1000 million is specified by the information pointed to by the `val` parameter.

- **[EIO]** An error occurred while accessing the clock specified by the `clktyp` parameter.

- **[EPERM]** The requesting process does not have the appropriate privilege to set the clock specified by the `clktyp` parameter.

Related Information

Functions: `getclock(3)`, `gettimer(3)`, `time(3)`
setgid

Purpose
Sets the group ID

Synopsis
#include <sys/types.h>

int setgid (       
    gid_t gid );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gid</td>
<td>Specifies the new group ID.</td>
</tr>
</tbody>
</table>

Description
The setgid() function sets the real group ID, effective group ID, and the saved set group ID to the value specified by the gid parameter.

If the process does not have superuser privilege, but the gid parameter is equal to the real group ID or the saved set group ID, the setgid() function sets the effective group ID to gid; the real group ID and saved set group ID remain unchanged.

Any supplementary group IDs of the calling process remain unchanged.

Notes

AES Support Level: Full use

Return Values
Upon successful completion, the setgid() function returns 0 (zero). Otherwise, -1 is returned and errno is set to indicate the error.

Errors
If the setgid() function fails, errno may be set to one of the following values:

- [EINVVAL] The value of the gid parameter is invalid.
- [EPERM] The process does not have superuser privilege and the gid parameter does not match the real group ID or the saved set group ID.
Related Information

Functions: **exec(2), getgid(2), setuid(2)**
setgroups

Purpose       Sets the group access list

Synopsis      
#include <unistd.h>
#include <sys/types.h>
int setgroups (
    int gidsetsize,
    gid_t grouplist[]);

Parameters    
gidsetsize    Indicates the number of entries in the array pointed to by the
              grouplist parameter. Must not be more than NGROUPS_MAX, as
              defined in the limits.h header file.

grouplist     Points to the array that contains the group access list of the current
              user process. Element grouplist[0] becomes the new effective group
              ID.

Description   
The setgroups() function sets the group access list of the current user process
according to the array pointed to by the grouplist parameter.

This function fails unless the invoking process has superuser privilege.

Notes         
AES Support Level: Trial use

Return Values 
Upon successful completion, a value of 0 (zero) is returned. If the setgroups() function fails, a value of -1 is returned and errno is set to indicate the error.
Errors

If the `setgroups()` function fails, `errno` may be set to one of the following values:

- **[EPERM]** The caller does not have the appropriate system privilege.
- **[EINVAL]** The value of the `gidsetsize` parameter is greater than NGROUPS_MAX or an entry in the `grouplist` parameter is not a valid group ID.
- **[EFAULT]** The `grouplist` parameter points outside of the allocated address space of the process.

Related Information

Functions: `getgroups(2)`, `initgroups(3)`
sethostid

Purpose
Sets the unique identifier of the current host

Synopsis

```
int sethostid (  
    int host_id );
```

Parameters

```
host_id
```
Specifications the unique 32-bit identifier for the current host.

Description

The `sethostid()` function allows a calling process with a root user ID to set a new
32-bit identifier for the current host. The `sethostid()` function enables an
application program to reset the host ID.

The host ID is a unique number which may be used by application programs. It is
usually set to the primary IP address of the local machine.

The `sethostid()` function fails if the calling process does not have superuser
privilege.

Return Values

Upon successful completion, the `sethostid()` function returns a value of 0 (zero).
If the `sethostid()` function fails, a value of -1 is returned and `errno` is set to
indicate the error.

Errors

If the `sethostid()` function fails, `errno` may be set to the following value:

```
[EPERM] The calling process does not have the appropriate privilege.
```

Related Information

Functions: `gethostid(2), gethostname(2)`
sethostname

Purpose
Sets the name of the current host

Synopsis

```c
int sethostname (char *name,
int name_len);
```

Parameters

- **name**: Points to an array of bytes where the hostname is stored.
- **name_len**: Specifies the length of the array pointed to by the *name* parameter.

Description

The `sethostname()` function allows a calling process with root user authority to set the internal hostname of a machine on a network.

System hostnames are limited to MAXHOSTNAMELEN as defined in the `/usr/include/sys/param.h` file.

The `sethostid()` function fails if the calling process does not have superuser privilege.

Return Values

Upon successful completion, the system returns a value of 0 (zero). If the `sethostname()` function fails, -1 is returned and `errno` is set to indicate the error.

Errors

If the `sethostname()` function fails, `errno` may be set to one of the following values:

- **EFAULT**: The *name* parameter or the *name_len* parameter gives an address that is not valid.
- **EPERM**: The calling process does not have appropriate privilege.

Related Information

Functions: `gethostid(2), sethostid(2), gethostname(2)`
setjmp, longjmp

Purpose
Saves and restores the current execution context

Library
Standard C Library (libc.a)

Synopsis
#include <setjmp.h>

int setjmp (
    jmp_buf environment );

void longjmp ( 
    jmp_buf environment, 
    int value );

int _setjmp ( 
    jmp_buf environment );

void _longjmp ( 
    jmp_buf environment, 
    int value );

Parameters

environment  Specifies an address for a jmp_buf structure.

value  Specifies any nonzero value.

Description

The setjmp() and longjmp() functions are useful when handling errors and interrupts encountered in low-level functions of a program.

The setjmp() function saves the current stack context and signal mask in the buffer specified by the environment parameter.

The longjmp() function restores the stack context and signal mask that were saved by the setjmp() function in the corresponding environment buffer. After the longjmp() function runs, program execution continues as if the corresponding call to the setjmp() function had just returned the value of the value parameter. The function that called the setjmp() function must not have returned before the completion of the longjmp() function. The setjmp() function and the longjmp() function save and restore the signal mask, while _setjmp() and _longjmp() manipulate only the stack context.
As it bypasses the usual function call and return mechanisms, the `longjmp()` function executes correctly in contexts of interrupts, signals, and any of their associated functions. However, if the `longjmp()` function is invoked from a nested signal handler (that is, from a function invoked as a result of a signal raised during the handling of another signal), the behavior is undefined.

**Notes**

The reentrant versions of the `setjmp()` and `longjmp()` functions are identical in behavior to the `_setjmp()` and `_longjmp()` functions.

The System V versions of the `setjmp()` and `longjmp()` functions, which are equivalent to `_setjmp()` and `_longjmp()` respectively, are also supported for compatibility. To use the System V versions of `setjmp()` and `longjmp()`, you must link with the `libsys5` library before you link with `libc`.

AES Support Level: Full use

**Caution**

If the `longjmp()` function is called with an `environment` parameter that was not previously set by the `setjmp()` function, or if the function that made the corresponding call to the `setjmp()` function has already returned, then the results of the `longjmp()` function are undefined. If the `longjmp()` function detects such a condition, it calls the `longjmperror()` function. If `longjmperror()` returns, the program is aborted. The default version of `longjmperror()` prints an error message to standard error and returns. Users wishing to exit more gracefully can write their own versions of the `longjmperror()` program.

**Return Values**

The `setjmp()` function returns a value of 0 (zero), unless the return is from a call to the `longjmp()` function, in which case `setjmp()` returns a nonzero value.

The `longjmp()` function cannot return 0 (zero) to the previous context. The value 0 is reserved to indicate the actual return from the `setjmp()` function when first called by the program. If the `longjmp()` function is passed a `value` parameter of 0, then execution continues as if the corresponding call to the `setjmp()` function had returned a value of 1. All accessible data have values as of the time the `longjmp()` function is called.

**Related Information**

Functions: `siglongjmp(3), sigsetjmp(3)`
setlocale

**Purpose**
Changes or queries the program's entire current locale or portions thereof

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <locale.h>
char *setlocale (int category,
    const char *locale);
int setlocale_r (int category,
    const char *locale,
    char *result);
```

**Parameters**
- **category**
  Specifies a value from the locale.h header file that names the program's entire locale or a portion thereof.
- **locale**
  Points to a string defining the locale.
- **result**
  Points to the string associated with category for the new locale.

**Description**
The `setlocale()` function selects the appropriate portion of the program's locale as specified by the category and locale parameters. The `setlocale()` function can be used to change or query the program's entire current locale or portions thereof. The LC_ALL value for the category parameter names the entire locale; the other values name only a portion of the program locale, as follows:

- **LC_COLLATE**
  Affects the behavior of the `strcoll()` and `strxfrm()` functions.

- **LC_CTYPE**
  Affects the behavior of the character handling functions (except for the `isdigit()` and `isxdigit()` functions) and the multibyte functions.

- **LC_MONETARY**
  Affects the monetary formatting information returned by the `localeconv()` function.
LC_NUMERIC
Affects the decimal-point character for the formatted input/output functions and the string conversion functions, as well as the nonmonetary formatting information returned by the `localeconv()` function.

LC_TIME
Affects the behavior of the `strftime()` function.

The behavior of the language information function defined in the `nl_langinfo()` function is also affected by settings of the `category` parameter.

The `locale` parameter points to a character string containing the required setting of the `category` parameter. The following values of `locale` are defined for all settings of `category`:

- **C** Specifies the minimal environment for C-language translation. If `setlocale()` is not invoked, the C locale is the default. Operational behavior within the C locale is defined separately for each interface function that is affected by the locale string.

- **“”** Specifies a native environment, corresponding to the value of the associated environment variables.

In all cases, the `setlocale()` function first checks the value of the corresponding environment variable and if valid, `setlocale()` sets the specified category of the international environment to that value and returns the string corresponding to the locale set (that is, the value of the environment variable, not """). If the value is invalid, `setlocale()` returns a null pointer and the international environment is not changed by this function call.

If the environment variable corresponding to the specified category is not set or is set to the empty string, and the `LANG` environment variable is set and valid, then `setlocale()` sets the category to the corresponding value of `LANG`. If the `LANG` environment variable is not set, the `setlocale()` function uses a system-wide default.

To set all categories in the international environment, the `setlocale()` function is invoked in the following manner:

```c
setlocale (LC_ALL, "");
```

To satisfy this request, the `setlocale()` function first checks all the environment variables. If any environment variable is invalid, `setlocale()` returns a null pointer and the international environment is not
changed by this function call. If all the relevant environment variables are valid, setlocale() sets the international environment to reflect the values of the environment variables. The categories are set in the following order:

- LC_CTYPE
- LC_COLLATE
- LC_TIME
- LC_NUMERIC
- LC_MONETARY

Using this scheme, the categories corresponding to the environment variables will override the value of the LANG environment variable for a particular category.

NULL Used to direct setlocale() to query the current internationalized environment and return the name of the locale.

The reentrant version of the setlocale() function, setlocale_r(), stores the string associated with the category for the new locale in the buffer pointed to by the result parameter.

Notes

AES Support Level: Full use (setlocale())

Return Values

If a pointer to a string is given for the locale parameter and the selection can be honored, the setlocale() function returns the string associated with the specified category parameter for the new locale. If the selection cannot be honored, a null pointer is returned and the program locale is unchanged.

If a null pointer for the locale parameter causes the setlocale() function to return the string associated with the category parameter for the program current locale, the program locale is unchanged.

The string returned by the setlocale() function is such that a subsequent call with that string and its associated category restores that part of the program locale. The string returned is not modified by the program, but can be overwritten by a subsequent call to the setlocale() function.

Upon successful completion, the setlocale_r() function returns a value of 0 (zero). Otherwise, -1 is returned and errno is set to indicate the error.
Errors

If the `setlocale_r()` function fails, `errno` may be set to the following value:

[EINVAL] Either `result` is a null pointer, or the selection is invalid.

Related Information

Functions: `atof(3)`, `ctype(3)`, `jctype(3)`, `localeconv(3)`, `nl_langinfo(3)`, `printf(3)`, `scanf(3)`, `strftime(3)`, `string(3)`
setnetent

Purpose Opens and rewinds the networks file

Library Standard C Library (libc.a)

Synopsis
#include <netdb.h>

void setnetent ( int stay_open );

Parameters

stay_open Specifies a value that indicates when to close the networks file. Specifying a value of 0 (zero) closes the networks file after each call to the getnetent( ) function. Specifying a nonzero value leaves the /etc/networks file open after each call.

Description

The setnetent( ) (set network entry) function opens the /etc/networks file and sets the file marker at the beginning of the file.

Return Values

If an error occurs or if the end of the file is reached, the setnetent subroutine returns a null pointer.

Files

/etc/networks Contains official network names.

Related Information

Functions: endnetent(3), getnetbyaddr(3), getnetbyname(3), getnetent(3)
setpgid, setpgrp

Purpose
Sets the process group ID

Synopsis
#include <sys/types.h>
int setpgid (  
    pid_t process_id,  
    pid_t process_group_id );

Parameters
process_id     Specifies the process whose process group ID is to be changed.
process_group_id     Specifies the new process group ID.

Description
The setpgid() function is used either to join an existing process group or to create a new process group within the session of the calling process. The process group ID of a session leader will not change.

The process group ID of the process designated by the process_id parameter is set to the value of the process_group_id parameter. If process_id is 0 (zero), the process ID of the calling process is used. If process_group_id is 0 (zero), the process group ID of the indicated process is used.

This function is implemented to support job control.

Notes
The setpgrp() function is supported by OSF/1 for binary compatibility only.
AES Support Level: Full use (setpgid())

Return Values
Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
Errors

If the `setpgid()` function fails, `errno` may be set to one of the following values:

- **[EINVAL]** The value of the `process_group_id` parameter is less than or equal to 0 (zero), or is not a valid process ID.
- **[EPERM]** The value of the `process_group_id` parameter is a valid process ID, but that process is not in the same session as the calling process.
- **[EPERM]** The process indicated by the `process_id` parameter is a session leader.
- **[EPERM]** The value of the `process_id` parameter matches the process ID of a child process of the calling process and the child process is not in the same session as the calling process.
- **[EPERM]** The value of the `process_group_id` parameter is valid but does not match the process ID of the process indicated by the `process_id` parameter, and there is no process with a process group ID that matches the value of the `process_group_id` parameter in the same session as the calling process.
- **[ESRCH]** The value of the `process_id` parameter does not match the process ID of the calling process or of a child process of the calling process.
- **[EACCES]** The value of the `process_id` parameter matches the process ID of a child process of the calling process and the child process has successfully executed one of the `exec` functions.

Related Information

Functions: `getpid(2)`
setprotoent

**Purpose**  
Opens and rewinds the `/etc/protocols` file

**Library**  
Standard C Library (`libc.a`)

**Synopsis**  
```c
#include <netdb.h>

void setprotoent (  
    int stay_open  
);
```

**Parameters**  
- `stay_open`  
  Indicates when to close the protocols file. Specifying a value of 0 (zero) closes the file after each call to the `getprotoent()` function. Specifying a nonzero value allows the `/etc/protocols` file to remain open after each function.

**Description**  
The `setprotoent()` (set protocol entry) function opens the `/etc/protocols` file and sets the file marker to the beginning of the file.

**Return Values**  
The return value points to static data that is overwritten by subsequent calls.

**Files**  
- `/etc/protocols`  
  Contains the protocol names.

**Related Information**  
Functions:  
- `endprotoent(3)`, `getprotobyname(3)`, `getprotobynumber(3)`, `getprotoent(3)`
setquota

Purpose Enables or disables quotas on a file system

Synopsis int setquota(
    char *special,
    char *file);

Parameters

special Points to the pathname of the block special device on which a mounted file system exits.

file Points to the pathname of a file in the file system pointed to by the special parameter from which to take quotas.

Description

The setquota() function enables and disables disk quotas on a file system. The special parameter specifies a block special device on which a mounted file system currently exists. When the file parameter has a positive value, the file in the file system pointed to by special is the one from which to take the quotas. When file has a null value, quotas are disabled on the file system pointed to by special.

The setquota() function fails unless the calling process has superuser privilege.

Return Values

Upon successful completion, a value of 0 (zero) is returned. Upon failure, a value of -1 is returned and errno is set to indicate the error.

Errors

If the setquota() function fails, errno may be set to one of the following values:

[ENOTDIR] A component of either path prefix is not a directory.
[EINVAL] Either pathname contains a character with its high-order bit set.
[EINVAL] The kernel has not been compiled with the QUOTA option.
[ENAMETOOLONG] A component of either pathname exceeded NAME_MAX characters, or the entire length of either pathname exceeds PATH_NAME characters.
The block special device pointed to by the `special` parameter does not exist.

The file pointed to by the `file` parameter does not exist.

Too many symbolic links were encountered when translating either pathname.

The caller does not have the appropriate privilege.

The `special` parameter does not point to a block device.

The major device number of the block special device pointed to by the `special` parameter is out of range (this indicates no device driver exists for the associated hardware).

The file pointed to by the `file` parameter resides on a read-only file system.

Search permission is denied for a component of either path prefix.

The file pointed to by the `file` parameter resides on a file system different from the one pointed to by the `special` parameter.

The file pointed to by the `file` parameter is not a plain file.

An I/O error occurred while reading quotas from or writing quotas to the file pointed to by the `file` parameter.

The `file` or `special` parameter points outside allocated address space accessible by the process.

**Related Information**

Functions: `quotactl(2)`
setregid

Purpose
Sets the real and effective group ID.

Synopsis
```
setregid(
    int rgid,
    int egid);
```

Parameters
- `rgid`: Specifies the new real group ID.
- `egid`: Specifies the new effective group ID.

Description
The `setregid()` function sets the real group ID of the current process to the value specified by the `rgid` parameter, and sets the effective group ID to the value specified by the `egid` parameter.

Unprivileged users may change the effective group ID to the real group ID; only the superuser may make other changes.

Supplying a value of -1 for either the real or effective group ID forces the system to substitute the current ID in place of the -1 parameter.

Return Values
- Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors
If the `setregid()` function fails, `errno` may be set to the following value:

- [EPERM] The current process does not have superuser privilege and a change other than changing the effective group ID to the real group ID was specified.

Related Information
Functions: `getgid(2), setgid(2), setrgid(3), setreuid(2)`
setreuid

Purpose
Sets real and effective user ID's

Synopsis
setreuid(
    int ruid,
    int euid);

Parameters
ruid Specifies the new real user ID.
euid Specifies the new effective user ID.

Description
The setreuid() function sets the real and effective user ID's of the current process to the values specified by the ruid and euid parameters. If ruid or euid is -1, the current uid is filled in by the system.

Unprivileged users may change the effective user ID to the real user ID; only processes with superuser privilege may make other changes. This is normally done by the system's authentication program (for example, login), but is not done for system daemons.

Return Values
Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors
If the setreuid() function fails, errno may be set to the following value:

[EPERM] The current process is not the superuser and a change other than changing the effective user ID to the real user ID was specified.

Related Information
Functions: getuid(2), setgid(2), setregid(2), setruid(3)
setrgid, setegid

Purpose
Sets the process group IDs

Library
Standard C Library (libc.a)

Synopsis
#include <sys/types.h>
int setrgid (            
    gid_t rgid );
int setegid (            
    gid_t egid );

Parameters
    rgid          Specifies the value of the real group ID to be set.
    egid          Specifies the value of the effective group ID to be set.

Description
The setegid() function sets the process' effective group ID to the value of the egid parameter if the egid parameter is equal to the current real, effective, or saved group ID.

The setrgid() function sets the process' real group ID to the value of the rgid parameter.

Only the superuser may change the real or effective group ID to a value other than the current real or saved group ID of the process.

Return Values
Upon successful completion, the setegid() and setrgid() functions return a value of 0 (zero). If the either function fails, a value of -1 is returned and errno is set to indicate the error.
Errors

If the `setrgid()` or `setegid()` function fails, `errno` may be set to one of the following values:

[EPERM] The `rgid` or `egid` parameter is not equal to either the real or saved group IDs of the process and the calling process does not have superuser privilege.

Related Information

Functions: `getgroups(2)`, `setgroups(2)`, `setregid(2)`
Commands: `setgroups(1)`
setruid, seteuid

Purpose

Sets the process user IDs

Library

Standard C Library (libc.a)

Synopsis

#include <sys/types.h>

int setruid(
    uid_t ruid);

int seteuid(
    uid_t euid);

Parameters

*euid* Specifies the effective user ID to set.

*ruid* Specifies the real user ID to set.

Description

The *setruid()* and *seteuid()* functions reset the process' real and effective user IDs, respectively.

A process with superuser privilege can set either ID to any value. An unprivileged process can only set the effective user ID if the *euid* parameter is equal to either the real, effective, or saved user ID of the process. An unprivileged process cannot set the real user ID.

Return Values

Upon successful completion, the *seteuid()* and *setruid()* functions return a value of 0 (zero). Otherwise, a value of -1 is returned and *errno* is set to indicate the error.
Errors

If the `seteuid()` or `setruid()` function fails, `errno` may be set to the following value:

[EPERM] The `euid` parameter is not equal to either the real or saved user IDs of the process and the calling process does not have appropriate privilege.

Related Information

Functions: `getuid(2), setreuid(2)`
setservent

**Purpose**

Gets service file entry

**Library**

Standard C Library (libc.a)

**Synopsis**

```c
#include <netdb.h>

void setservent (
    int stay_open);
```

**Parameters**

- `stay_open` Indicates when to close the services file. Specifying a value of 0 (zero) closes the file after each call to the `getservent()` function. Specifying a nonzero value allows the file to remain open after each call.

**Description**

The `setservent()` (set service entry) function opens the `/etc/services` file and sets the file marker at the beginning of the file.

**Return Values**

If an error occurs or the end of the file is reached, the `setservent()` function returns a null pointer.

**Files**

- `/etc/services` Contains service names.

**Related Information**

Functions: `endprotoent(3)`, `getprotobyname(3)`, `getprotobynumber(3)`, `getprotoent(3)`, `getservbyname(3)`, `getservbyport(3)`, `getservent(3)`, `setprotoent(3)`
setsid

Purpose

Sets the process group ID

Synopsis

```
#include <unistd.h>
#include <sys/types.h>
pid_t setsid( void );
```

Description

The `setsid()` function creates a new session when the calling process is not a process group leader. The calling process then becomes the session leader of this session, becomes the process leader of the new process group, and has no controlling terminal. The process group ID of the calling process is set equal to its process ID. The calling process becomes the only process in the new process group and the only process in the new session.

Notes

AES Support Level: Full use

Return Values

Upon successful completion, the value of the new process group ID is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `setsid()` function fails, `errno` may be set to the following value:

```
[EPERM] The calling process is already the process group leader, or the process group ID of a process other than the calling process matches the process ID of the calling process.
```

Related Information

Functions: `getpid(2)`, `setpgid(2)`
setsockopt

**Purpose**
Sets socket options

**Synopsis**
```
#include <sys/types.h>
#include <sys/socket.h>

int setsockopt (  
    int socket,  
    int level,  
    int option_name,  
    char *option_value,  
    int option_len );
```

**Parameters**

- **socket**
  Specifies the unique socket name.

- **level**
  Specifies the protocol level at which the option resides. To set options at the socket level, specify the `level` parameter as `SOL_SOCKET`. To set options at other levels, supply the appropriate protocol number for the protocol controlling the option. For example, to indicate that an option will be interpreted by the TCP protocol, set `level` to the protocol number of TCP, as defined in the `netinet/in.h` file or as determined by using the `getprotobyname()` function.

- **option_name**
  Specifies the option to set. The `option_name` parameter and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The `sys/socket.h` header file defines the socket level options. The socket level options can be enabled or disabled. The options are:

  - **SO_DEBUG**
    Turns on recording of debugging information. This option enables or disables debugging in the underlying protocol modules. This option takes an `int` value.

  - **SO_ACCEPTCONN**
    Enables or disables socket listening. This option takes an `int` value.
SO_BROADCAST
   Permits sending of broadcast messages. This option takes an int value.

SO_REUSEADDR
   Specifies that the rules used in validating addresses supplied by a bind() function should allow reuse of local addresses. This option takes an int value.

SO_KEEPALIVE
   Keeps connections active. Enables the periodic transmission of messages on a connected socket. If the connected socket fails to respond to these messages, the connection is broken and processes using that socket are notified with a SIGPIPE signal.

SO_DONTROUTE
   Indicates that outgoing messages should bypass the standard routing facilities. Instead, they are directed to the appropriate network interface according to the network portion of the destination address.

SO_USELOOPBACK
   Valid only for routing sockets. Determines if a sending socket receives a copy of its own message.

SO_LINGER
   Lingers on a close() function if data is present. This option controls the action taken when unsent messages queue on a socket and a close() function is performed. If SO_LINGER is set, the system blocks the process during the close() function until it can transmit the data or until the time expires. If SO_LINGER is not specified and a close() function is issued, the system handles the call in a way that allows the process to continue as quickly as possible. This option takes a struct linger value, defined in the sys/socket.h header file, to specify the state of the option and linger interval.

SO_OOBINLINE
   Leaves received out-of-band data (data marked urgent) in line. This option takes an int value.

SO_SNDBUF
   Sets send buffer size. This option takes an int value.
OSF/1 Programmer's Reference

setsockopt(2)

SO_RCVBUF
Sets receive buffer size. This option takes an int value.

SO_SNDFLOWAT
Sets send low-water mark. This option takes an int value.

SO_RCVFLOWAT
Sets receive low-water mark. This option takes an int value.

SO_SNDFTIMEO
Sets send time out. This option takes an int value.

SO_RCVTIMEO
Sets receive time out. This option takes an int value.

Options at other protocol levels vary in format and name.

option_value
To enable a Boolean option, set the option_value parameter to a nonzero value. To disable an option, set the option_value parameter to 0 (zero).

option_len
The option_len parameter contains the size of the buffer pointed to by the option_value parameter.

Description

The setsockopt() function sets options associated with a socket. Options may exist at multiple protocol levels. The SO_ options are always present at the uppermost socket level.

The setsockopt() function provides an application program with the means to control a socket communication. An application program can use the setsockopt() function to enable debugging at the protocol level, allocate buffer space, control timeouts, or permit socket data broadcasts. The sys/socket.h file defines all the options available to the setsockopt() function.

When setting socket options, specify the protocol level at which the option resides and the name of the option.

Use the option_value and option_len parameters to access option values for the setsockopt() function. These parameters identify a buffer in which the value for the requested option or options is returned.
Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the setsockopt() function fails, errno may be set to one of the following values:

[EBADF] The socket parameter is not valid.
[ENOTSOCK] The socket parameter refers to a file, not a socket.
[ENOPROTOOPT] The option is unknown.
[EFAULT] The option_value parameter is not in a readable part of the user address space.

Related Information

Functions: bind(2), endprotoent(3), getsockopt(2), getprotobynumber(3), getprotoent(3), setprotoent(3), socket(2)
Purpose
Sets the user ID

Synopsis
#include <sys/types.h>
int setuid (uid_t uid);

Parameters
uid
Specifies the new user ID.

Description
The setuid() function sets the real user ID, effective user ID, and the saved set user ID to the uid parameter.

To change the real user ID, the effective user ID, and the saved set user ID, the calling process must have superuser privilege. If the process does not have appropriate privilege, but the uid parameter is equal to the real user ID or the saved set user ID, the setuid() function sets the effective user ID to the uid parameter; the real user ID and saved set user ID remain unchanged.

Notes
AES Support Level: Full use

Return Values
Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
Errors

If the `setuid()` function fails, `errno` may be set to one of the following values:

- **[EINVAL]** The value of the `uid` parameter is invalid and not supported by the implementation.
- **[EPERM]** The process does not have superuser privileges, and the `uid` parameter does not match the real user ID or the saved set user ID.

Related Information

Functions: `exec(2)`, `getuid(2)`, `setreuid(2)`
shmat

Purpose
Attaches a shared memory region

Synopsis
```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
char *shmat(
    int shmid,
    caddr_t *addr,
    int flags);
```

Parameters

- **shmid**: Specifies the ID for the shared memory region. The ID is typically returned by a previous `shmget()` function.
- **addr**: Specifies the virtual address at which the process wants to attach the shared memory region. The process can also specify 0 (zero) to have the kernel select an appropriate address.
- **flags**: Specifies the attach flags. Possible values are:
  - `SHM_RND`: If the `addr` parameter is not 0 (zero), the kernel rounds off the address, if necessary.
  - `SHM_RDONLY`: If the calling process has read permission, the kernel attaches the region for reading only.

Description

The `shmat()` function attaches the shared memory region identified by the `shmid` parameter to the virtual address space of the calling process. For the `addr` parameter, the process can specify either an explicit address or 0 (zero), to have the kernel select the address. If an explicit address is used, the process can set the `SHM_RND` flag to have the kernel round off the address, if necessary.
Access to the shared memory region is determined by the operation permissions in the `shm_perm.mode` member in the region's `shmid_ds` structure. The low-order bits in `shm_perm.mode` are interpreted as follows:

- 00400  Read by user
- 00200  Write by user
- 00040  Read by group
- 00020  Write by group
- 00004  Read by others
- 00002  Write by others

The calling process is granted read and write permissions on the attached region if at least one of the following is true:

- The effective user ID of the process is superuser.
- The effective user ID of the process is equal to `shm_perm.cuid` or `shm_perm.uid` and bit 0600 in `shm_perm.mode` is set.
- The effective group ID of the process is equal to `shm_perm.cgid` or `shm_perm.gid` and bit 0060 in `shm_perm.mode` is set.
- Bit 0006 in `shm_perm.mode` is set.

If the process has read permission, it can attach the region as read only by setting the SHM_RDONLY flag.

**Return Values**

Upon successful completion, the starting address for the attached region is returned. If the `shmat()` function fails, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `shmat()` function fails, the shared memory region is not attached and `errno` may be set to one of the following values:

- [EACCES]  The calling process does not have the appropriate privilege.
- [ENOMEM]  There was not enough data space available to attach the shared memory region.
The `shmid` parameter does not specify a valid shared memory region ID; the `addr` parameter is not 0 (zero) and not a valid address; or the `addr` parameter is not 0 (zero) and not a valid address, and SHM_RND is not set.

An attempt to attach a shared memory region exceeded the maximum number of attached regions allowed for any one process.

**Related Information**

Functions: `exec(2)`, `exit(2)`, `fork(2)`, `shmctl(2)`, `shmdt(2)`, `shmget(2)`

Data structures: `shmid_ds(4)`
shmctl

**Purpose**
Performs shared memory control operations

**Synopsis**
```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

**Parameters**

- **shmid**
  Specifies the ID of the shared memory region.

- **cmd**
  Specifies the type of command. The possible commands and the operations they perform are as follows:

  **IPC_STAT**
  Queries the shared memory region ID by copying the contents of its associated `shmid_ds` data structure into the `buf` structure.

  **IPC_SET**
  Sets the shared memory region ID by copying values found in the `buf` structure into corresponding fields in the `shmid_ds` structure associated with the shared memory region ID. This is a restricted operation. The effective user ID of the calling process must be equal to that of superuser or equal to the value of `shm_perm.cuid` or `shm_perm.uid` in the associated `shmid_ds` structure.

  **IPC_RMID**
  Removes the shared memory region ID and deallocates its associated `shmid_ds` structure. This is a restricted operation. The effective user ID of the calling process must be equal to that of superuser or equal to the value of `shm_perm.cuid` or `shm_perm.uid` in the associated `shmid_ds` structure.

- **buf**
  Specifies the address of a `shmid_ds` structure. This structure is used only with the IPC_STAT and IPC_SET commands. With IPC_STAT, the results of the query are copied to this structure. With IPC_SET, the values in this structure are used to set the
corresponding fields in the `shmid_ds` structure associated with the shared memory region ID. In either case, the calling process must have allocated the structure before making the call.

**Description**

The `shmctl()` function allows a process to query or set the contents of the `shmid_ds` structure associated with the specified shared memory region ID. It also allows a process to remove the shared memory region’s ID and its associated `shmid_ds` structure. The `cmd` value determines which operation is performed.

The IPC_SET command uses the user-supplied contents of the `buf` structure to set corresponding fields in the `shmid_ds` structure associated with the shared memory region ID. The fields are set as follows:

- The `shm_perm.uid` field is set to the owner’s user ID.
- The `shm_perm.gid` field is set to the owner’s group ID.
- The `shm_perm.mode` field is set to the access modes for the shared memory region. Only the low-order nine bits are set.

**Return Values**

Upon successful completion, a value of 0 (zero) is returned. If the `shmctl()` function fails, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `shmctl()` function fails, `errno` may be set to one or more of the following values:

- **[EINVAL]** The `shmid` parameter does not specify a valid shared memory region ID, or `cmd` is not a valid command.
- **[EACCES]** The `cmd` parameter is IPC_STAT, but the calling process does not have read permission.
- **[EPERM]** The `cmd` parameter is equal to either IPC_RMID or IPC_SET, and the calling process does not have appropriate privilege.
- **[EFAULT]** The `cmd` parameter is IPC_STAT or IPC_SET. An error occurred in accessing the `buf` structure.

**Related Information**

- Functions: `shmat(2), shmdt(2), shmget(2)`
- Data structures: `shmid_ds(4)`
shmdt

**Purpose**  
Detaches a shared memory region

**Synopsis**
```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmdt(
    caddr_t *addr);
```

**Parameters**
- `addr` Specifies the starting virtual address for the shared memory region to be detached. This is the address returned by a previous `shmat()` call.

**Description**
The `shmdt()` function detaches the shared memory region at the address specified by the `addr` parameter. Other instances of the region attached at other addresses are unaffected.

**Return Values**
Upon successful completion, the `shmdt()` function returns 0 (zero). Upon failure, -1 is returned and `errno` is set to indicate the error.

**Errors**
If the `shmdt()` function fails, the shared memory segment is not detached and `errno` may be set to the following value:
- [EINVAL] The `addr` parameter does not specify the starting address of a shared memory region.

**Related Information**
- Functions: `shmat(2), shmctl(2), shmget(2)`
- Data structures: `shmid_ds(4)`
shmget

**Purpose**
Returns (and possibly creates) the ID for a shared memory region.

**Synopsis**
```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(
    key_t key,
    u_int size,
    u_int flags);
```

**Parameters**
- **key**
  Specifies the key that identifies the shared memory region. The IPC_PRIVATE key can be used to assure the return of a new, unused, shared memory region.

- **size**
  Specifies the minimum number of bytes to allocate for the region.

- **flags**
  Specifies the creation flags. Possible values are:
  - IPC_CREATE
    - If the key does not exist, the `shmget()` function creates a shared memory region using the given key. If the key does exist, it forces an error notification.
  - IPC_EXCL
    - If the key already exists, the `shmget()` function fails and returns an error notification.

  The low-order nine bits of `shm_perm.mode` are set equal to the low-order nine bits of `flags`.

**Description**
The `shmget()` function returns (and possibly creates) the ID for the shared memory region identified by the `key` parameter. If IPC_PRIVATE is used for the `key` parameter, the `shmget()` function returns the ID for a private (that is, newly created) shared memory region. The `flags` parameter supplies creation options for the `shmget()` function. If the `key` parameter does not already exist, the IPC_CREAT flag instructs the `shmget()` function to create a new shared memory region for the key and return the kernel-assigned ID for the region.
After creating a new shared memory region ID, the `shmget()` function initializes the `shmid_ds` structure associated with the ID as follows:

- The `shm_perm.cuid` and `shm_perm.uid` fields are set equal to the effective user ID of the calling process.
- The `shm_perm.cgid` and `shm_perm.gid` fields are set equal to the effective group ID of the calling process.
- The low-order nine bits of the `shm_perm.mode` field are set equal to the low-order nine bits of `flags`.
- The `shm_segsz` field is set equal to `size`.
- The `shm_lpid`, `shm_nattch`, `shm_atime`, and `shm_dtime` fields are all set equal to 0 (zero).
- The `shm_ctime` field is set equal to the current time.
- The `shm_cpid` field is set to the process ID of the calling process.

**Return Values**

Upon successful completion, a shared memory identifier is returned. If the `shmget()` function fails, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `shmget()` function fails, `errno` may be set to one of the following values:

- **[EINVAL]** The value of the `size` parameter is less than the system-defined minimum or greater than the system-defined maximum. Or, a shared memory region ID already exists for the `key` parameter, but the number of bytes allocated for the region is less than `size` and `size` is not equal to 0 (zero).
- **[EACCES]** A shared memory region ID already exists for the `key` parameter, but operation permission as specified by the low-order nine bits of the `flags` parameter was not granted.
- **[ENOENT]** A shared memory region ID does not exist for the `key` parameter, and IPC_CREAT was used for the `flags` parameter.
- **[ENOSPC]** An attempt to create a new shared memory region ID exceeded the system-wide limit on the maximum number of IDs allowed.
shmget(2)

**[ENOMEM]** An attempt was made to create a shared memory region ID and its associated `shmid_ds` structure, but there was not enough physical memory available.

**[EEXIST]** A shared memory region ID already exists for the `key` parameter, but IPC_CREAT and IPC_EXCL were used for the `flags` parameter.

**Related Information**

Functions: `shmat(2), shmdt(2), shmdt(2)`

Data structures: `shmid_ds(4)`
shutdown

**Purpose**  Shuts down socket send and receive operations

**Synopsis**

```c
int shutdown (  
  int socket,  
  int how );
```

**Parameters**

- `socket`: Specifies the file descriptor of the socket.
- `how`: Specifies the type of shutdown. Values are:
  - 0: To disable further receive operations
  - 1: To disable further send operations
  - 2: To disable further send operations and receive operations

**Description**

The `shutdown()` function disables receive and/or send operations on the specified socket.

**Return Values**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `shutdown()` function fails, `errno` may be set to one of the following values:

- [EBADF]  The `socket` parameter is not valid.
- [ENOTSOCK]  The `socket` parameter refers to a file, not a socket.

**Related Information**

Functions: `getsockopt(2)`, `read(2)`, `recv(2)`, `recvfrom(2)`, `recvmsg(2)`, `select(2)`, `send(2)`, `sendto(2)`, `setsockopt(2)`, `socket(2)`, `write(2)`
sigaction, signal

**Purpose**
Specifies the action to take upon delivery of a signal

**Synopsis**
```c
#include <signal.h>
int sigaction (  
    int signal,  
    const struct sigaction *action,  
    struct sigaction *o_action );

void (*signal(  
    int signal,  
    void (*function)( int ) ))( int );
```

**Parameters**
- **signal**: Defines the signal.
- **action**: Points to a `sigaction` structure that describes the action to be taken upon receipt of the `signal` parameter.
- **o_action**: Points to a `sigaction` structure in which the signal action data in effect at the time the `sigaction()` function is returned.
- **function**: Specifies the action associated with a signal.

**Description**
The `sigaction()` function allows the calling process to examine and/or change the action to be taken when a specific signal is delivered to the process issuing this function.

The `signal` parameter specifies the signal. If the `action` parameter is not null, it points to a `sigaction` structure that describes the action to be taken on receipt of the `signal` parameter signal. If the `o_action` parameter is not null, it points to a `sigaction` structure in which the signal action data in effect at the time of the `sigaction()` call is returned. If the `action` parameter is null, signal handling is unchanged; thus, the call can be used to inquire about the current handling of a given signal.
The `sigaction` structure has the following members:

```c
void (*sa_handler)( );
sigset_t sa_mask;
int sa_flags;
```

The `sa_handler` field can have the SIG_DFL or SIG_IGN value, or can point to a function. A SIG_DFL value requests default action to be taken when the signal is delivered. A value of SIG_IGN requests that the signal have no effect on the receiving process. A pointer to a function requests that the signal be caught; that is, the signal should cause the function to be called. These actions are more fully described in the `signal.h` file.

The `sa_mask` field can be used to specify that individual signals, in addition to those in the process signal mask, be blocked from being delivered while the signal handler function specified in `sa_handler` is executing. The `sa_flags` field can have the SA_ONSTACK, SA_RESTART, or SA_NOCLDSTOP bits set to specify further control over the actions taken on delivery of a signal.

If the SA_ONSTACK bit is set, the system runs the signal-catching function on the signal stack specified by the `sigstack()` function. If this bit is not set, the function runs on the stack of the process to which the signal is delivered.

If the `signal` parameter is SIGCHLD and a child process of the calling process stops, a SIGCHLD signal will be sent to the calling process if and only if SA_NOCLDSTOP is not set for SIGCHLD.

If a signal for which a signal-catching function exists is sent to a process while that process is executing certain system calls, the call can be restarted if the SA_RESTART bit is set. The affected system calls are the `read()` and `write()` functions on a slow device (such as a terminal, but not a regular file) and the `wait()` function. If SA_RESTART is not set, and such a system call is interrupted by a signal which is caught, then the system call returns -1 and sets `errno` to [EINTR].

The `signal` parameter can be any one of the signal values defined in the `signal.h` header file, except SIGKILL.

The `signal()` function is provided for compatibility with older versions of UNIX operating systems. It sets the action associated with a signal. The `function` parameter can have the same values that are described for the `sa_handler` field in the `sigaction` structure of the `sigaction()` function. However, no signal handler mask or flags can be specified.

The effect of calling the `signal()` function differs in some details depending on whether the calling program is linked with either of the special libraries `libbsd` or `libsys5`. If neither library is used, the behavior is the same as that of the
sigaction() function with all flags set to 0 (zero). If the libbsd library is used (through compilation with the -lbsd switch), the behavior is the same as that of the sigaction() function with the SA_RESTART flag set. If the libsys5 library is used (though compilation with the -lsys5 switch), then the specified signal is not blocked from delivery when the handler is entered, and the disposition of the signal reverts to SIG_DFL when the signal is delivered. See the OSF/1 Applications Programmer's Guide for details on these switches.

Notes

In a multi-threaded environment, the sigaction() function should only be used for the synchronous signals.

The sigvec() and signal() functions are provided for compatibility to old UNIX systems; their function is a subset of that available with the sigaction() function.

AES Support Level: Full use

Return Values

Upon successful completion of the sigaction() function, a value of 0 (zero) is returned. If the sigaction() function fails, a value of -1 is returned and errno is set to indicate the error.

Upon successful completion of a signal() function, the value of the previous signal action is returned. If the call fails, a value of -1 is returned and errno is set to indicate the error as in the sigaction() call.

Errors

If the sigaction() function fails, no new signal handler is installed and errno may be set to one of the following values:

[EFAULT] The action or o_action parameter points to a location outside of the allocated address space of the process.

EINVAL] The signal parameter is not a valid signal number.

EINVAL] An attempt was made to ignore or supply a handler for the SIGKILL, SIGSTOP, and SIGCONT signals.
Related Information

Functions: acct(2), exit(2), kill(2), pause(3), ptrace(2), setjmp(3), sigblock(2), sigpause(3), sigprocmask(2), sigstack(2), sigsuspend(2), sigvec(2), umask(2), wait(2)

Commands: kill(1)

Files: signal(4)
sigblock

**Purpose**

Provides a compatibility interface to the `sigprocmask` function

**Library**

Standard C Library (`libc.a`)

**Synopsis**

```c
int sigblock(
    int mask);
```

**Parameters**

- `mask`  
  Specifies the signals to be added to the set of signals currently being blocked from delivery.

**Description**

The `sigblock()` function causes the signals specified by the `mask` parameter to be added to the set of signals currently being blocked from delivery. The signals are blocked from delivery by logically ORing the `mask` parameter into the signal mask of the process. Signal `i` is blocked if the `i`-th bit in the `mask` parameter is a value of 1. Only signals with values 1-31 can be masked with the `sigblock()` function.

**Notes**

It is not possible to block SIGKILL. The system provides no indication of this restriction.

The `sigblock()` function is provided for compatibility to other UNIX systems. Its function is a subset of the `sigprocmask()` function.

**Return Values**

On completion, the previous set of masked signals is returned.

**Related Information**

Functions: `kill(2)`, `sigaction(2)`, `sigpause(3)`, `sigprocmask(2)`, `sigsuspend(2)`, `sigvec(2)`
sigemptyset, sigfillset, sigaddset, sigdelset, sigismember

Purpose  Creates and manipulates signal masks

Library  Standard C Library (libc.a)

Synopsis  
#include <signal.h>

int sigemptyset (  
    sigset_t *set);

int sigfillset (  
    sigset_t *set);

int sigaddset (  
    sigset_t *set,  
    int sig_number);

int sigdelset (  
    sigset_t *set,  
    int sig_number);

int sigismember (  
    sigset_t *set,  
    int sig_number);

Parameters  
set Specifies the signal set.
sig_number Specifies the individual signal.

Description  
The sigemptyset(), sigfillset(), sigaddset(), sigdelset(), and sigismember()  
functions manipulate sets of signals. These functions operate on data objects that  
can be addressed by the application, not on any set of signals known to the system,  
such as the set blocked from delivery to a process or the set pending for a process.
The `sigemptyset()` function initializes the signal set pointed to by the `set` parameter such that all signals are excluded. The `sigfillset()` function initializes the signal set pointed to by the `set` parameter such that all signals are included. A call to either the `sigfillset()` or `sigemptyset()` function must be made at least once for each object of the type `sigset_t` prior to any other use of that object.

The `sigaddset()` and `sigdelset()` functions respectively add and delete the individual signal specified by the `sig_number` parameter from the signal set specified by the `set` parameter. The `sigismember()` function tests whether the `sig_number` parameter is a member of the signal set pointed to by the `set` parameter.

**Notes**

**AES Support Level:** Full use

**Example**

To generate and use a signal mask that blocks only the SIGINT signal from delivery, enter:

```c
#include <signal.h>
int return_value;
sigset_t newset;
sigset_t *newset_p;
...
newset_p = &newset;
    sigemptyset(newset);
sigaddset(newset, SIGINT);
return_value = sigprocmask (SIG_SETMASK, newset_p, NULL);
```

**Return Values**

Upon successful completion, the `sigismember()` function returns a value of 1 if the specified signal is a member of the specified set, or a value of 0 (zero) if it is not. Upon successful completion, the other functions return a value of 0. For all the preceding functions, if an error is detected, a value of -1 is returned and `errno` is set to indicate the error.
Errors

If the `sigfillset()`, `sigdelset()`, `sigismember()`, or `sigaddset()` function fails, `errno` may be set to the following value:

[EINVAL] The value of the `sig_number` parameter is not a valid signal number.

Related Information

Functions: `sigaction(2)`, `sigprocmask(2)`, `sigsuspend(2)`, `sigvec(2)`

Files: `signal(4)`
siginterrupt

Purpose

Allows signals to interrupt functions

Library

Berkeley Compatibility Library (libbsd.a)

Synopsis

```c
int siginterrupt(
    int sig,
    int flag);
```

Parameters

- `sig`: Specifies the expected interrupt signal.
- `flag`: Indicates whether the function is to restart when interrupted by the specified signal. When the `flag` parameter is TRUE, restart is disabled. When the `flag` parameter is FALSE, restart is enabled.

Description

The `siginterrupt()` function is used to change the restart behavior of a system call when it is interrupted by a signal specified by the `sig` parameter. When the `flag` parameter is FALSE (0), system calls restart when they are interrupted by the `sig` signal and no data has yet been transferred.

When the `flag` parameter is TRUE (1), restart of system calls is disabled. When a system call is interrupted by the `sig` signal and no data has been transferred, the function returns a value of -1 with `errno` set to [EINTR]. Otherwise, interrupted system calls that have started transferring data return a value that is the number of data bytes actually transferred.

System call interrupt is the default behavior unless the calling program has been linked with the `libbsd` library.

Notes

The use of the `siginterrupt()` function does not affect signal-handling semantics in any other way. Programs may switch between restartable and interruptible system call operation as often as desired in the execution of a program.

Issuing a `siginterrupt()` call during the execution of a signal handler causes the new action to take place when the next instance of the specified signal is caught.
The `siginterrupt()` function is provided for compatibility with BSD systems, and programs that use it should be linked with the `libbsd` library. The recommended method for controlling whether a signal is restartable or interruptible is to use the `sigaction()` function.

**Return Values**

Upon successful completion, `siginterrupt()` returns a value of 0 (zero). Otherwise, a value of -1 is returned to indicate that an invalid signal value has been used.

**Errors**

If the `siginterrupt()` function fails, `errno` may be set to the following value:

- `[EINVAL]` The value of the `sig` parameter does not represent a valid signal.

**Related Information**

Functions: `sigaction(2)`, `sigprocmask(2)`, `sigsuspend(2)`
siglongjmp

Purpose
Nonlocal goto with signal handling

Library
Standard C Library (libc.a)

Synopsis
#include <setjmp.h>

void siglongjmp (
    sigjmp_buf env,
    int value );

Parameters
env Specifies an address for a sigjmp_buf structure.
value Specifies any nonzero value.

Description
The siglongjmp() function restores the environment saved by the most recent sigsetjmp() function in the same process with the corresponding sigjmp_buf parameter.

All accessible objects have values as of the time siglongjmp() was called, except that the values of objects of automatic storage duration that have been changed between the sigsetjmp() call and siglongjmp() call are indeterminate.

As it bypasses the usual function call and return mechanisms, the siglongjmp() function executes correctly in contexts of interrupts, signals, and any of their associated functions. However, if the siglongjmp() function is invoked from a nested signal handler (that is, from a function invoked as a result of a signal raised during the handling of another signal), the behavior is undefined.

The siglongjmp() function restores the saved signal mask if and only if the env parameter was initialized by a call to the sigsetjmp() function with a nonzero savemask parameter.
Functions

siglongjmp(3)

Notes

AES Support Level: Full use

Return Values

After the `siglongjmp()` function is completed, program execution continues as if the corresponding call of the `sigsetjmp()` function had just returned the value specified by the `value` parameter. The `siglongjmp()` function cannot cause the `sigsetjmp()` function to return 0 (zero); if `value` is 0, the `sigsetjmp()` function returns 1.

Related Information

Functions: `setjmp(3)`, `sigprocmask(2)`, `sigsetjmp(3)`, `sigsuspend(2)`
sigpause

**Purpose**
Provides a compatibility interface to the `sigsuspend` function

**Library**
Standard C Library (`libc.a`)

**Synopsis**
```
#include <signal.h>

int sigpause (int signal_mask);
```

**Parameters**

*signal_mask*
Specifies which signals to block.

**Description**

The `sigpause()` function call blocks the signals specified by the `signal_mask` parameter and then suspends execution of the process until delivery of a signal whose action is either to execute a signal-catching function or to end the process. Signal of value *i* is blocked if the *i*-th bit of the mask is set. Only signals with values 1 to 31 can be blocked with the `sigpause()` function. In addition, the `sigpause()` function does not allow the SIGKILL, SIGSTOP, or SIGCONT signals to be blocked. If a program attempts to block one of these signals, the `sigpause()` function gives no indication of the error.

The `sigpause()` function sets the signal mask and waits for an unblocked signal as one atomic operation. This means that signals cannot occur between the operations of setting the mask and waiting for a signal.

The `sigpause()` function is provided for compatibility with older UNIX systems; its function is a subset of the `sigsuspend()` function.

**Return Values**

If a signal is caught by the calling process and control is returned from the signal handler, the calling process resumes execution after the `sigpause()` function, which always returns a value of -1 and sets `errno` to [EINTR].
If delivery of a signal causes the process to end, the `sigpause()` function does not return.

If delivery of a signal causes a signal-catching function to execute, the `sigpause()` function returns after the signal-catching function returns, with the signal mask restored to the set that existed prior to the `sigpause()` call.

**Related Information**

Functions: `pause(3)`, `sigaction(2)`, `sigblock(2)`, `sigprocmask(2)`, `sigsuspend(2)`, `sigvec(2)`
sigpending

Purpose
Examines pending signals

Synopsis
#include <signal.h>

int sigpending (
    sigset_t *set);

Parameters

set Points to a sigset_t structure.

Description
The sigpending() function stores the set of signals that are blocked from delivery
and pending to the calling process in the object pointed to by the set parameter.

Applications should call either the sigemptyset() or the sigfillset() function at
least once for each object of type sigset_t prior to any other use of that object. If
such an object is not initialized in this way, but is nonetheless supplied as an
argument to the sigpending() function, the results are undefined.

Notes
AES Support Level: Full use

Return Values
Upon successful completion, the sigpending() function returns a value of 0 (zero).
Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors
If the sigpending() function fails, errno may be set to the following value:

[EFAULT] The set parameter points to a location outside the allocated address
space of the process.

Related Information
Functions: sigemptyset(3), sigprocmask(2)
Files: signal(4)
sigprocmask, sigsetmask

**Purpose** Sets the current signal mask

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <signal.h>

int sigprocmask(
    int how,
    sigset_t *set,
    sigset_t *o_set);

int sigsetmask (int signal_mask);
```

**Parameters**

- **how** Indicates the manner in which the set of masked signals is changed; it has one of the following values:
  - **SIG_BLOCK**
    The resulting set is the union of the current set and the signal set pointed to by the `set` parameter.
  - **SIG_UNBLOCK**
    The resulting set is the intersection of the current set and the complement of the signal set pointed to by the `set` parameter.
  - **SIG_SETMASK**
    The resulting set is the signal set pointed to by the `set` parameter.

- **set** Specifies the signal set. If the value of the `set` parameter is not null, it points to a set of signals to be used to change the currently blocked set. If the value of the `set` parameter is null, the value of the `how` parameter is not significant and the process signal mask is unchanged; thus, the call can be used to inquire about currently blocked signals.
sigprocmask(2)

\textit{o_set} \quad \text{If the} \ o_{\text{set}} \ \text{parameter is not the null value, the signal mask in effect at the time of the call is stored in the space pointed to by the} \ o_{\text{set}} \ \text{parameter.}

\textit{signal\_mask} \quad \text{Specifies the signal mask of the process.}

\textbf{Description}

The \textit{sigprocmask()} function is used to examine or change the signal mask of the calling process.

Typically, you would use the \textit{sigprocmask} (SIG\_BLOCK) function to block signals during a critical section of code, and then use the \textit{sigprocmask} (SIG\_SETMASK) function to restore the mask to the previous value returned by the \textit{sigprocmask} (SIG\_BLOCK) function.

If there are any unblocked signals pending after the call to the \textit{sigprocmask()} function, at least one of those signals will be delivered before the \textit{sigprocmask()} function returns.

The \textit{sigprocmask()} function does not allow the SIGKILL or SIGSTOP signals to be blocked. If a program attempts to block one of these signals, the \textit{sigprocmask()} function gives no indication of the error.

The \textit{sigsetmask()} function allows the process signal mask to change for signal values 1 to 31. This same function can be accomplished for all values with the \textit{sigprocmask}(SIG\_SETMASK) function. The signal of value \textit{i} will be blocked if the \textit{i}-th bit of \textit{signal\_mask} parameter is set.

\textbf{Example}

To set the signal mask to block only the SIGINT signal from delivery, enter:

\begin{verbatim}
#include <signal.h>
int return_value;
.sigset_t newset;
.sigset_t *newset_p;
  ...
newset_p = &newset;
sigemptyset(newset_p);
sigaddset(newset_p, SIGINT);
return_value = sigprocmask (SIG_SETMASK, newset_p, NULL);
\end{verbatim}
Notes

AES Support Level: Full use (sigprocmask( ))

Return Values

Upon successful completion, the sigprocmask() function returns a value of 0 (zero). If the sigprocmask() function fails, the signal mask of the process is unchanged, a value of -1 is returned, and errno is set to indicate the error.

Upon successful completion, the sigsetmask() function returns the value of the previous signal mask. If the function fails, a value of -1 is returned.

Errors

If the sigprocmask() function fails, errno may be set to one of the following values:

- [EINVAL] The value of the how parameter is not equal to one of the defined values.
- [EFAULT] The set or o_set parameter points to a location outside the allocated address space of the process.

Related Information

Functions: kill(2), sigaction(2), sigpause(3), sigsuspend(2), sigvec(2)
sigreturn

Purpose

Returns from signal

Synopsis

```
#include <signal.h>

int sigreturn(
    struct sigcontext *scp);
```

Parameters

scp

Points to a sigcontext structure whose members contain the processor state to be restored. The contents of the sigcontext structure should have been previously obtained by entry to a signal handler or by the setjmp() or sigsetjmp() function.

Description

The sigreturn() function restores the processor state of the calling process from a sigcontext structure. The sigcontext structure contains the state of all application-visible registers as well as the signal mask. The specific members of the sigcontext structure depend on the machine architecture. Each machine-dependent structure member is defined in the signal.h include file.

The sigreturn() function is used internally by the system software to restore the processor state on return from a signal handler and from a longjmp() function, to restore the state saved by a previous setjmp() or sigsetjmp() function.

Notes

An application should only use sigreturn() with great caution.

Return Values

Upon successful completion, the sigreturn() function does not return. Otherwise, a value of -1 is returned and errno may be set to indicate the error.
Errors

If the `sigreturn()` function fails, the process context remains unchanged and `errno` is set to one of the following values:

- **EFAULT** The `scp` parameter points to memory space that is not a valid part of the process address space.
- **EINVAL** The `sigcontext` structure contains unsupported or illegal values.

Related Information

Functions: `setjmp(3), sigaction(2), sigvec(2)`
sigset, sighold, sigrelse, sigignore

Purpose
Compatibility interfaces for signal management

Library
Standard C Library (libc.a)

Synopsis
#include<signal.h>

void (*sigset(
    int signal,
    void (*function) ( int ))) ( int )

int sighold(
    int signal );

int sigrelse(
    int signal );

int sigignore(
    int signal );

Parameters

signal Specifies the signal. The signal parameter can be assigned any of the signals defined in the signal.h header file.

function Specifies one of four values: SIG_DFL, SIG_IGN, SIG_HOLD, or an address of a signal-catching function. The function parameter is declared as type pointer to a function returning void. The following actions are prescribed by these values:

SIG_DFL
System default handling of the signal.

SIG_IGN
Ignore signal.

Any pending signal specified by the signal parameter is discarded. A pending signal is a signal that has occurred but for which no action has been taken. The system signal action is set to ignore future occurrences of this signal type.
SIG_HOLD

Hold signal.

The signal specified by the signal parameter is to be held. Any pending signal of this type remains held. Only one signal of each type is held.

(Address of signal-catching function.)

Catch signal.

Upon receipt of the signal specified by the signal parameter, the receiving process is to execute the signal catching function pointed to by the function parameter. Any pending signal of this type is released. This address is retained across calls to the other signal management functions, sighold() and sigrelse(). The signal number signal will be passed as the only argument to the signal-catching function.

Before entering the signal-catching function, the value of function for the caught signal will be set to SIG_HOLD. During normal return from the signal-catching handler, the system signal action is restored to function and any held signal of this type is released. If a nonlocal goto (see the setjmp() function) is taken, the sigrelse() function must be invoked to restore the system signal action and to release any held signal of this type.

Upon return from the signal-catching function, the receiving process will resume execution at the point at which it was interrupted, except for implementation-defined signals where this may not be true.

The signal-catching function will be executed and then the interrupted routine may return a value of -1 to the calling process with errno set to [EINTR] under the following conditions:

- A signal to be caught occurs during a nonatomic operation such as a call to the read(), write(), open(), or ioctl() function on a slow device (such as a terminal).
- A signal to be caught occurs during a pause() or sigsuspend() function.
- A signal to be caught occurs during a wait function that does not return immediately.
Description

The `sigset()`, `sighold()`, `sigrelse()`, and `sigignore()` functions enhance the signal facility and provide signal management for application processes.

The `sigset()` function specifies the system signal action to be taken upon receipt of `signal`.

The `sighold()` and `sigrelse()` functions establish critical regions of code. A call to the `sighold()` function has the effect of deferring or holding a signal until a subsequent call to the `sigrelse()` function. The `sigrelse()` function restores the system signal action to the action that was previously specified by `sigset()`.

The `sigignore()` function sets the action for `signal` to SIG_IGN.

The `signal()` routine should not be used in conjunction with these routines for a particular signal type.

Notes

These interfaces are provided for compatibility only. New programs should use `sigaction()` and `sigprocmask()` to control the disposition of signals.

Return Values

Upon successful completion, the `sigset()` function returns the previous value of the system signal action for the specified `signal`. Otherwise, it returns SIG_ERR and `errno` is set to indicate the error.

For the `sighold()`, `sigrelse()`, and `sigignore()` functions, a value of 0 (zero) is returned upon success. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `sigset()`, `sighold()`, `sigrelse()`, or `sigignore()` function fails, `errno` is set to the following value:

[EINVAL] The `signal` parameter is either an illegal signal number or SIGKILL, or the default handling of `signal` cannot be changed.

Related Information

Functions: `kill(2)`, `setjmp(3)`, `sigaction(2)`, `sigprocmask(2)`, `wait(2)`

Files: `signal(4)`
sigsetjmp

**Purpose**
Sets jump point for a nonlocal goto

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <setjmp.h>

int sigsetjmp (  
        sigjmp_buf env,  
        int savemask );
```

**Parameters**
- `env` Specifies an address for a `sigjmp_buf` structure.
- `savemask` Specifies whether the current signal mask should be saved.

**Description**
The `sigsetjmp()` function saves its calling environment in its `env` parameter for later use by the `siglongjmp()` function.

If the value of the `savemask` parameter is not 0 (zero) the `sigsetjmp()` function will also save the process' current signal mask as part of the calling environment.

**Notes**

AES Support Level: Full use

**Return Values**
If the return is from a successful direct invocation, the `sigsetjmp()` function returns the value 0 (zero). If the return is from a call to the `siglongjmp()` function, the `sigsetjmp()` function returns a nonzero value.

**Related Information**
Functions: `sigaction(2)`, `siglongjmp(3)`, `sigprocmask(2)`, `sigsuspend(2)`
sigstack

Purpose
Sets and gets signal stack context

Synopsis
#include <signal.h>

int sigstack (
    struct sigstack *instack,
    struct sigstack *outstack );

Parameters

instack Specifies the stack pointer of the new signal stack.
If the value of the instack parameter is nonzero, it points to a
sigstack() structure, which has the following members:

    struct sigstack {
        caddr_t ss_sp;
        int ss_onstack;
    }

The value of instack->ss_sp specifies the stack pointer of the new
signal stack. The value of instack->ss_onstack should be set to 1 if
the process is currently running on that stack; otherwise, it should be
0 (zero).
If the value of the instack parameter is 0 (that is, a null pointer), the
signal stack state is not set.

outstack Points to the structure where the current signal stack state is stored.
If the value of the outstack parameter is nonzero, it points to a
sigstack() structure into which the sigstack() function stores the
current signal stack state. If the value of the outstack parameter is 0
(zero), the previous signal stack state is not reported.

Description
The sigstack() function defines an alternate stack on which signals are to be
processed.
When a signal occurs and its handler is to run on the signal stack, the system checks to see if the process is already running on that stack. If so, the process continues to run even after the handler returns. If not, the signal handler runs on the signal stack, and the original stack is restored when the handler returns.

Use the `sigaction()` function to specify whether a given signal handler routine is to run on the signal stack.

Caution

A signal stack does not automatically increase in size as a normal stack does. If the stack overflows, unpredictable results can occur.

Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `sigstack()` function fails, `errno` may be set to the following value:

[EFAULT] The `instack` or `outstack` parameter points outside of the address space of the process.

Related Information

Functions: `setjmp(3), sigaction(2), sigvec(2)`
sigsuspend

Purpose
Atomically changes the set of blocked signals and waits for a signal

Library
Standard C Library (libc.a)

Synopsis
```
#include <signal.h>

int sigsuspend (
    sigset_t *signal_mask);
```

Parameters

*signal_mask* Points to a set of signals.

Description

The *sigsuspend()* function replaces the signal mask of the process with the set of signals pointed to by the *signal_mask* parameter, and then suspends execution of the process until delivery of a signal whose action is either to execute a signal-catch function or to terminate the process. The *sigsuspend()* function does not allow the SIGKILL or SIGSTOP signals to be blocked. If a program attempts to block one of these signals, the *sigsuspend()* function gives no indication of the error.

If delivery of a signal causes the process to terminate, the *sigsuspend()* function does not return. If delivery of a signal causes a signal-catch function to execute, the *sigsuspend()* function returns after the signal-catch function returns, with the signal mask restored to the set that existed prior to the call to the *sigsuspend()* function.

The *sigsuspend()* function sets the signal mask and waits for an unblocked signal as one atomic operation. This means that signals cannot occur between the operations of setting the mask and waiting for a signal. If a program invokes *sigprocmask*(SIG_SETMASK) and *sigpause()* separately, a signal that occurs between these functions might not be noticed by *sigpause()*.
In normal usage, a signal is blocked by using the `sigprocmask(SIG_BLOCK,...)` function at the beginning of a critical section. The process then determines whether there is work for it to do. If no work is to be done, the process waits for work by calling the `sigsuspend()` function with the mask previously returned by the `sigprocmask()` function.

**Notes**

The `sigpause()` function is provided for compatibility with older UNIX systems; its function is a subset of the `sigsuspend()` function.

**AES Support Level:** Full use

**Return Values**

If a signal is caught by the calling process and control is returned from the signal handler, the calling process resumes execution after the `sigsuspend()` function, which always return a value of -1 and sets `errno` to [EINTR].

**Related Information**

Functions: `pause(3)`, `sigaction(2)`, `sigblock(2)`, `sigprocmask(2)`, `sigvec(2)`
sigvec

Purpose Provides a compatibility interface to the sigaction() function

Synopsis

```c
#include <sys/signal.h>

int sigvec (
    int signal,
    struct sigvec *in_vec,
    struct sigvec *out_vec);
```

Parameters

- `signal`: Specifies the signal number.
- `in_vec`: Points to a sigvec() structure that specifies the action to be taken when the specified signal is delivered, the mask to be used when calling the signal handler, and the flags that modify signal behavior.
- `out_vec`: Points to a sigvec() structure that is set to the previous signal action state on successful return from the sigvec() function.

Description

The sigvec() function is provided for compatibility to old UNIX systems; its function is a subset of that available with the sigaction() function. Like the sigaction() function, the sigvec() function allows the user to set the action to take upon the receipt of a signal and to specify a signal handler mask to block signals before calling the signal handler. However, only signals with values 1 to 31 can be masked on entry to a signal-handler set up with the sigvec() function.

The sigvec() structure has the following members:

```c
    void (*sv_handler)( );
    int sv_mask;
    int sv_flags;
```

The sv_handler field specifies the action for the signal, and can be SIG_DFL, SIG_IGN, or the address of a signal handler function. See the sigaction() function for a detailed description of the signal actions.

The sv_mask field specifies a mask which specifies signals to block (in addition to any signals already blocked at time of delivery) when the signal handler function is called for the signal. Signal $i$ is blocked if the $i$-th bit of the mask is set. Only signals with values 1 to 31 can be masked with the sigvec() function. The sv_flags field contains flags that further specify signal behavior. If SV_ONSTACK is set,
the signal handler runs on the signal stack specified by the `sigstack()` function; otherwise, the signal handler runs on the stack of the process receiving the signal. If `SV_INTERRUPT` is set, a system call that is interrupted by `signal` returns a value of -1 with `errno` set to [EINTR]; otherwise, a system call interrupted by `signal` is restarted.

If the value of the `in_vec` parameter is a null pointer, then the signal handler information is not set. If the value of the `out_vec` parameter is null, then the previous signal handler information is not returned.

Once a signal handler is assigned, it remains assigned until another call to the `sigvec()`, `signal()`, `sigaction()`, or `exec` function is made.

Notes

The `sigvec()` function is provided for compatibility only, and its use is not recommended. Programs should use the `sigaction()` function instead.

The `sigvec()` function does not check the validity of the `sv_handler` field pointer. If it points to a location outside of the process address space, the process receives a memory fault when the system attempts to call the signal handler. If the `sv_handler` field points to anything other than a function, the results are unpredictable.

The signal-handler function can be declared as follows:

```c
void handler (  
    int signal );
```

Return Values

Upon successful completion, a value of 0 (zero) is returned. If the `sigvec()` function fails, a value of -1 is returned and `errno` is set to indicate the error.

Errors

If the `sigvec()` function fails, no new signal handler is installed and `errno` may be set to one of the following values:

- **[EFAULT]** The `in_vec` or `out_vec` parameter points to a location outside of the process’ address space.
- **[EINVAL]** The `signal` parameter is not a valid signal number.
- **[EINVAL]** An attempt was made to ignore or supply a handler for the `SIGKILL` signal.
Related Information

Functions: kill(2), ptrace(2), sigaction(2), sigblock(2), sigpause(3), sigstack(2)
sigwait

Purpose
Suspends a calling thread

Library
Threads Library (libpthreads.a)

Synopsis
#include <signal.h>

int sigwait(
    sigset_t *set);

Parameters
set Specifies the set of signals to wait for.

Description
The sigwait() function suspends the calling thread until at least one of the signals in the set parameter is in the threads set of pending signals. When this happens, one of those signals is atomically chosen and removed from the set of pending signals and that signal number is returned.

The effect is unspecified if any signals in the set parameter are not blocked when the sigwait() function is called.

The set parameter is created using the set manipulation functions sigemptyset(), sigfillset(), sigaddset(), and sigdelset().

Return Values
Upon successful completion, the signal number of the pending signal is returned. Otherwise, -1 is returned and errno is set to indicate the error.
Errors

If the `sigwait()` function fails, `errno` may be set to the following value:

[EINVAL] The value of the `set` parameter contains an invalid or unsupported signal number.

Related Information

Functions: `sigaction(2)`, `sigpending(2)`, `sigsuspend(2)`
sin, cos, tan, asin, acos, atan, atan2

Purpose
Computes the trigonometric and inverse trigonometric functions.

Library
Math Library (libm.a)

Synopsis
```
#include <math.h>

double sin (double x);

double asin (double x;

double cos (double x);

double acos (double x);

double tan (double x);

double atan (double x);

double atan2 (double y, x);
```

Parameters
- **x** Specifies some double value.
- **y** Specifies some double value.

Description
The `sin()` function computes the sine of `x`, measured in radians.
The `cos()` function computes the cosine of `x`, measured in radians.
The \texttt{tan()} function computes the tangent of \( x \), measured in radians.

The \texttt{asin()} function computes the principal value of the arc sine of \( x \), in the range \([-\pi/2, \pi/2]\) radians. The value of \( x \) must be in the domain \([-1, 1]\).

The \texttt{acos()} function computes the principal value of the arc cosine of \( x \), in the range \([0, \pi]\) radians. The value of \( x \) must be in the domain \([-1, 1]\).

The \texttt{atan()} function computes the principal value of the arc tangent of \( x \), in the range \([-\pi/2, \pi/2]\) radians.

The \texttt{atan2()} function computes the principal value of the arc tangent of \( y/x \), in the range \([-\pi, \pi]\) radians, using the signs of both arguments to determine the quadrant of the return value.

**Notes**

The \texttt{sin()}, \texttt{cos()}, and \texttt{tan()} functions lose accuracy when passed a large value for the \( x \) parameter.

**AES Support Level:** Full use

**Return Values**

The \texttt{sin()} and \texttt{cos()} functions return the sine and cosine, respectively, of their parameters. If \( x \) is \texttt{NaN}, \texttt{NaN} is returned. Otherwise, either \texttt{errno} is set to indicate an error, or \texttt{NaN} is returned.

The \texttt{tan()} function returns the tangent of its parameter. If \( x \) is \texttt{NaN}, \texttt{NaN} is returned. Otherwise, either \texttt{errno} is set to indicate an error, or \texttt{NaN} is returned.

The \texttt{asin()} function returns the principal value of the arc sine of \( x \). Otherwise, the \texttt{asin()} function returns \texttt{NaN} and sets \texttt{errno} to \texttt{[EDOM]} if its parameters are not in the range \([-1, 1]\).

The \texttt{acos()} function returns the principal value of the arc cosine of \( x \). Otherwise, the \texttt{acos()} function returns \texttt{NaN} and sets \texttt{errno} to \texttt{[EDOM]} if its parameters are not in the range \([-1, 1]\).

The \texttt{atan()} function returns the principal value of the arc tangent of \( x \). If \( x \) is \texttt{NaN}, \texttt{NaN} is returned. Otherwise, \texttt{NaN} is returned.

The \texttt{atan2()} function returns the principal value of the arc tangent of \( y/x \). If \( x \) or \( y \) is \texttt{NaN}, \texttt{NaN} is returned.
Errors

If the `sin()` or `cos()` function fails, `errno` may be set to one of the following values:

- **[EDOM]** The value of `x` is NaN, or `x` is ±HUGE_VAL.
- **[ERANGE]** The magnitude of `x` is such that total or partial loss of significance resulted.

If the `tan()` function fails, `errno` may be set to one of the following values:

- **[ERANGE]** The value to be returned would have caused overflow.
- **[ERANGE]** The value to be returned would have caused underflow, or the magnitude of `x` is such that total or partial loss of significance would result.
- **[EDOM]** The value `x` is NaN.

If the `asin()` or `acos()` function fails, `errno` may be set to the following value:

- **[EDOM]** The `x` parameter is not in the domain [-1,1].

If the `atan()` function fails, `errno` may be set to the following value:

- **[EDOM]** The value of `x` is NaN.

If the `atan2()` function fails, `errno` may be set to the following value:

- **[EDOM]** Both arguments are zero or one of the arguments is NaN.

Related Information

Functions: `isnan(3)`, `sinh(3)`
sinh, cosh, tanh

Purpose
Computes hyperbolic functions

Library
Math Library (libm.a)

Synopsis
#include <math.h>

double sinh (double x);
double tanh (double x);
double cosh (double x);

Parameters
x Specifies some double value.

Description
The sinh(), cosh(), and tanh() functions compute the hyperbolic sine, hyperbolic cosine, and hyperbolic tangent of x, respectively.

Notes
AES Support Level: Full use

Return Values
The sinh() function returns the hyperbolic sine of its parameter. If the result would cause an overflow, HUGE_VAL is returned and errno is set to [ERANGE]. If x is NaN, NaN is returned. Otherwise, ±HUGE_VAL or NaN is returned.

The cosh() function returns the hyperbolic cosine of its parameter. If the result would cause an overflow, HUGE_VAL is returned and errno is set to [ERANGE]. If x is NaN, NaN is returned. Otherwise, either errno is set to indicate the error or NaN is returned.
The `tanh()` function returns the hyperbolic tangent of its parameter. If \( x \) is NaN, NaN is returned. Otherwise, either zero is returned and `errno` is set to indicate the error, or NaN is returned.

**Errors**

If the `sinh()`, `cosh()`, or `tanh()` function fails, `errno` may be set to one of the following values:

- **[EDOM]** The value of \( x \) is NaN.
- **[ERANGE]** The result of the `sinh()` or `cosh()` function would cause an overflow.

**Related Information**

Functions: `isnan(3)`, `sin(3)`
sleep

Purpose  
Suspends execution for an interval

Library  
Standard C Library (libc.a)
Threads Library (libpthreads.a)

Synopsis  
unsigned int sleep(
  unsigned int seconds);

Parameters  
seconds  
Specifies the number of seconds to sleep.

Description  
The sleep() function suspends execution of a process for the interval specified by the seconds parameter. The suspension time may be longer than requested due to the scheduling of other activity by the system.

In a multi-threaded environment, the sleep() function, is redefined so that only the calling thread is suspended.

Notes  
AES Support Level: Full use

Return Values  
If the sleep() function returns because the requested time has elapsed, 0 (zero) is returned. If the sleep() function returns because a signal was caught, the amount of time still remaining to be "slept" is returned.

Related Information  
Functions: alarm(3), pause(3), sigaction(2), sleep(3)
Commands: shutdown(8), wall(1)
socket

**Purpose**  Creates an end point for communication and returns a descriptor

**Synopsis**  
#include <sys/types.h>  
#include <sys/socket.h>  
int socket (  
    int addr_family,  
    int type,  
    int protocol );

**Parameters**  

`addr_family`  Specifies an address family with which addresses specified in later socket operations should be interpreted. The `sys/socket.h` file contains the definitions of the address families. Commonly used families are:

- AF_UNIX  
  UNIX pathnames
- AF_INET  
  ARPA Internet addresses
- AF_NS  
  Xerox Network Software addresses

`type`  Specifies the semantics of communication. The `sys/socket.h` file defines the socket types. The following types are supported:

- SOCK_STREAM  
  Provides sequenced, reliable, two-way byte streams with a transmission mechanism for out-of-band data.
- SOCK_DGRAM  
  Provides datagrams, which are connectionless messages of a fixed maximum length.
- SOCK_RAW  
  Provides access to internal network protocols and interfaces. This type of socket is available only to a process with superuser privilege.


socket(2)

protocol Specifies a particular protocol to be used with the socket. Specifying a protocol of 0 (zero) causes the socket() function to default to the typical protocol for the requested type of returned socket.

Description

The socket() function creates a socket of the specified type in the specified addr_family.

The socket() function returns a descriptor (an integer) that can be used in later system calls that operate on sockets.

Socket level options control socket operations. The getsockopt() and setsockopt() functions are used to get and set these options, which are defined in the sys/socket.h file.

Return Values

Upon successful completion, the socket() function returns a nonnegative integer (the socket descriptor). Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the socket() function fails, errno may be set to one of the following values:

[EAFNOSUPPORT] The addresses in the specified address family are not available in the kernel.

[EPROTONOSUPPORT] The socket in the specified address family is not supported.

[EMFILE] The per-process descriptor table is full.

[ENOBUFS] Insufficient resources were available in the system to complete the call.

[EPERM] The process is attempting to open a raw socket and does not have superuser privilege.
Related Information

Functions: accept(2), bind(2), connect(2), listen(2), getsockname(2), getsockopt(2), recv(2), recvfrom(2), recvmsg(2), send(2), sendto(2), sendmsg(2), setsockopt(2), shutdown(2), socketpair(2)
socketpair

**Purpose**  
Creates a pair of connected sockets

**Synopsis**

```c
#include <sys/types.h>
#include <sys/socket.h>

int socketpair(
    int domain,
    int type,
    int protocol,
    int socket_vector[2] );
```

**Parameters**

- **domain**: Specifies the communications domain in which the sockets are created. This function does not create sockets in the Internet domain.
- **type**: Specifies the communications method that sockets use, for example SOCK_DGRAM or SOCK_STREAM.
- **protocol**: Specifies an optional identifier used to define the communications protocols used in the transport layer interface.
- **socket_vector**: Specifies a two-integer array used to hold the file descriptors of the socket pair created with the call to this function.

**Description**

The `socketpair()` function creates an unnamed pair of connected sockets in a specified `domain`, of a specified `type`, under the protocol optionally specified by the `protocol` parameter. The two sockets are identical. The file descriptors used in referencing the created sockets are returned to `socket_vector[0]` and `socket_vector[1]`. The `sys/socket.h` include file contains definitions for socket domains, types, and protocols.

Not all protocol families support the `socketpair()` function.
Return Values

Upon successful completion, this function returns a value of 0 (zero). Otherwise, -1 is returned and_errno is specified to indicate the error.

Errors

If the socketpair( ) function fails, _errno may be set to one of the following values:

[EMFILE] The current process has too many open file descriptors.

[EAFNOSUPPORT] The addresses in the specified address family cannot be used to create this socket pair.

[EPROTONOSUPPORT] The specified protocol cannot be used in this system.

[EOPNOTSUPP] The specified protocol does not permit creation of socket pairs.

[EFAULT] The socket_vector array is not located in a writable part of user address space.

Related Information

Functions: socket(2)
sqrt, cbrt

**Purpose**
Computes square root and cube root functions

**Library**
Math Library (libm.a)

**Synopsis**
```
#include <math.h>

double sqrt (double x);

double cbrt (double x);
```

**Parameters**

- **x**
  Specifies some double value.

**Description**

The sqrt() and cbrt() functions compute the square root and cube root, respectively, of their parameters.

**Notes**

AES Support Level: Full use (sqrt())

**Return Values**

The sqrt() function returns the square root of x. The value of x must be positive. If x is NaN, NaN is returned. Otherwise, NaN is returned and errno is set to indicate the error.

The cbrt() function returns the cube root of x.
Errors

If the sqrt() function fails, errno may be set to the following value:

[EDOM] The value of the x parameter is negative.

Related Information

Functions: exp(3), isnan(3)

1-751
stat(2)

stat, fstat, lstat

**Purpose**
Provides information about a file

**Synopsis**
```c
#include <sys/stat.h>
#include <sys/types.h>

int stat(
    const char *path,
    struct stat *buffer);

int lstat(
    const char *path,
    struct stat *buffer);

int fstat(
    int filedes,
    struct stat *buffer);
```

**Parameters**
- `path` Specifies the pathname identifying the file.
- `filedes` Specifies the file descriptor identifying the open file.
- `buffer` Points to the `stat` structure in which information is returned. The `stat` structure is described in the `sys/stat.h` header file.

**Description**
The `stat()` function obtains information about the file named by the `path` parameter. Read, write, or execute permission for the named file is not required, but all directories listed in the pathname leading to the file must be searchable. The file information is written to the area specified by the `buffer` parameter, which is a pointer to a `stat` structure, defined in `sys/stat.h`.

The `fstat()` function is like the `stat()` function except that the information obtained is about an open file referenced by the `filedes` parameter.

The `lstat()` function is like the `stat()` function except in the case where the named file is a symbolic link. In this case, the `lstat()` function returns information about the link, while the `stat()` and `fstat()` functions return information about the file the link references. In the case of a symbolic link, the `stat()` functions set the `st_size` field of the `stat` structure to the length of the symbolic link, and sets the `st_mode` field to indicate the file type.
The `stat()`, `lstat()` , and `fstat()` functions update any time-related fields associated with the file before writing into the `stat` structure.

**Notes**

**AES Support Level:** Full use (`stat()`, `fstat()`)
Trial use (`lstat()`)  

**Return Values**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

**Errors**

If the `stat()` or `lstat()` function fails, `errno` may be set to one of the following values:

- `[ENOENT]` The file named by the `path` parameter does not exist or is an empty string.
- `[ELOOP]` Too many links were encountered in translating `path`.
- `[ENAMETOOLONG]` The length of the `path` parameter exceeds `PATH_MAX` or a pathname component is longer than `NAME_MAX`.
- `[EACCES]` Search permission is denied for a component of the `path` parameter.
- `[ENOTDIR]` A component of the `path` parameter is not a directory.
- `[EFAULT]` Either the `buffer` parameter or the `path` parameter points to a location outside of the allocated address space of the process.

If the `fstat()` function fails, `errno` may be set to one of the following values:

- `[EBADF]` The `filedes` parameter is not a valid file descriptor.
- `[EFAULT]` The `buffer` parameter points to a location outside of the allocated address space of the process.

**Related Information**

Functions: `chmod(2)`, `chown(2)`, `link(2)`, `mknod(2)`, `mount(3)`, `open(2)`, `pipe(2)`, `symlink(2)`, `utime(2)`
statfs, fstatfs, ustat

**Purpose**
Gets file system statistics

**Synopsis**
```
#include <sys/statfs.h>

int statfs(
    char *path,
    struct statfs *buffer,
    int length);

int fstatfs(
    int file_descriptor,
    struct statfs *buffer,
    int length);

#include <sys/types.h>
#include <ustat.h>

int ustat(
    dev_t device,
    struct ustat *buffer);
```

**Parameters**
- **path**
  Specifies any file within the mounted file system.
- **file_descriptor**
  Specifies a file descriptor obtained by a successful `open()` or `fcntl()` function.
- **buffer**
  Points to a `statfs` buffer to hold the returned information for the `statfs()` or `fstatfs()` function; points to a `ustat` buffer to hold the returned information for the `ustat()` function.
- **length**
  Specifies the size of the buffer pointed to by the `buffer` parameter.
- **device**
  Specifies the ID of the device. It corresponds to the `st_rdev` member of the structure returned by the `stat()` function.

**Description**
The `statfs()` and `fstatfs()` functions return information about a mounted file system. The returned information is in the format of a `statfs` structure, defined in the `sys/statfs.h` header file.
The `ustat()` function also returns information about a mounted file system. The returned information is in the format of a `ustat` structure, defined in the `ustat.h` header file. This function is superseded by the `statfs()` and `fstatfs()` functions.

**Return Values**

Upon successful completion, 0 (zero) is returned. Otherwise, -1 is returned, and `errno` is set to indicate the error.

**Errors**

If the `statfs()` function fails, `errno` may be set to one of the following values:

- `[EFAULT]` The `buffer` or `path` parameter points to a location outside of the allocated address space of the process.
- `[ENOTDIR]` A component of the path prefix of the `path` parameter is not a directory.
- `[EINVAL]` The `path` parameter contains a character with the high-order bit set.
- `[ENAMETOOLONG]` The length of a component of the `path` parameter exceeds `NAME_MAX` characters, or the length of the `path` parameter exceeds `PATH_MAX` characters.
- `[ENOENT]` The file referred to by the `path` parameter does not exist.
- `[EACCES]` Search permission is denied for a component of the path prefix of the `path` parameter.
- `[ELOOP]` Too many symbolic links were encountered in translating the `path` parameter.
- `[EIO]` An I/O error occurred while reading from or writing to the file system.

If the `fstatfs()` or `ustat()` function fails, `errno` may be set to one of the following values:

- `[EBADF]` The `file_descriptor` parameter is not a valid file descriptor.
- `[EIO]` An I/O error occurred while reading from the file system.
- `[EFAULT]` The `buffer` parameter points to an invalid address.

**Related Information**

Functions: `stat(2)`
stime

Purpose
Sets the system-wide time-of-day clock

Library
Standard C Library (libc.a)

Synopsis
#include <sys/time.h>
int stime(
    long *tz);

Parameters
    tz Points to the value of time, to be interpreted as the number of
         seconds since 00:00:00 GMT on January 1, 1970.

Description
The stime( ) function sets the time and date of the system.

Return Values
Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is
returned and errno is set to indicate the error.

Errors
If the stime( ) function fails, errno is set to one of the following values:
[EPERM] The calling process does not have the appropriate system privilege.

Related Information
Functions: gettimeofday(2), gettimer(3)
**strftime**

**Purpose**
Converts date and time to string

**Library**
Standard C Library (*lib*.a)

**Synopsis**
```c
#include <time.h>
size_t strftime(
    char *s,
    size_t maxsize,
    const char *format,
    const struct tm *timeptr);
```

**Parameters**

- **s**
  Points to the array containing the specified date and time string.

- **maxsize**
  Specifies the maximum number of characters to be written to the array pointed to by the `s` parameter.

- **format**
  Points to a sequence of control characters (refer to the foregoing list) that specify the format of the date and time string pointed to by the `s` parameter.

- **timeptr**
  Points to a type `tm` structure that contains broken-down time information.

**Description**

The `strftime()` function places characters into the array pointed to by the `s` parameter as controlled by the string pointed to by the `format` parameter. The string pointed to by the `format` parameter is a multibyte character sequence, beginning and ending in its initial shift state.

Local time zone information is used as though the `strftime()` function called the `tzset()` function. Time information used in this subroutine is fetched from space containing type `tm` structure data, which is defined in the `time.h` include file. The type `tm` structure must contain the time information used by this subroutine to construct the time and date string.
The *format* string consists of zero or more conversion specifications and ordinary multibyte characters. A conversion specification consists of a % (percent) character followed by a character that determines how the conversion specification constructs the formatted string.

All ordinary multibyte characters (including the terminating null character) are copied unchanged into the `s` array. When copying between objects that overlap takes place, behavior of this function is undefined. No more than the number of characters specified by the `maxsize` parameter are written to the array. Each conversion specification is replaced by appropriate characters as described in the following list. The appropriate characters are determined by the LC_TIME category of the current locale and by values specified by the type `tm` structure pointed to by the `timeptr` parameter.

- `%a` Is replaced by the abbreviated weekday name appropriate for the locale
- `%A` Is replaced by the full weekday name appropriate for the locale
- `%b` Is replaced by the abbreviated month name appropriate for the locale
- `%B` Is replaced by the full month name appropriate for the locale
- `%c` Is replaced by the date and time representation appropriate for the locale
- `%d` Is replaced by the day of the month as a decimal number [01, 31]
- `%D` Is replaced by the date (%m/%d/%y)
- `%h` Is replaced by the abbreviated month name appropriate for the locale
- `%H` Is replaced by the hour (24-hour clock) as a decimal number [00, 23]
- `%i` Is replaced by the hour (12-hour clock) as a decimal number [01, 12]
- `%j` Is replaced by the day of the year as a decimal number [001, 366]
- `%m` Is replaced by the month as a decimal number [01, 12]
- `%M` Is replaced by the minute as a decimal number [00, 59]
- `%n` Is replaced by a newline character
- `%p` Is replaced by the locale equivalent of either a.m. or p.m.
- `%r` Is replaced by the time in a.m./p.m. notation according to British/US conventions (%I:%M:%S [AM|PM])
- `%S` Is replaced by the second as a decimal number [00, 61]
- `%t` Is replaced by a tab character
- `%T` Is replaced by the time (%H:%M:%S)
- `%U` Is replaced by the week number of the year (Sunday as the first day of the week) as a decimal number [00, 53]
%w  Is replaced by the weekday as a decimal number [0(Sunday), 6]
%H  Is replaced by the week number of the year (Monday as the first day of the
    week) as a decimal number [00, 53]
%x  Is replaced by the date representation appropriate for the locale
%X  Is replaced by the time representation appropriate for the locale
%y  Is replaced by the year without century as a decimal number [00, 99]
%Y  Is replaced by the year with century as a decimal number
%Z  Is replaced by the time zone name or abbreviation, or by no characters
    when no time zone exists
%%  Is replaced by %
When a directive is not one of the above, the behavior of this function is undefined.

Notes

AES Support Level:  Full use

Return Values

When the total number of resulting characters, including the terminating null
character, is not more than maxsize, the strftime() function returns the number of
characters written into the array pointed to by the s parameter. The returned value
does not include the terminating null character. Otherwise, a value of (size_t) 0
(zero) is returned and the contents of the array are undefined.

Related Information

Functions:  ctime(3), setlocale(3)
string(3)

strcat, strchr, strcmp, strcoll, strcpy, strcspsn, strdup, strerror, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr, strtok, strtok_r, strxfrm

**Purpose**

Performs operations on strings

**Library**

Standard C Library (libc.a)

**Synopsis**

```
#include <string.h>
char *strcat(
    char *s1,
    const char *s2);
char *strchr(
    const char *s,
    int c);
int strcmp(
    const char *s1,
    const char *s2);
int strcoll(
    const char *s1,
    const char *s2);
char *strcpy(
    char *s1,
    const char *s2);
size_t strcspsn(
    const char *s1,
    const char *s2);
char *strdup(
    char *s1);
char *strerror(
    int errnum);
size_t strlen(
    char *s);
```
char *strncat(   
    char *sl,   
    const char *s2,   
    size_t n) ;

int strncmp(   
    const char *sl,   
    const char *s2,   
    size_t n

char *strncpy(   
    char *sl,   
    const char *s2,   
    size_t n);

char *strpbrk(   
    const char *sl,   
    const char *s2);

char *strstr(   
    const char *sl,   
    const char *s2)

size_t strspn(   
    const char *sl,   
    const char *s2);

size_t strxfrm(   
    char *sl,   
    const char *s2,   
    size_t n);
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$</td>
<td>Specifies a character expressed as an int data type in functions <code>strchr()</code> and <code>strrchr()</code>.</td>
</tr>
<tr>
<td><code>errno</code></td>
<td>Specifies an error-number value in the <code>strerror()</code> function.</td>
</tr>
<tr>
<td>$n$</td>
<td>Specifies the number of characters in a string referenced in the <code>strncat()</code>, <code>strncmp()</code>, <code>strcpy()</code>, and <code>strncpy()</code> functions.</td>
</tr>
<tr>
<td>$s$</td>
<td>Specifies a string variable referenced in the <code>strchr()</code>, <code>strlen()</code>, and <code>strrchr()</code> functions.</td>
</tr>
<tr>
<td>$sl$</td>
<td>Points to a location containing one of two strings referenced in the <code>strcat()</code>, <code>strcpy()</code>, <code>strcoll()</code>, <code>strncpy()</code>, <code>strncat()</code>, <code>strncpy()</code>, <code>strpbrk()</code>, <code>strspn()</code>, <code>strstr()</code>, <code>strtok()</code>, and <code>strxfrm()</code> functions.</td>
</tr>
<tr>
<td>$s2$</td>
<td>Points to a location containing the second of two strings referenced in the same functions that use the $sl$ parameter.</td>
</tr>
<tr>
<td><code>last_string</code></td>
<td>Points to the first character of the next token.</td>
</tr>
</tbody>
</table>

Description

The **string** functions copy, compare, and append strings in memory, and determine such values as location, size, and the existence of strings in memory.

The `strcat()` function appends a copy of the string pointed to by the $s2$ parameter, including the terminating null character, to the end of the string pointed to by the $sl$ parameter. The beginning character of the string pointed to by the $s2$ parameter overwrites the null character at the end of the string pointed to by the $sl$ parameter. When operating on overlapping strings, the behavior of this function is unreliable.

The `strchr()` function locates the first occurrence of the integer specified by the $c$ parameter, which is converted to a char, in the string pointed to by the $s$ parameter. The terminating null character is treated as part of the string pointed to by the $s$ parameter.

The `strcmp()` function compares the string pointed to by the $sl$ parameter to the string pointed to by the $s2$ parameter. The sign of a nonzero value returned by `strcmp()` is determined by the sign of the difference between the values of the first pair of bytes (both interpreted as unsigned char) that differ in the two compared objects.

The `strcoll()` function compares the string pointed to by the $sl$ parameter with the string pointed to by the $s2$ parameter, both interpreted as appropriate to the LC_COLLATE category of the current locale. The sign of a nonzero value
Functions

string(3)

returned by `strcoll()` is determined by the relative ordering within the current
collating sequence of the first pair of characters that differ in the objects under
comparison.

The `strcpy()` function copies the string pointed to by the `s2` parameter, including
the terminating null character, to the location pointed to by the `sl` parameter. When operating on overlapping strings, the behavior of this function is unreliable.

The `strdup()` function returns a pointer to a new string that is an exact duplicate of
the string pointed to by the `sl` parameter. The `malloc()` function is used to allocate
space for the new string.

The `strerror()` function maps the error number specified by the `errnum` parameter
to a language-dependent error message string, and returns a pointer to the string. The string pointed to by the return value is not modified by the program, but may be overwritten by a subsequent call to this function. The implementation behaves as though no other function calls the `strerror()` function.

The `strlen()` function returns the number of bytes in the string pointed to by the `s` parameter. The string length value does not include the string terminating null character.

The `strcmp()` function compares the string pointed to by the `sl` parameter with the
string pointed to by the `s2` parameter. The sign of any nonzero value returned by `strcmp()` is determined by the sign resulting from the difference in integer values of the first character-pair comparison (both converted to `unsigned char`) in which the characters are different.

The `strncat()` function appends `n` bytes in the string pointed to by the `s2` parameter
to the end of the string pointed to by the `sl` parameter. The initial character of the string pointed to by `s2` overwrites the null character at the end of the string pointed to by `sl`. The number of characters specified by the `n` parameter and a terminating null character are always appended to the string pointed to by the `sl` parameter. When operating on overlapping strings, the behavior of this function is unreliable.

The `strncpy()` function copies no more than the number of characters specified by
the `n` parameter from the location pointed to by the `s2` parameter to the location pointed to by the `sl` parameter. Characters following a null character are not copied. When operating on overlapping strings, the behavior of this function is unreliable. When the location pointed to by the `s2` parameter is a string whose character length is less than the value specified by the `n` parameter, null characters are appended to the `sl` string until `n` characters are contained in the string.
The `strpbrk()` function scans the string pointed to by the `sl` parameter for the first occurrence of any character in the string pointed to by the `s2` parameter.

The `strrchr()` function locates the last occurrence of the integer specified by the `c` parameter, which is converted to a `char` value, in the string pointed to by the `s` parameter. The terminating null character is treated as a part of the string pointed to by the `s` parameter.

The `strspn()` function computes the length of the maximum initial segment of the string pointed to by the `sl` parameter, which consists entirely of characters from the string pointed to by the `s2` parameter.

The `strcspn()` function computes the byte length of the maximum initial segment of the string pointed to by the `sl` parameter, which consists entirely of characters that are not from the string pointed to by the `s2` parameter.

The `strstr()` function locates the first occurrence in the string pointed to by the `sl` parameter of the sequence of bytes in the string pointed to by the `s2` parameter, excluding the terminating null character.

The `strtok()` function expects that the string pointed to by the `sl` parameter consists of multiple tokens separated by one or more characters that match those in a separator string pointed to by the `s2` parameter. A sequence of calls to the `strtok()` function breaks the `sl` string into a sequence of expected tokens, each of which is delimited by one or more characters from the `s2` string.

The initial call to function `strtok()` in the token-sequence search specifies the `sl` parameter as the address of the token string. This call is followed by subsequent calls that have a null pointer as the value of the `sl` parameter. The separator string pointed to by the `s2` parameter may be different in every call to this function.

The first call in the token-sequence search tests every character in the `sl` string for any character that is not contained in the current separator string pointed to by the `s2` parameter. When no matching character is found, there are no tokens in that string and a null pointer is returned. When a nonseparator character is found, it becomes the starting character of the next token.

The `strtok()` function then searches for a character that matches any character in the current separator string pointed to by the `s2` parameter. When no matching character is found, the current token extends to the end of the string pointed to by the `sl` parameter and subsequent searches for a token return a null pointer. When a matching separator-string character is found, it is overwritten by the null character, which terminates the current token. The `strtok()` function saves a pointer to the next character, which is the character from which the next search for a token starts.
Each subsequent call having a null pointer as the value of the the $sl$ parameter begins a search at the character pointed to by the saved pointer and behaves as described above.

The implementation behaves as though no function calls the `strtok()` function.

The `strtok_r()` function is the reentrant version of `strtok()`. Upon successful completion, the first character of the next token is stored in `**last_string`, and a value of 0 (zero) is returned.

The `strxfrm()` function transforms the string pointed to by the $sl$ parameter and places the result in the address specified by $s2$. When the `strcmp()` function is applied to two transformed strings, a value greater than, equal to, or less than 0 (zero) is returned. The returned value corresponds to the same value that is returned when the `strcoll()` function is applied to the same two original transformed strings. No more than $n$ characters are placed in the location pointed to by the $sl$ parameter, including the terminating null character. When $n$ is 0 (zero), the $sl$ parameter is a null pointer. When operating on overlapping strings, the behavior of this function is unreliable.

**Notes**

AES Support Level: Full use (strcat(), strchr(), strcmp(), strcoll(), strcpy(), strcspn(), strlen(), strcat(), strncpy(), strpbrk(), strrchr(), strspn(), strstr(), strtok(), strxfrm())

**Return Values**

Upon successful completion, the `strcat()`, `strcpy()`, `strncat()`, and `strncpy()` functions return a pointer to the resulting string. Otherwise these functions return a null pointer.

Upon successful completion, the `strchr()` and `strrchr()` functions return a pointer to the matching character in the scanned string, When the character specified by parameter $c$ is not found, a null pointer is returned.

Upon successful completion, the `strcmp()`, `strcoll()`, and `strncmp()` functions return an integer whose value is greater than, equal to, or less than 0 (zero), according to whether the $sl$ string is greater than, equal to, or less than the $s2$ string. When a successful comparison can not be made, these functions return 0 (zero).
Upon successful completion, the `strcspn()`, `strspn()`, and `strxfrm()` functions return the length of the string segment. Otherwise, `(size_t)-1` is returned and `errno` is set to indicate the error.

Upon successful completion, the `strerror()` function returns a pointer to the generated message string. If the error number is not valid, `errno` is set to `[EINVAL]`.

Upon successful completion, the `strlen()` function returns the number of characters in the string to which the `s` parameter points. Otherwise, `(size_t)-1` is returned and `errno` is set to indicate the error.

Upon successful completion, the `strpbrk()` function returns a pointer to the matched character. When no character in the string pointed to by the `s2` parameter occurs in the string pointed to by the `sl` parameter, a null pointer is returned and the value of `errno` remains unchanged. On error, the `strpbrk()` function returns a null pointer and sets `errno` to indicate the error.

Upon successful completion, the `strstr()` function returns a pointer to the located string or a null pointer when the string is not found. When the `s2` parameter points to a string having 0 (zero) length, the `strstr()` function returns the string pointed to by parameter `sl`. On error, a null pointer is returned and `errno` is set to indicate the error.

Upon successful completion, the `strtok()` function returns a pointer to the first character of the parsed token in the string. When there is no token in the string, a null pointer is returned.

Errors

If one of the `string` functions fails, `errno` may be set to the following value:

`[EINVAL]` The string pointed to by the `sl` or `s2` parameter contains characters outside the domain of the collating sequence, or the value of the `errnum` parameter used in the `strerror()` function is not a valid message number.

Related Information

Functions: `memccpy(3)`, `setlocale(3)`, `swab(3)`
swab

Purpose
Swaps bytes

Library
Standard C Library (libc.a)

Synopsis
#include <string.h>

void swab(
    const char *src,
    char *dest,
    int nbytes
    );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Points to the location of the string to copy.</td>
</tr>
<tr>
<td>dest</td>
<td>Points to the location to which the resulting string is copied.</td>
</tr>
<tr>
<td>nbytes</td>
<td>Specifies the number of even nonnegative bytes to be copied. The nbytes parameter should have an even nonnegative value. When the nbytes parameter is odd positive, the swab() function uses nbytes-1 instead. When the nbytes parameter is negative, the swab() function does nothing.</td>
</tr>
</tbody>
</table>

Description
The swab() function copies the number of bytes specified by the nbytes parameter from the location pointed to by the src parameter to the array pointed to by the dest parameter, exchanging adjacent even and odd bytes.

Notes

AES Support Level: Trial use

Return Values
The swab() function returns no values.

Related Information
Functions: memccpy(3), string(3)
swapon

Purpose  Adds a swap device for interleaved paging and swapping

Synopsis  swapon(
            char *path,
            int flags,
            int lowat,
            int hiwat);

Parameters

path    Specifies the file or block device to be made available.
flags   Specifies a flag. Only one flag is currently supported:
        MS_PREFER
        The specified path becomes the preferred paging file or
device.
lowat   Specifies the low water mark.
hiwat   Specifies the high water mark.

Description

The swapon() function makes a file or block special device available to the system
for allocation of paging and swapping space.

The lowat and hiwat parameters specify the low water and high water marks that
the paging file will float between. If the low water mark is 0 (zero), then the file
will not shrink after paging space is freed. If the high water mark is 0 (zero), then
the file will grow without bounds. These parameters are not used for swapping
devices. The size of the swap area on the block device is calculated at the time the
device is first made available for swapping.

The calling process must have superuser privilege to call the swapon() function.

Return Values

Upon successful completion, the swapon() function returns a value of 0 (zero). If
an error has occurred, -1 is returned and errno is set to indicate the error.
Errors

If the `swapon()` function fails, `errno` may be set to one of the following values:

- **[ENOTDIR]** A component of the path prefix is not a directory.
- **[EINVAL]** The pathname contains a character with the high-order bit set, the device was not specified, the device configured by the `path` parameter was not configured into the system as a swap device, or the device does not allow paging.
- **[ENAMETOOLONG]** A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters.
- **[ENOENT]** The named file or device does not exist.
- **[EACCES]** Search permission is denied for a component of the path prefix.
- **[ELOOP]** Too many symbolic links were encountered in translating the pathname.
- **[EPERM]** The caller does not have appropriate privilege.
- **[EBUSY]** The file or device specified by the `path` parameter has already been made available for swapping.
- **[ENXIO]** The major device number of the `path` parameter is out of range (this indicates no device driver exists for the associated hardware).
- **[EIO]** An I/O error occurred while opening the swap device.
- **[EFAULT]** The `path` parameter points outside the process’ allocated address space.
- **[EROFS]** An attempt was made to activate a paging file on a read-only file system.

Related Information

Commands: `swapon(8), config(8)`
symlink

Purpose
Makes a symbolic link to a file

Synopsis
#include <symlink.h>

int symlink ( 
    const char *path1,
    const char *path2 );

Parameters
path1 Specifies the contents of the symbolic link to create.
path2 Names the symbolic link to be created.

Description
The symlink() function creates a symbolic link with the name specified by the path2 parameter which refers to the file named by the path1 parameter.

Like a hard link (described in the link() function), a symbolic link allows a file to have multiple names. The presence of a hard link guarantees the existence of a file, even after the original name has been removed. A symbolic link provides no such assurance; in fact, the file named by the path1 parameter need not exist when the link is created. Unlike hard links, a symbolic link can cross file system boundaries.

When a component of a pathname refers to a symbolic link rather than a directory, the pathname contained in the symbolic link is resolved. If the pathname in the symbolic link starts with a / (slash), the symbolic link pathname is resolved relative to the process root directory. If the pathname in the symbolic link does not start with a / (slash), the symbolic link pathname is resolved relative to the directory that contains the symbolic link.

If the symbolic link is the last component of the original pathname, remaining components of the original pathname are appended to the contents of the link and pathname resolution continues.

The symbolic link pathname may or may not be traversed, depending on which function is being performed. Most functions traverse the link.
The functions which refer only to the symbolic link itself, rather than to the object to which the link refers, are:

**link()**  
An error will be returned if a symbolic link is named by the `path2` parameter.

**lstat()**  
If the file specified is a symbolic link, the status of the link itself is returned.

**mknod()**  
An error will be returned if a symbolic link is named as the `path` parameter.

**readlink()**  
This call applies only to symbolic links.

**remove()**  
A symbolic link can be removed by invoking the `remove()` function.

**rename()**  
If the file to be renamed is a symbolic link, the symbolic link is renamed. If the new name refers to an existing symbolic link, the symbolic link is destroyed.

**rmdir()**  
An error will be returned if a symbolic link is named as the `path` parameter.

**symlink()**  
An error will be returned if the symbolic link named by the `path2` parameter already exists. A symbolic link can be created that refers to another symbolic link; that is, the `path1` parameter can refer to a symbolic link.

**unlink()**  
A symbolic link can be removed by invoking `unlink()`.

Search access to the symbolic link is required to traverse the pathname contained therein. Normal permission checks are made on each component of the symbolic link pathname during its resolution.

**Notes**

**AES Support Level:** Trial use

**Return Values**

Upon successful completion, the `symlink()` function returns a value of 0 (zero). If the `symlink()` function fails, a value of -1 is returned and `errno` is set to indicate the error.
Errors

If the symlink() function fails, errno may be set to one of the following values:

[EEXIST] The path specified by the path2 parameter already exists.

[EACCES] The requested operation requires writing in a directory with a mode that denies write permission, or search permission is denied on a component of path2.

[EROFS] The requested operation requires writing in a directory on a read-only file system.

[ENOSPC] The directory in which the entry for the symbolic link is being placed cannot be extended because there is no space left on the file system containing the directory.

[EDQUOT] The directory in which the entry for the symbolic link is being placed cannot be extended because the user’s quota of disk blocks on the file system containing the directory has been exhausted.

[ENOENT] The path2 parameter points to a null pathname, or a component of path2 does not exist.

[ENOTDIR] A component of path2 is not a directory.

[ENAMETOOLONG] The length of the path1 parameter or path2 parameter exceeds PATH_MAX, or a pathname component of path2 is longer than NAME_MAX.

Related Information

Functions: link(2), readlink(2), unlink(2)

Commands: ln(1)
sync

Purpose Updates all file systems

Synopsis void sync ( void);

Description
The sync() function causes all information in memory that should be on disk to be written out. The writing, although scheduled, is not necessarily complete upon return from the sync() function. Types of information to be written include modified superblocks, inodes, data blocks, and indirect blocks.

The sync() function should be used by programs that examine a file system, such as the df command and the fsck command.

Related Information
Functions: fsync(2)
Commands: sync(1)
sysconf

Purpose  Gets configurable system variables

Library  Standard C Library (libc.a)

Synopsis  #include <unistd.h>

long sysconf (  
    int name );

Parameters  
  name  Specifies the system variable to be queried.

Description  
The sysconf() function provides a method for determining the current value of a configurable system limit or whether optional features are supported.

The set of system variables from the limits.h or unistd.h include file that are returned by the sysconf() function, and the symbolic constants, defined in the unistd.h header file that correspond to the name parameter, are as follows:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value of name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARG_MAX</td>
<td>_SC_ARG_MAX</td>
</tr>
<tr>
<td>CHILD_MAX</td>
<td>_SC_CHILD_MAX</td>
</tr>
<tr>
<td>clock ticks/second</td>
<td>_SC_CLK_TCK</td>
</tr>
<tr>
<td>NGROUPS_MAX</td>
<td>_SC_NGROUPS_MAX</td>
</tr>
<tr>
<td>OPEN_MAX</td>
<td>_SC_OPEN_MAX</td>
</tr>
<tr>
<td>_POSIX_JOB_CONTROL</td>
<td>_SC_JOB_CONTROL</td>
</tr>
<tr>
<td>_POSIX_SAVED_IDS</td>
<td>_SC_SAVED_IDS</td>
</tr>
<tr>
<td>_POSIX_VERSION</td>
<td>_SC_VERSION</td>
</tr>
<tr>
<td>PASS_MAX</td>
<td>_SC_PASS_MAX</td>
</tr>
<tr>
<td>_XOPEN_VERSION</td>
<td>_SC_XOPEN_VERSION</td>
</tr>
</tbody>
</table>
### Variable

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value of name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATEXIT_MAX</td>
<td>_SC_ATEXIT_MAX</td>
</tr>
<tr>
<td>PAGE_SIZE</td>
<td>_SC_PAGE_SIZE</td>
</tr>
<tr>
<td>AES_OS_VERSION</td>
<td>_SC_AES_OS_VERSION</td>
</tr>
</tbody>
</table>

### Notes

AES Support Level: Full use

### Return Values

If the `name` parameter is an invalid value, the `sysconf()` function returns -1 and sets `errno` to indicate the error. If the variable corresponding to `name` is undefined, the `sysconf()` function returns -1 without changing the value of `errno`.

If the `name` parameter is `_SC_JOB_CONTROL` or `_SC_SAVED_IDS`, the `sysconf()` function returns a nonnegative value.

Otherwise, the `sysconf()` function returns the current variable value on the system. The value will not change during the lifetime of the calling process.

### Errors

If the `sysconf()` function fails, `errno` may be set to the following value:

- [EINVAL]: The value of the `name` parameter is invalid.

### Related Information

Functions: `pathconf(3)`
syslog, openlog, closelog, setlogmask

**Purpose**  Controls the system log

**Library**  Standard C Library (*libc.a*)

**Synopsis**  
```c
#include <syslog.h>

int openlog (  
    char *id,  
    int log_option,  
    int facility );  

int syslog (  
    int priority,  
    char *message [, value... ] );  

int closelog ( void );  
int setlogmask(  
    int mask_priority );
```

**Parameters**

- **id**  Specifies a string that is attached to the beginning of every message.
- **log_option**  Specifies logging options. Values of the *log_option* parameter include:
  - Log the process ID with each message. This option is useful for identifying daemons.
  - LOG_CONS  Send messages to the console if unable to send them to *syslogd*. This option is useful in daemon processes that have no controlling terminal.
  - LOG_NDELAY  Open the connection to *syslogd* immediately, instead of when the first message is logged. This option is useful for programs that need to manage the order in which file descriptors are allocated.
Functions

syslog(3)

LOG_NOWAIT
Log messages to the console without waiting for child processes that are forked. Use this option for processes that enable notification of termination of child processes through SIGCHLD; otherwise, the syslog() function may block, waiting for a child process whose exit status has already been collected.

facility
Specifications the facility that generated the message, which is one of the following:

LOG_KERN
Messages generated by the kernel. These cannot be generated by any user processes.

LOG_USER
Messages generated by user processes. This is the default facility when none is specified.

LOG_MAIL
Messages generated by the mail system.

LOG_DAEMON
Messages generated by system daemons.

LOG_AUTH
Messages generated by the authorization system: login, su, and so on.

LOG_LPR
Messages generated by the line printer spooling system.

LOG_RFS
Messages generated by remote file systems.

LOG_LOCAL0 through LOG_LOCAL7
Reserved for local use.

priority
Messages are tagged with codes indicating the type of priority for each. The priority parameter is encoded as a facility (as listed above), which describes the part of the system generating the message, and as a level, which indicates the severity of the message. The level of severity is selected from the following list:

A panic condition reported to all users.

LOG_ALERT
A condition to be corrected immediately; for example, a corrupted database.
Description

The `syslog()` function writes messages to the system log maintained by the `syslogd` daemon.

The `message` parameter is similar to the `printf(fmt)` string, with the difference that `%m` is replaced by the current error message obtained from `errno`. A trailing new line can be added to the message if needed. The `value` parameters are the same as the `value` parameters of the `printf()` function.

The `syslogd` daemon reads messages and writes them to the system console or to a log file, or forwards them to the `syslogd` daemon on the appropriate host.

If `syslog()` cannot pass the message to `syslogd`, it writes the message on `/dev/console`, provided the LOG_CONS option is set.

If special processing is required, the `openlog()` function can be used to initialize the log file. The `id` parameter contains a string that is attached to the beginning of every message. The `facility` parameter encodes a default facility from the previous list to be assigned to messages that do not have an explicit facility encoded.

The `closelog()` function closes the log file.
The `setlogmask()` function uses the bit mask in the `mask_priority` parameter to set the new log priority mask and returns the previous mask. Logging is enabled for the levels indicated by the bits in the mask that are set and is disabled where the bits are not set. The default mask allows all priorities to be logged. If the `syslog()` function is called with a priority mask that does not allow logging of that level of message, then the function returns without logging the message.

**Return Values**

The `syslog()` function returns -1 if either the priority mask excludes this message from being logged, or if an error occurs and it is impossible to send the message to the `syslogd` daemon or to the system console.

**Related Information**

Functions: `profil(2)`

Commands: `cc(1)`
system

Purpose    Executes a shell command

Library    Standard C Library (libc.a)

Synopsis    

```
#include <stdio.h>
#include <stdlib.h>

int system (const char *string);
```

Parameters

```
string      Specifies a valid sh shell command.
```

Description

The `system()` function passes the `string` parameter to the `sh` command, which interprets `string` as a command and executes it.

The `system()` function invokes the `fork()` function to create a child process that in turn uses the `exec` function to run `sh`, which interprets the shell command contained in the `string` parameter. The current process waits until the shell has completed, then returns the exit status of the shell.

Notes

AES Support Level: Full use

Return Values

Upon successful completion, the `system()` function returns the exit status of the shell. Otherwise, the `system()` function returns a value of -1 and sets `errno` to indicate the error. Exit status 127 indicates that the shell could not be executed.
Errors

If the `system()` function fails, `errno` may be set to one of the following values:

- **[EAGAIN]** The system-imposed limit on the total number of processes under execution, system-wide or by a single user ID, would be exceeded.
- **[EINTR]** The `system()` function was interrupted by a signal which was caught.
- **[ENOMEM]** There is not enough space left for this process.

Related Information

Functions: `exec(2)`, `exit(2)`, `fork(2)`, `wait(2)`

Commands: `sh(1)`
t_accept

Purpose  Accepts a connect request

Library  XTI Library (libtli.a)

Synopsis  
```c
#include <xti.h>

int t_accept(
    int fd,
    int resfd,
    struct t_call *call);
```

Parameters

The t_accept() function can only be called in the T_INCON transport provider state. The following table summarizes the relevance of input and output parameters before and after t_accept( ) is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>resfd</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;addr.maxlen</code></td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;addr.len</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;addr.buf</code></td>
<td>o(o)</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;opt.maxlen</code></td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;opt.len</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;opt.buf</code></td>
<td>o(o)</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;udata.maxlen</code></td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;udata.len</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;udata.buf</code></td>
<td>o(o)</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;sequence</code></td>
<td>y</td>
<td>n</td>
</tr>
</tbody>
</table>

Notes to table:

- **y**: This is a meaningful parameter.
- **n**: This is not a meaningful parameter.
- **o**: This is an optional parameter.
- **(o)**: The content of the object pointed to by `o` is optional.
Functions

t_accept(3)

fd
Specifies a file descriptor returned by the t_open() function that identifies the local transport endpoint from which the connect indication arrived.

resfd
Specifies the local transport endpoint where the connection is to be established. A calling transport user may accept a connection on either the same, or on a different local transport endpoint than the one on which the connect indication arrived.

Before the connection can be accepted on the same transport endpoint (resfd == fd), the calling transport user must have responded to any previous connect indications received on that same transport endpoint using the t_accept() or t_snddis() functions. Otherwise, the t_accept() function fails and sets t_errno to [TBADF].

When a different transport endpoint (resfd != fd) is specified, the transport endpoint must be bound to a protocol address with a call to the t_bind() function. When the address bound to the resfd parameter is the same as that bound to the fd parameter, the req->qlen parameter of t_bind() must be set to 0 (zero).

Also, the transport provider state must be T_IDLE, (refer to the t_getstate() function) before the t_accept() function is called. For both types of transport endpoint, t_accept() fails and sets t_errno to [TLOOK] when there are indications, such as connect or disconnect, waiting to be received at that endpoint.

call
Points to a type t_call structure used to store information required by the transport provider to complete the connection. The t_call structure has the following four members:

struct netbuf addr
Specifies a buffer for protocol address information sent by the calling transport user. The type netbuf structure referenced by this member is defined in the xti.h include file. This structure, which is used to define buffer parameters, has the following members:

unsigned int maxlen
Specifies the maximum byte length of the data buffer.

unsigned int len
Specifies the actual byte length of data written to the buffer.

char *buf
Points to the buffer location.
struct netbuf opt
    Specifies protocol-specific parameters associated with the
calling transport user.

struct netbuf udata
    Specifies parameters of user data returned to the calling
transport user from the remote transport user.

int sequence
    Specifies a unique identification number used to identify the
previously received connect indication.

The values of parameters specified by call->opt and the syntax of
those values are protocol-specific.

The call->udata parameters enable the remote transport user to
send data to the calling transport user. The amount of user data must
not exceed the limits specified by the transport provider as returned
in the info->connect parameter of the t_open() and t_getinfo() functions. When the call->udata.len parameter is 0 (zero), no data
is sent to the calling transport user.

Data specified by all call->udata.maxlen parameters are
meaningless.

The call->sequence parameter is a value returned by the t_listen()
function that uniquely associates the response with a previously
received connect indication.

Description

The t_accept() function is an XTI connection-oriented service function that is
issued by a calling transport user to accept a connect request after a connect
indication has arrived. Structures of types t_call and netbuf, which are defined in
the xti.h include file, are used by this function.

Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of
-1 is returned and t_errno is set to indicate the error.
Errors

If the `t_accept( )` function fails, `t_errno` may be set to one of the following:

[TBADF] The `fd` or `resfd` file descriptor does not refer to a transport endpoint, or the user is illegally accepting a connection on the same transport endpoint on which the connect indication arrived.

[TOUTSTATE] The `t_accept( )` function was called in the wrong sequence at the transport endpoint referenced by the `fd` parameter, or the transport endpoint referred to by the `resfd` parameter is not in the appropriate state.

[TACCES] The user does not have permission to accept a connection on the responding transport endpoint or to use the specified options.

[TBADOPT] The specified options were in an incorrect format or contained illegal information.

[TBADDATA] The amount of user data specified was not within the bounds allowed by the transport provider.

[TBADADDR] The specified protocol address was in an incorrect format or contained illegal information.

[TBADSEQ] An invalid sequence number was specified.

[TLOOK] An asynchronous event has occurred on the transport endpoint referenced by the `fd` parameter and requires immediate attention.

[TSYSERR] A system error occurred during execution of this function.

Related Information

Functions: `t_connect(3)`, `t_getstate(3)`, `t_listen(3)`, `t_open(3)`, `t_optmgmt(3)`, `t_rcvconnect(3)`
t_alloc

Purpose Allocates a library structure

Library XTI Library (libtli.a)

Synopsis #include <xti.h>
char *t_alloc(
    int fd,
    int struct_type,
    int fields);

Parameters
The t_alloc() function can be called in any transport provider state except T_UNINIT. (If called in T_UNIT, the function returns the TBADF error and an invalid fd). The following table summarizes the relevance of input and output parameters before and after t_alloc() is called:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>struct_type</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>fields</td>
<td>y</td>
<td>n</td>
</tr>
</tbody>
</table>

Notes to table:
y This is a meaningful parameter.
n This is not a meaningful parameter.

fd Specifies a file descriptor that identifies the local transport endpoint. Because the length of the allocated buffer is based on size information that is returned to the user on a call to the t_open() and t_getinfo() functions, the fd parameter must refer to the transport endpoint through which a newly allocated structure passes.

struct_type Specifies the structure type for the function for which memory is to be allocated; the struct_type parameter must specify one of the symbolic names listed in the Symbolic Name column of the following table.
The structures listed in the Structure Type column of the preceding table are referenced as a parameter in one or more of the various XTI transport service functions. Each structure type, except `struct t_info`, contains at least one member of structure type `struct netbuf`, which is defined in the `xti.h` include file. For each structure type in the preceding table, you may specify that the buffer for the `struct netbuf` member should be allocated as well. The length of the buffer allocated for the referenced structure member depends on protocol-specific size limits returned as `info` member information of the `t_open()` and `t_getinfo()` functions. Refer to the description of the `fields` parameter for the relevant sizes returned in `info`.

Specifies buffers for `t_info` type structures that are allocated for members of structures named by the `struct_type` parameter for a given function. The following table lists the symbolic name that must be specified for the `fields` parameter, identifies the `t_info` structure member that is the source of relevant size information, and lists the XTI function structure reference for which `t_info` Member memory space is reserved. The value of this parameter must be the bitwise logical OR of any of the symbolic names listed in the Symbol Name column.
### t_alloc(3)

<table>
<thead>
<tr>
<th>Symbol Name</th>
<th>t_info Member</th>
<th>Structure Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_ADDR</td>
<td>addr</td>
<td>Member addr of structures t_bind, t_call, t_unitdata, t_underr.</td>
</tr>
<tr>
<td>T_OPT</td>
<td>options</td>
<td>Member opt of structures t_optmgmt, t_call, t_unitdata, t_underr.</td>
</tr>
<tr>
<td>T_UDATA</td>
<td>tsdu</td>
<td>Member udata of structures t_call, t_discon, t_unitdata.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For struct_type T_CALL_STR, size is the greater value of members connect and discon of structure t_info.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For struct_type T_DIS_STR, size is the value of member discon of structure t_info.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For struct_type T_UNITDATA_STR, size is the value of member tsdu of structure t_info.</td>
</tr>
<tr>
<td>T_ALL</td>
<td>addr, options, tsdu</td>
<td>All relevant members of the specified structures.</td>
</tr>
</tbody>
</table>

For each field type specified by the fields parameter, the `t_alloc()` function reserves function memory for the associated buffer. Additionally, its len member is set to 0 (zero) and its buf pointer and maxlen members are initialized.

When the size value associated with any specified t_info structure member is -1 or -2 (see the `t_open()` or `t_getinfo()` functions), the `t_alloc()` function can not determine the size of the buffer, causing failure. On failure, t_errno is set to [TSYSERR] and errno is set to [EINVAL]. For any structure member not specified by this parameter, its buf member is set to the null pointer and its maxlen member is set to 0 (zero).
Description

The `t_alloc()` XTI memory utility function is used to dynamically allocate memory for structures required by various XTI transport interface functions. The structure to allocate is specified by a structure symbolic name used as a mnemonic. In most cases, the mnemonic is similar to the name of the corresponding function in which the structure is used.

The `t_alloc()` function allocates memory for the named structure as well as for other buffers referenced by the named structure. Use of this function to allocate structures ensures compatibility with the corresponding XTI transport interface functions in which the allocated structures are used.

Return Value

Upon successful completion, this function returns a pointer to the newly allocated structure. Upon failure, a null pointer is returned.

Errors

If the `t_alloc()` function fails, `t_errno` may be set to one of the following values:

- [TBADF] The `fd` file descriptor does not refer to a valid transport endpoint.
- [TSYSERR] A system error occurred during execution of this function.
- [TNOSTRUCTYPE] An unsupported structure type is specified.

Related Information

Functions: `t_free(3)`, `t_getinfo(3)`, `t_open(3)`
t_bind

**Purpose**  Binds an address to a transport endpoint

**Library**  XTI Library (libtli.a)

**Synopsis**  
```
#include <xti.h>

int t_bind(
    int fd,
    struct t_bind *req,
    struct t_bind *ret);
```

**Parameters**  
The `t_bind()` function can only be called in the `T_UNBND` transport provider state. The following table summarizes the relevance of input and output parameters before and after `t_bind()` is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>req-&gt;addr.maxlen</code></td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td><code>req-&gt;addr.len</code></td>
<td>y &gt;= 0</td>
<td>n</td>
</tr>
<tr>
<td><code>req-&gt;addr.buf</code></td>
<td>y(y)</td>
<td>n</td>
</tr>
<tr>
<td><code>req-&gt;qlen</code></td>
<td>y &gt;= 0</td>
<td>n</td>
</tr>
<tr>
<td><code>ret-&gt;addr.maxlen</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>ret-&gt;addr.len</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>ret-&gt;addr.buf</code></td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td><code>ret-&gt;qlen</code></td>
<td>n</td>
<td>y &gt;= 0</td>
</tr>
</tbody>
</table>

**Notes to table:**
- y This is a meaningful parameter.
- n This is not a meaningful parameter.
- (y) The content of the object pointed to by y is meaningful.

`fd` Specifies a file descriptor returned by the `t_open()` function that identifies the local transport endpoint. More than a single transport endpoint may be bound to the same protocol address, but only one protocol address can be bound to a transport endpoint.
When a transport user binds more than one transport endpoint to the same protocol address, only one endpoint can be used to listen for connect indications associated with that protocol address using the \texttt{t\_listen()} function. Consequently, for a given protocol address, only one \texttt{t\_bind()} function may specify a value greater than 0 (zero) for the \texttt{req->qlen} parameter. In this way, the transport provider can identify the transport endpoint that should be notified of an incoming connect indication is called.

No other transport endpoint may be bound for listening to that same protocol address when the initial listening endpoint is active, during data transfer, or during the T\_IDLE state. This prevents more than one transport endpoint, which is bound to the same protocol address, from accepting any connect indication.

\texttt{req}

Points to a type \texttt{t\_bind()} structure used to define the protocol address of the caller and to hold the allowable number of outstanding connect indications in connection-oriented transport protocol service. An outstanding connect indication is one that has been passed to the transport provider, but has not been accepted or rejected. The \texttt{t\_bind()} structure has the following two members:

\begin{verbatim}
struct netbuf addr
    Specifies a buffer for protocol address information sent by the calling transport user. The type netbuf structure referenced by this member is defined in the xti.h include file.

    This structure, which is used to specify the address to be bound to the endpoint, has the following members:

    unsigned int maxlen
        Specifies the maximum byte length of the data buffer.

    unsigned int len
        Specifies the actual byte length of data written to the buffer.

    char *buf
        Points to the buffer location.

    unsigned qlen
        Specifies the allowable number of outstanding connect indications in connection-oriented service.

The \texttt{req} parameter is used to request that the protocol address, pointed to by \texttt{req->addr.buf} be bound to the transport endpoint specified by the \texttt{fd} parameter. The \texttt{req->addr.maxlen} parameter has no meaning.
\end{verbatim}
When the protocol address is not available, or when 0 (zero) is specified for `req->addr.len`, the transport provider assigns an alternate protocol address whenever automatic address generation is supported. A pointer to the returned alternate protocol address is specified by `req->addr.buf`.

When a transport user does not specify a protocol address, the value 0 (zero) is used for `req->addr.len`. When the transport provider does not support automatic address generation and the value 0 (zero) is specified by `req->addr.len` as the data buffer length, a `t_bind()` call returns the value -1 and sets `t_errno` to `[TNOADDR]`.

A value greater than 0 (zero) for `req->qlen` has meaning whenever it is specified by a transport user expecting other transport users to call it. When the transport provider can not support the requested number of allowable outstanding connections, the value returned in `ret->qlen` may be different than the one requested.

The `req` parameter may be specified as a null pointer when a transport user does not need to use a protocol address for binding. The `req` parameter may also be specified as a null pointer when the protocol address is not significant.

When the protocol addresses pointed to by the `req` and `ret` parameters are not the same, a protocol address different than the one specified by `req` has been bound to the transport endpoint by the transport provider.

When the `t_bind()` function does not allocate a local transport protocol address (that is, automatic address generation is not supported), the protocol address pointed to by the `ret` parameter is always the same as the protocol address pointed to by the `req` parameter. In this case, values for variables pointed to by this parameter must be specified before the `t_bind()` function is called.

Points to a type `t_bind()` structure. The `addr` structure member returned by `t_bind()` specifies variables for the protocol address actually bound to the transport endpoint specified by the `fd` parameter. The bound address may be different than the address pointed to by the transport user with the `req->addr.buf` parameter.

The transport user must specify the maximum size in bytes of the protocol address with the `ret->addr.maxlen` parameter. On return, the `ret->addr.len` parameter specifies the actual number of bytes in the bound protocol address. When the `ret->addr.maxlen` parameter is not large enough to hold the returned protocol address, an error occurs.
The \textit{ret->qlen} parameter, which specifies the allowable number of outstanding connect indications that the transport provider can support, is meaningful only when initializing connection-oriented transport provider service.

**Description**

The \texttt{t_bind()} XTI function is used in connectionless and connection-oriented transport service to associate a protocol address with the transport endpoint returned by the \texttt{t_open()} function and to activate that transport endpoint. This function uses type \texttt{t_bind()} and \texttt{netbuf} structures, which are defined in the \texttt{xti.h} include file.

When connection-oriented transport service is in effect, and once this function has been called, the transport provider may begin enqueuing incoming connect indications or may service a connection request on the transport endpoint.

When connectionless transport service is in effect and once this function has been called, the transport user may send or receive data units through the transport endpoint.

**Return Value**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and \texttt{t_errno} is set to indicate the error.

**Errors**

If the \texttt{t_bind()} function fails, \texttt{t_errno} may be set to one of the following values:

- \texttt{TBADF} The specified file descriptor does not refer to a transport endpoint.
- \texttt{TOUTSTATE} The function was issued in the wrong sequence.
- \texttt{TBADADDR} The specified protocol address was in an incorrect format or contained illegal information.
- \texttt{TNOADDR} The transport provider could not allocate an address.
- \texttt{TACCES} The user does not have permission to use the specified address.
- \texttt{TBUFOVFLW} The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The provider's state will change to T_IDLE and the information to be returned in the \textit{ret} parameter will be discarded.
t_bind(3)

[TSYSERR] A system error occurred during execution of this function.

[TADDRBUSY] The address requested is in use and the transport provider could not allocate a new address.

Related Information

Functions: t_alloc(3), t_close(3), t_open(3), t_optmgmt(3), t_unbind(3)
t_close

Purpose  Closes a transport endpoint

Library  XTI Library (libtli.a)

Synopsis  
```
#include <xti.h>

int t_close(
   int fd);
```

Parameters  
The `t_close()` function is intended to be called in the T_UNBND transport provider state (see the DESCRIPTION section). The following table summarizes the relevance of the input parameter before and after `t_close()` is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td><code>y</code></td>
<td><code>n</code></td>
</tr>
</tbody>
</table>

Notes to table:
- `y` This is a meaningful parameter.
- `n` This is not a meaningful parameter.

`fd` Specifies a file descriptor returned by the `t_open()` function that identifies a local transport endpoint.

Description  
The `t_close()` XTI function is used in connection-oriented and connectionless transport service to inform a transport provider that the transport user has finished with the transport endpoint. The transport endpoint is specified by a file descriptor previously returned by the `t_open()` function. The `t_close()` function frees any local library resources associated with the transport endpoint referenced by the file descriptor.

The `t_close()` function does not check state information (see the `t_getstate()` function). Consequently, `t_close()` may be called in any transport provider state to close an open transport endpoint. When `t_close()` executes, local library resources associated with the transport endpoint are automatically freed. In addition,
close() function is called for the referenced file descriptor. The close() function aborts when there are no other file descriptors, in the current or any other process, that reference the same transport endpoint. When close() aborts, any connection that is associated with that transport endpoint is broken.

Return Value
Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate the error.

Errors
If the t_close() function fails, t_errno may be set to the following value:
[TBADF] File descriptor fd does not refer to a valid transport endpoint.

Related Information
Functions: t_getstate(3), t_open(3), t_unbind(3)
t_connect

**Purpose** Establishes a connection with another transport user

**Library**
XTI Library (libtli.a)

**Synopsis**
```c
#include <xti.h>

int t_connect(
    int fd,
    struct t_call *sndcall,
    struct t_call *rcvcall);
```

**Parameters**
The `t_connect()` function can only be called in the T_IDLE transport provider state. The following table summarizes the relevance of input and output parameters before and after `t_connect()` is called.
# Parameters Before Call After Call

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>sndcall-&gt;addr.maxlen</code></td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td><code>sndcall-&gt;addr.len</code></td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td><code>sndcall-&gt;addr.buf</code></td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td><code>sndcall-&gt;opt.maxlen</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>sndcall-&gt;opt.len</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>sndcall-&gt;opt.buf</code></td>
<td>o(o)</td>
<td>n</td>
</tr>
<tr>
<td><code>sndcall-&gt;udata.maxlen</code></td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td><code>sndcall-&gt;udata.len</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>sndcall-&gt;udata.buf</code></td>
<td>o(o)</td>
<td>n</td>
</tr>
<tr>
<td><code>sndcall-&gt;sequence</code></td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td><code>rcvcall-&gt;addr.maxlen</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>rcvcall-&gt;addr.len</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>rcvcall-&gt;addr.buf</code></td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td><code>rcvcall-&gt;opt.maxlen</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>rcvcall-&gt;opt.len</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>rcvcall-&gt;opt.buf</code></td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td><code>rcvcall-&gt;udata.maxlen</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>rcvcall-&gt;udata.len</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>rcvcall-&gt;udata.buf</code></td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td><code>rcvcall-&gt;sequence</code></td>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>

**Notes to table:**

- **y** This is a meaningful parameter.
- **n** This is not a meaningful parameter.
- **o** This is an optional parameter.
- **(o)** The content of the object pointed to by `o` is optional.

**fd**

Specifies a file descriptor returned by the `t_open()` function that identifies the local transport endpoint where the connection will be established.
Points to a type `t_call` structure. The `t_call` structure pointed to by the `sndcall` parameter provides information required by the transport provider to establish a connection at the transport endpoint specified by the `fd` parameter. The `t_call` structure has the following four members:

**struct netbuf addr**

Specifies protocol address parameters of the destination transport user needed by the transport provider. The type `netbuf` structure referenced by this member is defined in the `xti.h` include file. This structure, which is used to define buffer parameters explicitly, has the following members:

**unsigned int maxlen**

Specifies the maximum byte length of the data buffer.

**unsigned int len**

Specifies the actual byte length of the data written to the buffer.

**char *buf**

Points to the buffer location.

**struct netbuf opt**

Specifies protocol-specific information needed by the transport provider.

**struct netbuf udata**

Specifies user-data parameters passed to the destination transport user.

**int sequence**

This parameter is not meaningful.

The `sndcall->addr.maxlen`, `sndcall->opt.maxlen`, and `sndcall->udata.maxlen` parameters have no meaning when the `t_connect()` function is called.

When options are used, the `sndcall->opt.buf` parameter must specify the established options structure (such as isoco_options, isocl_options or tcp_options). A transport user may choose not to negotiate protocol options by setting the `sndcall->opt.len` parameter to 0 (zero). When options are not specified by the transport user, the transport provider has the option of returning default option values.
The amount of transport user data passed to the destination transport user must not exceed the limits specified by the transport provider as returned to the *info->connect* parameter of the *t_open()* or *t_getinfo()* function.

The *sndcall->opt.len* and *sndcall->udata.len* parameters must be set before the *t_connect()* function is called.

*rcveall* Points to a type *t_call* structure. The *t_call* structure pointed to by the *rcveall* parameter reserves storage for information associated with the connection established at the transport endpoint specified by the *fd* parameter. When *rcveall* is a null pointer, no data is returned to the caller. The structure pointed to by *rcveall* has the following members:

```c
struct netbuf addr
   Specifies protocol address parameters associated with the responding transport endpoint.

struct netbuf opt
   Specifies protocol-specific information associated with the transport provider.

struct netbuf udata
   Specifies parameters for user data that may be optionally returned to the caller from the destination transport user.

int sequence
   This parameter is not meaningful.
```

The *rcveall->addr.maxlen*, *rcveall->opt.maxlen*, and *rcveall->udata.maxlen* parameters must be set before the *t_connect()* function is called.

When it is provided, the *rcveall->udata.len* parameter specifies the actual destination user-data byte length and the data buffer pointed to by *rcveall->udata.buf* contains destination transport user data.

**Description**

The *t_connect()* is XTI function is a connection-oriented service function issued by a transport user to request connection to the specified destination transport user. By default, this function executes in the synchronous operating mode. In this mode, the *t_connect()* function waits for the destination user to respond and the connection to be set up before returning control to the transport user who called this function.
When the transport endpoint, specified by the file descriptor, has been previously opened with the O_NONBLOCK flag set in the t_open() or fcntl() function, the t_connect() function executes in asynchronous mode and does not wait for the transport user at the specified endpoint to respond before returning control to the caller, but returns a [TNODATA] error, which indicates that the connection has not yet been established. In asynchronous mode, use the t_rcvconnect() function to determine the status of a connect request.

The t_connect() function uses type t_call and netbuf structures, which are defined in the xti.h include file.

Return Value
Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.

Errors
If the t_connect() function fails, t_errno may be set to one of the following values:

[TBADF] File descriptor fd does not refer to a valid transport endpoint.
[TOUTSTATE] The t_connect() function was issued in the wrong sequence.
[TNODATA] Asynchronous mode is indicated because O_NONBLOCK was set, but no data is currently available from the transport provider.
[TBADADDR] The specified protocol address was in an incorrect format or contained illegal information.
[TBADOPT] The specified protocol options were in an incorrect format or contained illegal information.
[TBADDRDATA] The amount of user data specified was not within the bounds allowed by the transport provider.
t_connect(3)

[TACCESS] The user does not have permission to use the specified protocol address or options.

[TBUFOVFLW]
The number of bytes allocated for incoming data is not sufficient for storage of that data. In asynchronous mode only, the connect information normally returned to the \texttt{rcvcall} function was discarded. The transport provider state was changed to T\_DATAXFER.

[TLOOK] An asynchronous event that requires immediate attention has occurred on the transport endpoint specified by the \texttt{fd} parameter.

[TSYSERR] A system error occurred during execution of this function.

Related Information

Functions: \texttt{fcntl(2)}, \texttt{t_accept(3)}, \texttt{t_alloc(3)}, \texttt{t_getinfo(3)}, \texttt{t_listen(3)}, \texttt{t_open(3)}, \texttt{t_optmgmt(3)}, \texttt{t_rcvconnect(3)}
t_error

Purpose Produces error message

Library XTI Library (libtli.a)

Synopsis #include <xti.h>

int t_error(
    char *errmsg);
    extern char *t_errlist[] ;
    extern int t_nerr ;

Parameters

The t_errno() function can be called in any transport provider state except T_UNINIT. The following table summarizes the relevance of input parameter data before and after t_error() is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>errmsg</td>
<td>y</td>
<td>n</td>
</tr>
</tbody>
</table>

Notes to table:

y       This is a meaningful parameter.

n       This is not a meaningful parameter.

errmsg Points to a user-supplied error message character string that lends proper context to the nature of the detected error.

Description

The t_error() function is a general utility function used to produce an error message on the standard error output device. The error message describes the last error encountered during execution of an XTI function. The user-supplied error message is printed, followed by a colon and a standard error message for the current error defined in t_errno. When t_errno is [TSYSERR], t_error() also prints a standard error message for the current value contained in errno. The error number, t_errno, is set only when an error occurs and is not cleared when XTI functions execute successfully.
To simplify variant formatting of messages, the array of message strings named \texttt{t_errlist} is specified. Variable \texttt{t_errno} may be used as an index into this table to get a relevant message string without an ending newline character. External variable \texttt{t_nerr} specifies the maximum number of messages in the \texttt{t_errlist} table.

**Return Value**
Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and \texttt{t_errno} is set to indicate an error.

**Errors**
The \texttt{t_error( )} function does not have any error numbers.

**Related Information**
Functions: \texttt{t_accept(3)}, \texttt{t Alloc(3)}, \texttt{t Bind(3)}, \texttt{t Close(3)}, \texttt{t Connect(3)}, \texttt{t Free(3)}, \texttt{t Getinfo(3)}, \texttt{t Getstate(3)}, \texttt{t Listen(3)}, \texttt{t Look(3)}, \texttt{t Open(3)}, \texttt{t Optmgmt(3)}, \texttt{t Rcv(3)}, \texttt{t Rcvconnect(3)}, \texttt{t Rcvdis(3)}, \texttt{t Rcvrel(3)}, \texttt{t Rcvudata(3)}, \texttt{t Rcvuderr(3)}, \texttt{t SND(3)}, \texttt{t Snddis(3)}, \texttt{t Sndrel(3)}, \texttt{t Sndudata(3)}, \texttt{t Sync(3)}, \texttt{t Unbind(3)}
t_free

Purpose  Frees a library structure

Library  XTI Library (libtli.a)

Synopsis  
```
#include <xti.h>

int t_free(
    char *ptr,
    int struct_type);
```

Parameters  
The `t_free()` function can be called in all transport provider states. The following table summarizes the relevance of input parameter data before and after `t_free()` is called:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ptr</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>struct_type</code></td>
<td>y</td>
<td>n</td>
</tr>
</tbody>
</table>

Notes to table:

- **y**: This is a meaningful parameter.
- **n**: This is not a meaningful parameter.
- `ptr` Points to one of the seven structure types described for structures previously named by the `struct_type` parameter of the `t_alloc()` function, listed below.
struct_type Specifies the structure type for functions for which memory was previously allocated. This parameter must be one of the symbolic names listed in the following table:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Structure</th>
<th>Using Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_BIND_STR</td>
<td>struct t_bind</td>
<td>t_bind()</td>
</tr>
<tr>
<td>BT_CALL_STR</td>
<td>struct t_call</td>
<td>t_accept(),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t_connect(), t_listen(),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t_rcvconnect(),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t_snddis()</td>
</tr>
<tr>
<td>T_OPTMGMT_STR</td>
<td>struct t_optmgmt</td>
<td>t_optmgmt()</td>
</tr>
<tr>
<td>T_DIS_STR</td>
<td>struct t_discon</td>
<td>t_rcvdis()</td>
</tr>
<tr>
<td>T_UNITDATA_STR</td>
<td>struct t_unitdata</td>
<td>t_rcvudata(),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t_sndudata()</td>
</tr>
<tr>
<td>T_UDERROR_STR</td>
<td>struct t_uderr</td>
<td>t_rcvuderr()</td>
</tr>
<tr>
<td>T_INFO_STR</td>
<td>struct t_info</td>
<td>t_info()</td>
</tr>
</tbody>
</table>

Any structure symbolic name listed in the preceding table may be used as an argument to deallocate previously reserved memory. Each of the structures, except t_info, contains at least one member of type struct netbuf structure, which is defined in the xti.h include file.

This function checks all members of a netbuf structure and deallocates those buffers. When a netbuf structure buf parameter is a null pointer, no memory is deallocated. After all buffers are deallocated, this function frees all memory referenced by the ptr parameter.

Description

The t_free() function is an XTI general utility function used to deallocate memory buffers previously allocated with the t_alloc() function. When executed, t_free() deallocates memory for the named structure and for any buffers referenced by the named structure. When t_free() is executed, undefined results are obtained when structure pointers or buffer pointers point to memory blocks not previously allocated with the t_alloc() function.
Return Value
Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate the error.

Errors
If the t_free() function fails, t_errno may be set to the following value:
[T SYSERR] A system error occurred during execution of this function.

Related Information
Functions: t_alloc(3)
**t_getinfo**

**Purpose**  
Gets protocol-specific information

**Library**  
XTI Library (libtli.a)

**Synopsis**  
```c
#include <xti.h>
#include <fcntl.h>

int t_getinfo(
    int fd,
    struct t_info *info);
```

**Parameters**

The `t_getinfo()` function can be called in any transport provider state except T_UNINIT. The following table summarizes the relevance of input and output parameter data before and after `t_info()` is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>info-&gt;addr</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>info-&gt;options</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>info-&gt;tsdu</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>info-&gt;etsdu</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>info-&gt;connect</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>info-&gt;discon</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>info-&gt;servtype</code></td>
<td>n</td>
<td>y</td>
</tr>
</tbody>
</table>

**Notes to table:**

- **y** This a meaningful parameter.
- **n** This is not a meaningful parameter.

- `fd` Specifies a file descriptor returned by the `t_open()` function that identifies the local transport endpoint.

- `info` Points to a type `t_info` structure that is returned when `t_getinfo()` executes. Parameters defined by the `t_info` structure specify characteristics of the underlying transport protocol associated with the `fd` file descriptor.
When the \textit{info} parameter is set to the null pointer value by a transport user, no protocol information is returned by the \texttt{t\_getinfo()} function.

When a transport user must preserve protocol independence, data length information defined by members of the \texttt{t\_info} structure pointed to by the \textit{info} parameter may be accessed to determine how large data buffers must be to hold exchanged data. Alternatively, the \texttt{t\_alloc()} function may be used to allocate necessary data buffers. An error results when a transport user exceeds the allowed data length during any data exchange.

Values associated with parameters of the \texttt{t\_info} structure may change as the result of protocol option negotiations during initialization of a connection. The \texttt{t\_info} structure has the following seven members:

\begin{description}
\item[addr] Specifies the permitted number of bytes in the protocol address. A value greater than or equal to zero indicates the maximum number of permitted bytes in a protocol address. A value of -1 specifies that there is no limit on the protocol address size. A value of -2 specifies that the transport provider does not permit the transport user access to the protocol addresses.
\item[options] Specifies the permitted number of bytes of options. A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the transport provider. A value of -1 specifies that there is no limit to the number of options bytes. A value of -2 specifies that the transport provider does not permit a transport user to set options.
\item[tsdu] Specifies the permitted number of bytes in a Transport Service Data Unit (TSDU). A value greater than zero specifies the maximum number of bytes in a TSDU message. A value of zero specifies that the transport provider does not support TSDU data exchanges, although it does support the sending of a data stream with no logical boundaries preserved across a connection. A value of -1 specifies that there is no limit to the number of bytes in a TSDU data exchange. A value of -2 specifies that the transfer of normal data is not supported by the transport provider.
\end{description}
etsdu Specifies the permitted number of bytes in an Expedited Transport Service Data Unit (ETSDU). A value greater than zero specifies the maximum number of bytes in an ETSDU data exchange. A value of zero specifies that the transport provider does not support ETSDU data exchanges, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection. A value of -1 specifies that there is no limit on the number of bytes in an ETSDU data exchange. A value of -2 specifies that the transfer of expedited data is not supported by the transport provider.

connect Specifies the permitted number of bytes of data in a connect request. A value greater than or equal to zero specifies the maximum number of data bytes that may be exchanged using the t_connect() or t_rcvconnect() function. A value of -2 specifies that there is no limit to the number of data bytes that may be sent when a connection is requested. A value of -2 specifies that the transport provider does not permit data to be sent when a connection is established.

discon Specifies the permitted number of bytes of data in a disconnect request. A value greater than or equal to zero specifies the maximum number of data bytes that may be exchanged using the t_snddis() or t_rcvdis() function. A value of -1 specifies that there is no limit to the number of data bytes that may be sent when a connection is closed using these abortive release functions. A value of -2 specifies that the transport provider does not permit data to be sent with an abortive release function.

servtype Specifies only one of the following types of service supported by the transport provider:

T_COTS The transport provider supports connection-mode service but does not support the optional orderly release facility.

T_COTS_ORD The transport provider supports connection-mode service with the optional orderly release facility.

T_CLTS The transport provider supports connectionless mode service. For this service type, this function returns the value -2 for the etsdu, connect, and discon parameters.
Description
The t_getinfo() function is an XTI general utility function that provides information about the underlying transport protocol associated with a file descriptor previously returned by the t_open() function. The t_getinfo() function returns the same protocol-specific information as does t_open() in the info parameter.

Return Value
Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate the error.

Errors
If the t_getinfo() function fails, t_errno may be set to one of the following values:
[TBADF] File descriptor fd does not refer to a valid transport endpoint.
[TSYSERR] A system error occurred during execution of this function.

Related Information
Functions: t_alloc(3), t_open(3)
**t_getstate**

**Purpose**

Gets the current state of the transport provider

**Library**

XTI Library (libtli.a)

**Synopsis**

```
#include <xti.h>

int t_getstate(
  int fd);
```

**Parameters**

The `t_getstate()` function can be called in all transport provider states except T_UNINIT. The following table summarizes the relevance of input parameter data before and after the `t_getstate()` function is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>y</td>
<td>n</td>
</tr>
</tbody>
</table>

**Notes to Table:**

- y: This is a meaningful parameter.
- n: This is not a meaningful parameter.

- `fd` Specifies a file descriptor returned by the `t_open()` function that identifies the local transport endpoint.

**Description**

The `t_getstate()` function is a general utility function used to get the current state of the transport provider. The transport endpoint, which is specified by a file descriptor, is regarded as a finite-state machine that may be in any one of eight states. When the `t_getstate()` function is executed, the current state of the transport endpoint is returned.

**Notes**

If the transport provider is undergoing a change in state when `t_getinfo()` is called, a failure occurs.
Functions

t_getstate(3)

Return Value

Upon successful completion, the transport endpoint state is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate the error. The current state is one of the following:

[T_UNBND] Address not bound to transport endpoint.
[T_IDLE] The transport endpoint is inactive.
[T_OUTCON] Outgoing connection pending.
[T_INCON] Incoming connection pending.
[T_DATAXFER] Data transfer in progress.
[T_OUTREL] Outgoing orderly release (waiting for an orderly release indication).
[T_INREL] Incoming orderly release (waiting to send an orderly release request).

Errors

If the t_getstate() function fails, t_errno may be set to one of the following values:

[TBADF] The specified file descriptor does not refer to a transport endpoint. This error may be returned when the endpoint referenced by the fd parameter has been previously closed or an erroneous file descriptor value has been provided.
[TSTATECHNG] The transport provider is undergoing a change in state.
[TSYSERR] A system error occurred during execution of this function.
t_listen

Purpose
Listens for a connect request

Library
XTI Library (libtli.a)

Synopsis
```
#include <xti.h>

int t_listen(
    int fd, struct t_call *call);
```

Parameters
The `t_listen()` function can only be called in the T_IDLE and T_INCON transport provider states. The following table summarizes the relevance of input and output parameters before and after `t_listen()` is called:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;addr.maxlen</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;addr.len</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>call-&gt;addr.buf</code></td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td><code>call-&gt;opt.maxlen</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;opt.len</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>call-&gt;opt.buf</code></td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td><code>call-&gt;udata.maxlen</code></td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td><code>call-&gt;udata.len</code></td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td><code>call-&gt;udata.buf</code></td>
<td>y</td>
<td>(o)</td>
</tr>
<tr>
<td><code>call-&gt;sequence</code></td>
<td>n</td>
<td>y</td>
</tr>
</tbody>
</table>

Notes to Table
- y This is a meaningful parameter.
- n This is not a meaningful parameter.
- (y) The content of the object pointed to by y is meaningful.
- (o) The content of the object pointed to by o is optional.

`fd`
Specifies a file descriptor returned by the `t_open()` function that identifies the local transport endpoint where connect indication may arrive.
Functions

`t_listen(3)`

call

Points to a type `t_call` structure used to specify information that describes the connect indication. The `t_call` structure has the following four members:

```c
struct netbuf addr
```

Specifies a buffer for protocol address information sent by the calling transport user. The type `netbuf` structure referenced by this member is defined in the `xti.h` include file. This structure, which is used to define buffer parameters, has the following members:

```c
unsigned int maxlen
```

Specifies the maximum byte length of the data buffer.

```c
unsigned int len
```

Specifies the actual byte length of data written to the buffer. char *buf Points to the buffer location.

```c
struct netbuf opt
```

Specifies a buffer for protocol-specific parameters associated with the connect request.

```c
struct netbuf udata
```

Specifies a buffer for user data sent by the caller.

```c
int sequence
```

Specifies a unique identification number used to identify the returned connect indication.

The `sequence` parameter pointed to by the `call` parameter is used to uniquely identify the returned connection indication. Values greater than 1 for this parameter enable the transport user to listen for more than a single connect indication before responding to any of those returned.

Each `maxlen` parameter must be set before calling this function to indicate the maximum size of the buffer associated with values sent by the caller.

Description

The `t_listen()` function is an XTI connection-oriented service function that listens for a connect request from a calling transport user. The transport endpoint where the connect indications arrive is specified by a file descriptor previously returned by the `t_open()` function. By default, the `t_listen()` function executes in the synchronous operating mode. In the synchronous operating mode, `t_listen()` waits for a connect indication to arrive before returning control to the transport user who called this function.
When the transport endpoint specified by the `fd` file descriptor has been opened with the O_NONBLOCK flag set when the `t_open()` or `fcntl()` function is called, the `t_listen()` function executes in asynchronous mode.

When the `t_listen()` function executes in asynchronous mode, it does not wait for a connect indication before returning control to the caller, but returns a [TNODATA] error if a connection request has not yet been received.

The `t_listen()` function returns a pointer to a type `t_call` structure, which defines information associated with the arriving connect request. The `t_call` structure also references a type `netbuf` structure. Both structures are defined in the `xti.h` include file.

Notes

When operation is set for the asynchronous mode, and no connect indications are available, the `t_listen()` function fails, the value -1 is returned, and `t_errno` is set to [TNODATA].

Return Value

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate the error.

Errors

If the `t_listen()` function fails, `t_errno` may be set to one of the following values:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TBADQLEN] The `qlen` argument of the endpoint referenced by the `fd` parameter is zero.

[TBUFOVFLW] The number of bytes allocated for incoming information is not sufficient to store the value of that information. The transport provider state, as seen by the transport user, changes to T_INCON, and connect indication information, normally returned to the structure pointed to by the `call` parameter, is discarded. The returned value of `call->sequence` may be used to call a `t_snddis()` function.

[TNODATA] The O_NONBLOCK flag was set, but no connect indications had been queued.
An asynchronous event has occurred on this transport endpoint and requires immediate attention.

The function was issued in the wrong sequence on the transport endpoint referenced by the `fd` parameter.

A system error has occurred during execution of this function.

Related Information

Functions: `fcntl(2), t_accept(3), t_alloc(3), t_bind(3), t_connect(3), t_open(3), t_optmgmt(3), t_rcvconnect(3), t_snddis(3)`
t_look

Purpose
Looks at the current event on a transport endpoint

Library
XTI Library (libtli.a)

Synopsis
#include <xti.h>

int t_look(
    int fd);

Parameters
The t_look() function can be called in all transport provider states except T_UNINIT. The following table summarizes the relevance of the fd parameter when t_look() is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>y</td>
<td>n</td>
</tr>
</tbody>
</table>

Notes to Table
- y This is a meaningful parameter.
- n This is not a meaningful parameter.

fd Specifies a file descriptor returned by the t_open() function that identifies the local transport endpoint.

Description
The t_look() XTI function is used in connectionless and connection-oriented transport service to monitor the current event at the transport endpoint specified by a file descriptor previously returned by the t_open() function. The t_look() function permits a transport provider to notify a transport user of any one of the nine asynchronous events listed in the RETURN VALUE section when the transport user is calling other XTI functions in synchronous mode.

During synchronous operation, all events at a transport endpoint are saved by XTI so that any current event may be known to a transport user. Each of the nine asynchronous events listed under the RETURN VALUE section is defined by a symbolic name in the xti.h include file. This symbolic name can be retrieved when the t_look() function is called.
Some XTI functions fail unconditionally when they are called because the current event at the transport endpoint does not permit them to successfully execute. Four of the nine synchronous events listed in the Event column of the following table cause unconditional failure when any function listed in the Immediate T_LOOK Functions column is called. Any of these four synchronous events requires that the transport user be immediately notified. Unconditional failure returns a [T_LOOK] error during execution of the currently called function or the next called function when it is executed. This function can then be used to determine which event occurred.

<table>
<thead>
<tr>
<th>Event</th>
<th>Immediate T_LOOK Functions</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_LISTEN</td>
<td>t_accept(), t_connect(), t_unbind()</td>
<td>Connection indication received</td>
</tr>
<tr>
<td>T_DISCONNECT</td>
<td>t_accept(), t_connect(), t_listen(), t_rcv(), t_rcvconnect(), t_rcvrel(), t_snd(), t_sndrel()</td>
<td>Disconnect received</td>
</tr>
<tr>
<td>T_UDERR</td>
<td>t_rcvudata(), t_sndudata()</td>
<td>Datagram error indication</td>
</tr>
<tr>
<td>T_ORDREL</td>
<td>t_rcvudata(), t_sndudata()</td>
<td>Orderly release indication</td>
</tr>
</tbody>
</table>

Notes to Table

1. Connection indication received at a transport endpoint which has been bound with qlen > 0 (zero) and for which a connection indication is pending (refer to the t_bind() function).

2. Disconnect for an outstanding connect indication.

When multiple events occur, the order in which their value is returned is implementation dependent. All together, there are 11 XTI functions that fail when a particular synchronous event requiring immediate notification is detected.
The following table lists transport endpoint events and corresponding functions to which a [T_LOOK] error is immediately returned when the event causes function failure:

<table>
<thead>
<tr>
<th>Event</th>
<th>Cleared with T_LOOK?</th>
<th>Event Consuming Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_LISTEN</td>
<td>No</td>
<td>t_listen()</td>
</tr>
<tr>
<td>T_CONNECT</td>
<td>No</td>
<td>t_rcvconnect()</td>
</tr>
<tr>
<td>T_DATA</td>
<td>No</td>
<td>t_rcv(), t_rcvudata()</td>
</tr>
<tr>
<td>T_EXDATA</td>
<td>No</td>
<td>t_rcv()</td>
</tr>
<tr>
<td>T_DISCONNECT</td>
<td>No</td>
<td>t_rcvdis()</td>
</tr>
<tr>
<td>T_UDERR</td>
<td>No</td>
<td>t_rcvuderr()</td>
</tr>
<tr>
<td>T_ORDREL</td>
<td>No</td>
<td>t_rcvrel()</td>
</tr>
<tr>
<td>T_GODATA</td>
<td>Yes</td>
<td>t_snd(), t_sndudata()</td>
</tr>
<tr>
<td>T_GOEXDATA</td>
<td>Yes</td>
<td>t_snd()</td>
</tr>
</tbody>
</table>

An event at a transport endpoint remains outstanding until a consuming function clears it. Every event has an associated consuming function that handles the event and clears it. The Event Consuming Function column of the preceding table lists these events and the function that clears each one when successfully executed.

Return Value

Upon successful completion, the t_look() function returns one of the following values. Upon failure, 0 (zero) is returned.

[T_LISTEN] Connect indication received.
[T_CONNECT] Connect confirmation received.
[T_DATA] Normal data received.
[T_EXDATA] Expedited data received.
[T_DISCONNECT] Disconnect received.
[T_UDERR] Datagram error indication.
[T_ORDREL] Orderly release indication.
Flow control restrictions on normal data flow have been lifted. Normal data may be sent again.

Flow control restrictions on expedited data flow have been lifted. Expedited data may be sent again.

Upon failure, the value -1 is returned and `t_errno` is set to indicate the error.

Errors

If the `t_look()` function fails, `t_errno` is set to one of the following values:

- [TBADF] The specified file descriptor does not refer to a transport endpoint.
- [TSYSERR] A system error occurred during execution of this function.

Related Information

Functions:
- `t_bind(3)`, `t_connect(3)`, `t_listen(3)`, `t_open(3)`, `t_rcv(3)`, `t_rcvconnect(3)`, `t_rcvdis(3)`, `t_rcvrel(3)`, `t_rcvudata(3)`, `t_rcvuderr(3)`, `t_snd(3)`, `t_sndudata(3)`)
t_open

Purpose
Establishes a transport endpoint

Library
XTI Library (libtli.a)

Synopsis
#include <xti.h>
#include <fcntl.h>

int t_open(
    char *path,
    int oflag,
    struct t_info *info);

Parameters
The t_open() function can be called in the T_UNINIT transport provider state only. The following table summarizes the relevance of input and output parameters before and after the t_open() function is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>oflag</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>info-&gt;addr</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>info-&gt;options</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>info-&gt;tsdu</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>info-&gt;etsdu</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>info-&gt;connect</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>info-&gt;discon</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>info-&gt;servtype</td>
<td>n</td>
<td>y</td>
</tr>
</tbody>
</table>

Notes to Table:
- y This is a meaningful parameter.
- n This is not a meaningful parameter.

path Identifies the transport provider. The transport provider must define the type of transport service (protocol) to associate with the opened transport endpoint.
**oflag**

The `oflag` parameter is similar to the `oflag` parameter of the `open()` function and is used in the same way. Use `oflag` to establish synchronous or asynchronous operating modes of the transport provider pointed to by the `path` parameter. The transport provider operating mode is specified with the O_NONBLOCK flag. The actual value for this parameter is obtained from the symbolic name variable O_RDWR, which may be optionally bitwise combined with a logical inclusive OR of flag O_NONBLOCK, defined in the `fcntl.h` include file.

**info**

Points to a type `t_info` structure. The location of a type `t_info` structure is returned to the `info` parameter when the `t_open()` function successfully executes. Members of the `t_info` structure specify default characteristics of the underlying transport protocol pointed to by the `path` parameter.

When the `info` parameter is set to the null pointer value by a transport user, no protocol information is returned by this function.

When a transport user must preserve protocol independence, data length information defined by members of the type `t_info` structure may be accessed to determine how large data buffers must be to hold exchanged data. Alternatively, the `t_alloc()` function may be used to allocate necessary data buffers. An error results when a transport user exceeds the allowed data length during any data exchange. This structure has the following seven members:

- **addr** Permitted number of bytes in the protocol address. A value greater than or equal to zero indicates the maximum number of permitted bytes in a protocol address. A value of -1 specifies that there is no limit on the protocol address size. A value of -2 specifies that the transport provider does not permit the transport user access to the protocol addresses.

- **options** Permitted number of bytes of options. A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the transport provider. A value of -1 specifies that there is no limit to the number of options bytes. A value of -2 specifies that the transport provider does not permit a transport user to set options.

- **tsdu** Permitted number of bytes in a Transport Service Data Unit (TSDU). A value greater than zero specifies the maximum number of bytes in a TSDU message. A value of zero specifies that the transport provider does not support TSDU
data exchanges, although it does support the sending of a data stream with no logical boundaries preserved across a connection. A value of -1 specifies that there is no limit to the number of bytes in a TSDU data exchange. A value of -2 specifies that the transfer of normal data is not supported by the transport provider.

etsdu Permitted number of bytes in an Expedited Transport Service Data Unit (ETSDU). A value greater than zero specifies the maximum number of bytes in an ETSDU data exchange. A value of zero specifies that the transport provider does not support ETSDU data exchanges, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection. A value of -1 specifies that there is no limit on the number of bytes in an ETSDU data exchange. A value of -2 specifies that the transfer of expedited data is not supported by the transport provider.

connect Permitted number of bytes of data in connect request. A value greater than or equal to zero specifies the maximum number of data bytes that may be exchanged using the t_connect() and t_rcvconnect() functions. A value of -2 specifies that there is no limit to the number of data bytes that may be sent when a connection is requested. A value of -2 specifies that the transport provider does not permit data to be sent when a connection is established. discon Permitted number of bytes of data in a disconnect request. A value greater than or equal to zero specifies the maximum number of data bytes that may be exchanged using the t_snddis() and t_rcvdis() functions. A value of -1 specifies that there is no limit to the number of data bytes that may be sent when a connection is closed using these abortive release functions. A value of -2 specifies that the transport provider does not permit data to be sent with an abortive release function.
servtype
This member specifies only one of the following types of service supported by the transport provider:

T_COTS
The transport provider supports connection-mode service but does not support the optional orderly release facility.

T_COTS_ORD
The transport provider supports connection-mode service with the optional orderly release facility.

T_CLTS
The transport provider supports connectionless-mode service. For this service type, this function returns the value -2 for the etsdu, connect, and discon parameters.

Description
The t_open() XTI function must be the first one called when initializing a transport endpoint. Two modes of operation may be specified, synchronous and asynchronous. In synchronous mode, a transport user must wait for some specific event to occur before control is returned (refer to the t_look() function). In asynchronous mode, a transport user is not required to wait for the event to occur; control is returned immediately.

The t_open() function establishes the transport endpoint by supplying a transport provider identifier that specifies a particular transport protocol. A file descriptor, which must subsequently always be used to identify the established endpoint, is returned by this function.

Return Value
Upon successful completion, the function returns 0 (zero). Otherwise, a value of -1 is returned and t_errno is set to indicate the error.
t_open(3)

Errors
If the t_open() function fails, t_errno may be set to one of the following values:
[TBADFLAG]
   An invalid flag is specified.
[TBADNAME]
   Invalid transport provider name.
[TSYSERR]   A system error occurred during execution of this function.

Related Information
Functions: open(2)
t_optmgmt

Purpose  Manages protocol options for a transport endpoint

Library  XTI Library (libtli.a)

Synopsis  
#include <xti.h>

int t_optmgmt(
    int fd,
    struct t_optmgmt *req,
    struct t_optmgmt *ret);

Parameters  
The t_optmgmt() function can only be called in the T_IDLE transport provider state. The following table summarizes the relevance of input and output parameters before and after t_optmgmt() is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>req-&gt;opt.maxlen</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>req-&gt;opt.len</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>req-&gt;opt.buf</td>
<td>y(y)</td>
<td>n</td>
</tr>
<tr>
<td>req-&gt;flags</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>ret-&gt;opt.maxlen</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>ret-&gt;opt.len</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>ret-&gt;opt.buf</td>
<td>y (y)</td>
<td>(y)</td>
</tr>
<tr>
<td>ret-&gt;flags</td>
<td>n</td>
<td>—</td>
</tr>
</tbody>
</table>

Notes to Table:

- y This a meaningful parameter.
- n This is not a meaningful parameter.
- (y) The content of the object pointed to by y is meaningful.

fd Specifies a file descriptor returned by t_open() function that identifies the local transport endpoint.
Points to a type `t_optmgmt` structure. This structure is used to reserve space for a transport-user options data buffer that stores negotiable protocol options. The type `t_optmgmt` structure has the following members:

```c
struct netbuf opt
```

Specifies a buffer for protocol-optional parameters associated with the referenced transport endpoint. The type `netbuf` structure pointed to by this member is defined in the `xti.h` include file. This structure, which is used to define buffer parameters, has the following members:

- **unsigned int maxlen**
  Specifies maximum byte length of the data buffer.

- **unsigned int len**
  Specifies the actual byte length of data written to the buffer.

- **char *buf**
  Points to the buffer location.

**flags**
A longword (least significant bit rightmost) that specifies the response action that must be taken by a transport provider when the `t_optmgmt()` function is processed. Corresponding values and symbolic names for the following flag bits are defined in the `xti.h` include file. Note that the `flags` parameter can specify only one of these values, not a combination.
### Functions
t_optmgmt(3)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Symbolic Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>T_NEGOTIATE</td>
<td>The transport user wants to negotiate the values of the options stored in the options buffer. In response, the transport provider evaluates the options and writes acceptable (negotiated) values to the data buffer pointed to by ret-&gt;opt.buf.</td>
</tr>
<tr>
<td>3</td>
<td>T_CHECK</td>
<td>The transport user wants to verify that the options specified in the data buffer pointed to by req-&gt;opt.buf are supported by the transport provider. On return, the transport provider writes a ret-&gt;flags value, which is either T_SUCCESS or T_FAILURE.</td>
</tr>
<tr>
<td>4</td>
<td>T_DEFAULT</td>
<td>The transport user wants to know what the default options supported by the transport provider are. The transport provider writes default data into the options data buffer pointed to by ret-&gt;opt.buf. The req-&gt;opt.len parameter must be set to 0 (zero). The req-&gt;opt.buf member may be set to its null value.</td>
</tr>
</tbody>
</table>

**ret**

Points to a second type t_optmgmt structure. The ret->opt.maxlen parameter specifies the maximum length of the transport provider options data buffer. The ret->opt.len parameter specifies the actual length of the transport provider options data buffer. The ret->opt.buf parameter points to the transport provider options data buffer. On return, if T_CHECK was specified in req->flags, the ret->flags parameter is set to T_SUCCESS or T_FAILURE, indicating whether the transport provider supports the options specified by the transport user.
Description
The \texttt{t\_optmgmt()} XTI function is used in connectionless and connection-oriented transport service. The \texttt{t\_optmgmt()} function associates specific optional parameters with a bound transport endpoint previously defined by a file descriptor returned by the \texttt{t\_open()} function. The \texttt{t\_optmgmt()} function permits a transport user to retrieve, verify, or negotiate desired options with a transport provider.

A type \texttt{t\_optmgmt} structure defined in the \texttt{xti.h} include file is used to specify options.

Return Value
Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and \texttt{t\_errno} is set to indicate the error.

Errors
If the \texttt{t\_optmgmt()} function fails, \texttt{t\_errno} may be set to one of the following values:

- \texttt{[TBADF]} File descriptor \texttt{fd} does not refer to a valid transport endpoint.
- \texttt{[TOUTSTATE]} This function was called in the wrong sequence.
- \texttt{[TBADOPT]} The specified protocol options are either of an incorrect format or contain illegal information.
- \texttt{[TBADFLAG]} The specified flag is invalid.
- \texttt{[TACCES]} The transport user does not have permission to negotiate the specified options.
- \texttt{[TBUFOVFLW]} The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The information intended for the data buffer pointed to by the \texttt{ret} parameter is discarded.
- \texttt{[TSYSERR]} A system error occurred during execution of the \texttt{t\_optmgmt()} function.

Related Information
Functions: \texttt{t\_accept(3), t\_alloc(3), t\_connect(3), t\_getinfo(3), t\_listen(3), t\_open(3), t\_rcvconnect(3)}
t_rcv

Purpose
Receives normal data or expedited data on a connection

Library
XTI Library (libtli.a)

Synopsis
#include <xti.h>

int t_rcv(
    int fd,
    char *buf,
    unsigned nbytes,
    int *flags);

Parameters
The t_rcv() function can only be called in the T_DATAXFER and T_OUTREL
transport provider states. The following table summarizes the relevance of input
and output parameters before and after t_rcv() is called:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>buf</td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td>nbytes</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>flags</td>
<td>n</td>
<td>y</td>
</tr>
</tbody>
</table>

Notes to Table:
y This is a meaningful parameter.
n This is not a meaningful parameter.
(y) The content of the object pointed to by y is meaningful.

fd Specifies a file descriptor returned by the t_open() function that
identifies the local transport endpoint where an active connection
exists.

buf Points to the receive data buffer where returned data is to be written.
nbytes Specifies the length in bytes of the received-data buffer pointed to
by the buf parameter.
Points to an unsigned integer (least significant bit rightmost) whose bits are flags that specify the action that must be taken by the responding transport user when the `t_rcv()` function is processed. Corresponding values and symbolic names for the following flag bits are defined in the `xti.h` include file:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Symbolic Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T_MORE</td>
<td>When set, this bit notifies the transport user that received data is a fragment of a Transport Service Data Unit (TSDU) or Expedited Transport Service Data Unit (ETSDU), and that more data is available. The rest of the TSDU or ETSDU can be received through further <code>t_rcv()</code> function calls. Each time this flag is set on return, another <code>t_rcv()</code> call can receive additional pieces of the TSDU or ETSDU. When the final TSDU or ETSDU is received, this flag bit has a value of 0 (zero) on return. When the transport provider does not support TSDU or ETSDU data exchanges (refer to the <code>t_open()</code> and <code>t_getinfo()</code> functions), the state of this flag bit should be ignored. When an ETSDU is received during reception of a TSDU, no remaining pieces of the TSDU may be received until the current ETSDU has been completely received.</td>
</tr>
<tr>
<td>1</td>
<td>T_EXPEDITED</td>
<td>When set, this bit notifies the transport user that received data is an ETSDU. When the number of ETSDU data bytes exceeds the value specified by the <code>nbytes</code> parameter, this flag bit and the T_MORE flag bit is set on return of the initial <code>t_rcv()</code> call. Subsequent <code>t_rcv()</code> calls issued to retrieve the rest of the ETSDU have both these flag bits set on return. When the final piece of the ETSDU is received, the T_MORE flag bit has a value of 0 (zero) on return.</td>
</tr>
</tbody>
</table>

1–832
Functions

**t_rcv(3)**

### Description

The `t_rcv()` function is an XTI connection-oriented service function that is used to receive normal or expedited data. The transport endpoint through which data arrives is specified by a file descriptor previously returned by the `t_open()` function. By default, `t_rcv()` executes in the synchronous operating mode. In synchronous mode `t_rcv()` waits for data to arrive even when none is currently available before returning control to the calling transport user.

When the transport endpoint, specified by the `fd` parameter, has been opened with the O_NONBLOCK flag set in the `t_open()` or `fcntl()` functions, the `t_rcv()` function executes in asynchronous mode. In asynchronous mode, when no data is available, this function fails.

### Notes

In synchronous mode, the only way for a transport user to be notified of the arrival of normal or expedited data is to call the `t_rcv()` function or to check for states T_DATA or T_EXDATA using the `t_look()` function.

### Return Value

Upon successful completion, the `t_rcv()` function returns the number of bytes of data received. Otherwise, the value -1 is returned and `t_errno` is set to indicate the error.

### Errors

If the `t_rcv()` function fails, `t_errno` is set to one of the following values:

- **[TBADF]** The specified file descriptor does not refer to a valid transport endpoint.
- **[TNODATA]** Asynchronous mode is indicated because O_NONBLOCK was set, but no data is currently available from the transport provider.
- **[TLOOK]** An asynchronous event has occurred on this transport endpoint and requires immediate attention (refer to `t_look()` function).
- **[TOUTSTATE]** The `t_look()` function was issued in the wrong sequence on the transport endpoint referenced by the `fd` parameter.
- **[TSYSERR]** A system error occurred during execution of `t_look()`.

### Related Information

Functions: `fcntl(2)`, `t_getinfo(3)`, `t_look(3)`, `t_open(3)`, `t_snd(3)`
t_rccvconnect

Purpose  Receives the confirmation from a connect request

Library  XTI Library (libtli.a)

Synopsis  
\[
\texttt{#include <xti.h>}
\]
\[
\texttt{int t_rccvconnect(}
\]
\[
\texttt{ int \textit{fd},}
\]
\[
\texttt{ struct t\_call *\textit{call}};}
\]

Parameters  The \texttt{t_rccvconnect()} function can only be called in the T_OUTCON transport provider state. The following table summarizes the relevance of input and output parameters before and after \texttt{t_rccvconnect()} is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{fd}</td>
<td>\texttt{y}</td>
<td>\texttt{n}</td>
</tr>
<tr>
<td>\textit{call-&gt;addr.maxlen}</td>
<td>\texttt{y}</td>
<td>\texttt{n}</td>
</tr>
<tr>
<td>\textit{call-&gt;addr.len}</td>
<td>\texttt{n}</td>
<td>\texttt{y}</td>
</tr>
<tr>
<td>\textit{call-&gt;addr.buf}</td>
<td>\texttt{y}</td>
<td>(\texttt{y})</td>
</tr>
<tr>
<td>\textit{call-&gt;opt.maxlen}</td>
<td>\texttt{y}</td>
<td>\texttt{n}</td>
</tr>
<tr>
<td>\textit{call-&gt;opt.len}</td>
<td>\texttt{n}</td>
<td>\texttt{y}</td>
</tr>
<tr>
<td>\textit{call-&gt;opt.buf}</td>
<td>\texttt{y}</td>
<td>(\texttt{y})</td>
</tr>
<tr>
<td>\textit{call-&gt;udata.maxlen}</td>
<td>\texttt{y}</td>
<td>\texttt{n}</td>
</tr>
<tr>
<td>\textit{call-&gt;udata.len}</td>
<td>\texttt{n}</td>
<td>\texttt{y}</td>
</tr>
<tr>
<td>\textit{call-&gt;udata.buf}</td>
<td>\texttt{y}</td>
<td>(\texttt{o})</td>
</tr>
<tr>
<td>\textit{call-&gt;sequence}</td>
<td>\texttt{n}</td>
<td>\texttt{n}</td>
</tr>
</tbody>
</table>

Notes to Table  
\texttt{y}  This is a meaningful parameter.
\texttt{n}  This is not a meaningful parameter.
\texttt{(o)}  The content of the object pointed to by \texttt{o} is optional.
\texttt{(y)}  The content of the object pointed to by \texttt{y} is meaningful.

\texttt{fd}  Specifies a file descriptor returned by the \texttt{t_open()} function that identifies the local transport endpoint where the connection is to be established.
Points to a type `t_call` structure, used to reserve space for a buffer that stores information associated with the connection at the transport endpoint referenced by the `fd` parameter. When the `call` parameter is set to the null pointer value, no data is returned to the caller. The `t_call` structure has the following members:

```
call
```

- **struct netbuf addr**
  References a buffer for protocol address information returned from the transport endpoint specified by the `fd` parameter. The type `netbuf` structure referenced by this member is defined in the `xti.h` include file and has the following members:

  - **unsigned int maxlen**
    Specifies the maximum byte length of the data buffer.

  - **unsigned int len**
    Specifies the actual byte length of data written to the buffer.

  - **char *buf**
    Points to the buffer location.

- **struct netbuf opt**
  Specifies a buffer for protocol-specific parameters associated with the connection.

- **struct netbuf udata**
  Specifies a buffer for transport user data sent from the destination transport user.

- **int sequence**
  This parameter is not meaningful for the `t_rcvconnect()` function.

The `addr` parameters pointed to by the `call` parameter specify protocol address information associated with the responding transport endpoint. Before this function is called, the `addr.maxlen` parameter must be set to specify the maximum byte length of the protocol-address buffer pointed to by the `addr.buf` parameter, which is used to hold the protocol address of the responding transport endpoint.

On return, the `addr.len` parameter specifies the actual transport endpoint protocol-address byte length and the buffer pointed to by `addr.buf` contains the transport endpoint protocol address.
The `opt` parameters pointed to by the `call` parameter specify optional information associated with the responding transport endpoint. Before this function is called, the `opt.maxlen` parameter must be set to specify the maximum byte length of the options-data buffer pointed to by the `opt.buf` parameter, which is used to hold optional information from the responding transport endpoint when it is provided.

On return, the `opt.len` parameter specifies the actual transport endpoint optional-data byte length and the data buffer pointed to by `opt.buf` contains transport endpoint optional data.

The `udata` parameters pointed to by the `call` parameter specify user information associated with the responding transport endpoint. Before this function is called, the `udata.maxlen` parameter must be set to specify the maximum byte length of the user-data buffer pointed to by the `udata.buf` parameter, which is used to hold remote transport user information from the responding transport endpoint when it is provided.

On return, the `udata.len` parameter specifies the actual transport endpoint user-data byte length and the data buffer pointed to by `udata.buf` contains transport endpoint user data.

### Description

The `t_rcvconnect()` XTI function enables a calling transport user to determine the status of a previously sent connect request at a transport endpoint specified by a file descriptor returned by the `t_open()` function. The `t_rcvconnect()` function is used in conjunction with the `t_connect()` function to establish a connection in asynchronous mode. By default, this function executes in synchronous mode, waiting for the connection to be established before returning control to the caller.

However, when the transport endpoint specified by the `fd` file descriptor has been opened with the O_NONBLOCK flag set in the `t_open()` or `t_fcntl()` functions, the `t_connect()` function executes in asynchronous mode. In asynchronous mode, when no connection confirmation is available, control is immediately returned to the caller.

The `t_rcvconnect()` function uses type `t_call` and `netbuf` structures, which are defined in the `xti.h` include file.

### Return Value

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate the error.
Errors

If the `t_rcvconnect()` function fails, `t_errno` may be set to one of the following values:

- **[TBADF]** The specified file descriptor does not refer to a transport endpoint.
- **[TBUFOVFLW]** The number of bytes allocated for incoming data is not sufficient for storage of that data. The connect information normally returned to the `call` parameter is discarded. The transport provider state is changed to T_DATAXFER.
- **[TNODATA]** Asynchronous mode is indicated because O_NONBLOCK was set, but no connect confirmation is currently available from the transport provider.
- **[TLOOK]** An asynchronous event has occurred on this transport connection and requires immediate attention (refer to the `t_look()` function).
- **[TOUTSTATE]** The function was issued in the wrong sequence on the transport endpoint referenced by the `fd` parameter.
- **[TSYSERR]** A system error occurred during execution of this function.

Related Information

Functions: `t_accept(3)`, `t_alloc(3)`, `t_bind(3)`, `t_connect(3)`, `t_listen(3)`, `t_open(3)`, `t_optmgmt(3)`
t_rcvdis

Purpose  Retrieves disconnect information

Library  XTI Library (libtli.a)

Synopsis  

```
#include <xti.h>

int t_rcvdis(
    int fd, struct t_discon *discon);
```

Parameters

The t_rcvdis() function can be called in the following transport provider states: T_DATAXFER, T_OUTCON, T_OUTREL, T_INREL, and T_INCON (when the number of outstanding connections is greater than 0 (zero)). The following table summarizes the relevance of input and output parameters before and after t_rcvdis() is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>discon-&gt;u.data.maxlen</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>discon-&gt;u.data.len</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>discon-&gt;u.data.buf</td>
<td>y</td>
<td>(o)</td>
</tr>
<tr>
<td>discon-&gt;reason</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>discon-&gt;sequence</td>
<td>n</td>
<td>o</td>
</tr>
</tbody>
</table>

Notes to Table:

- y This is a meaningful parameter.
- n This is not a meaningful parameter.
- o This an optional parameter.
- (o) The content of the object pointed to by y is optional.

fd Specifies a file descriptor returned by the t_open() function that identifies the transport endpoint where a disconnect occurred.
Points to a type `t_discon` structure used to specify user-data parameters that can be returned by the transport user. The `t_discon` structure has the following members:

```
struct netbuf udata
```

Specifies a buffer for transport user data sent to the caller with the disconnect when the `t_rcvdis()` function is processed. The type `netbuf` structure referenced by this member is defined in the `xti.h` include file and has the following members:

```
unsigned int maxlen
```

Specifies the maximum byte length of the data buffer.

```
unsigned int len
```

Specifies the actual byte length of data written to the buffer.

```
char *buf
```

Points to the buffer location.

```
int reason
```

Specifies the reason the disconnect occurred.

```
int sequence
```

Specifies the sequence number identifying an outstanding connection indication that has been disconnected.

On return, the `discon->udata` buffer contains information associated with the disconnect. Before the `t_rcvdis()` function is called, `udata.maxlen` must be set to specify the maximum byte length of the user-data buffer.

The `discon->reason` parameter specifies the reason for the disconnect using a protocol-dependent reason code. When protocol independence is a concern, this information should not be examined.

When this function is called after issuing one or more `t_listen()` functions, and there is more than one outstanding transport endpoint connection (refer to the `t_listen()` function), the `discon->sequence` parameter is used to specify the the outstanding connection indication with which the disconnect is associated.
When a transport user is not concerned with incoming remote transport user data, with a reason for a disconnect, or with the sequence number of the transport endpoint where the disconnect took place, the `discon` parameter may be specified as a null pointer. When `discon` is specified as a null pointer, no data is returned to the caller.

When a transport user knows there is more than one active connection indication (refer to the `t_look()` function), and this function is called with the `discon` parameter set to the null pointer value, there is no way to identify the connection where the disconnect occurred.

**Description**

The `t_rcvdis()` XTI connection-oriented function is used to identify the cause of a disconnect at a transport endpoint specified by a file descriptor returned by the `t_open()` function, and to retrieve any user data queued with the disconnect.

**Return Value**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate the error.

**Errors**

If the `t_rcvdis()` function fails, `t_errno` is set to one of the following values:

- `[TBADF]` File descriptor `fd` does not refer to a valid transport endpoint.
- `[TNODIS]` No disconnect indication currently exists on the transport endpoint specified by the `fd` parameter.
- `[TBUFOVFLW]` The number of bytes allocated for incoming data is not sufficient to store the data. When `fd` specifies a passive transport endpoint (the number of outstanding connection indications is greater than 1), the transport endpoint remains in state `T_INCON`; otherwise, the transport endpoint state becomes `T_IDLE`.

---

1–840
Functions

t_rcvdis(3)

[TSYSERR] A system error occurred during execution of this function.

[TOUTSTATE] The t_rcvdis() function was issued in the wrong sequence on the transport endpoint referenced by the $fd$ parameter.

Related Information

Functions: t_alloc(3), t_connect(3), t_listen(3), t_open(3), t_snddis(3)
t_rcvrel

**Purpose**
Acknowledges receipt of an orderly release indication

**Synopsis**
```
#include <xti.h>

int t_rcvrel(
    int fd):
```

**Parameters**
The `t_rcvrel()` function can be called in the T_DATAXFER and T_OUTREL transport provider states only. The following table summarizes the relevance of input parameter data before and after `t_rcvrel()` is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>y</td>
<td>n</td>
</tr>
</tbody>
</table>

**Notes to Table:**
- `y` This is a meaningful parameter.
- `n` This is not a meaningful parameter.

- `fd` Specifies a file descriptor returned by the `t_open()` function that identifies a local transport endpoint that has been released.

**Description**
The `t_rcvrel()` XTI function is used in connection-oriented mode to acknowledge receipt of an orderly release indication at a transport endpoint. The released endpoint is specified by a file descriptor previously returned by the `t_open()` function.

After receipt of this orderly release indication, at the transport endpoint specified by the file descriptor, a transport user should not try to receive additional data from that transport endpoint. Any attempt to receive more data from a released transport endpoint blocks continuously. However, a transport user may continue to send data across the connection until a release is sent by a transport user who invokes a `t_sndrel()` function call.
The \texttt{t_rcvrel()} function should not be used unless the \texttt{servtype} type-of-service returned by the \texttt{t_open()} or \texttt{t_getinfo()} functions is \texttt{T_COTS_ORD} (supports connection-mode service with the optional orderly release facility).

Return Value

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and \texttt{t_errno} is set to indicate the error.

Errors

If the \texttt{t_rcvrel()} function fails, \texttt{t_errno} may be set to one of the following values:

- **[TBADF]** File descriptor \texttt{fd} does not refer to a valid transport endpoint.
- **[TNOREL]** No orderly release indication currently exists at the transport endpoint specified by the \texttt{fd} parameter.
- **[TLOOK]** An asynchronous event has occurred on the transport endpoint specified by the \texttt{fd} parameter and requires immediate attention.
- **[TSYSERR]** A system error occurred during execution of this function.
- **[TOUTSTATE]** The \texttt{t_rcvrel()} function was issued in the wrong sequence at the transport endpoint referenced by the \texttt{fd} parameter.

Related Information

Functions: \texttt{t_getinfo(3)}, \texttt{t_open(3)}, \texttt{t_sndrel(3)}
t_rcvudata

Purpose
Receives a data unit

Library
XTI Library (libtli.a)

Synopsis
#include <xti.h>
int t_rcvudata(
    int fd,
    struct t_unitdata *unitdata,
    int *flags);

Parameters
The t_rcvudata() function can only be called in the T_IDLE transport provider state. The following table summarizes the relevance of input and output parameter data before and after t_rcvudata() is called:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;addr.maxlen</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;addr.len</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>unitdata-&gt;addr.buf</td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td>unitdata-&gt;opt.maxlen</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;opt.len</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>unitdata-&gt;opt.buf</td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td>unitdata-&gt;udata.maxlen</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;udata.len</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>unitdata-&gt;udata.buf</td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td>flags</td>
<td></td>
<td>y</td>
</tr>
</tbody>
</table>

Notes to Table:
- y This is a meaningful parameter.
- n This is not a meaningful parameter.
- (y) The content of the object pointed to by y is meaningful.

fd Specifies a file descriptor returned by the t_open() function that identifies the transport endpoint.
Points to a type `t_unitdata` structure used to specify information required by the transport provider user to receive a data unit through the transport endpoint specified by the `fd` parameter. The `t_unitdata` structure has the following members:

**struct netbuf addr**

References a buffer for protocol address information required from the transport endpoint specified by the `fd` parameter. The type `netbuf` structure referenced by this member is defined in the `xti.h` include file and has the following members:

- `unsigned int maxlen`
  
  Specifies the maximum byte length of the data buffer.

- `unsigned int len`
  
  Specifies the actual byte length of the data written to the buffer.

- `char *buf`
  
  Points to the buffer location.

**struct netbuf opt**

Specifies a buffer for protocol-specific parameters associated with the data unit.

**struct netbuf udata**

Specifies parameters for any user data unit that may be returned to the caller.

Before the `t_rcvudata()` function is called, the `unitdata->addr.maxlen`, `unitdata->opt.maxlen`, and `unitdata->udata.maxlen` parameters must be set to specify the maximum byte length of the protocol address buffer, the protocol options buffer, and the user data buffer, respectively.

**flags**

Points to a flag integer that indicates that the complete data unit was not received. Corresponding values and symbolic names for flags are defined in the `xti.h` include file (see the `t_optmgmt()` and `t_rcv()` functions). The flag specified by this function is:

T_MORE.

When the data buffer specified by the `unitdata->udata.buf` parameter is not large enough to hold the current user data unit, the buffer is filled and this bit is set to indicate that another `t_rcvudata()` function should be called to retrieve the rest of the data unit.
t_rcvudata(3)

The set state of this bit notifies the local transport user that the received data unit is a fragment and that another data unit is available. When this bit is set on return of this function, another data unit must also be fetched with another t_rcvudata() call. Each time this flag is set on return, another t_rcvudata() call must immediately be made to receive additional current data units. When the final data unit is received, this flag bit has a value of 0 (zero) on return.

Subsequent calls to the t_rcvudata() function return 0 (zero) as the length of the address specified by the unitdata->addr.len and unitdata->opt.len parameters until the full data unit has been received.

Description

The t_rcvudata() function is an XTI connectionless service function that is used to receive a data unit from a remote transport provider user. By default, t_rcvudata() executes in the synchronous operating mode. The t_rcvudata() function waits for data to arrive at the transport endpoint specified by fd before returning control to the transport user who called this function.

However, when the transport endpoint, specified by the fd parameter, has been previously opened with the O_NONBLOCK flag set in the t_open() or fcntl() function, the t_rcvudata() function executes in asynchronous mode. In asynchronous mode, when a data unit is unavailable, control is immediately returned to the caller.

Return Value

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate the error.

Errors

If the t_rcvudata() function fails, t_errno may be set to one of the following values:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TNODATA] Asynchronous mode is indicated because O_NONBLOCK was set, but no data is currently available from the transport provider.

[TBUFOVFLW] The number of bytes allocated for the incoming protocol address or protocol options is not sufficient to store the information. The unit data information normally returned to the unitdata parameter is discarded.
Functions

t_rcvudata(3)

[TLOOK] An asynchronous event that requires immediate attention has occurred at the transport endpoint specified by the fd parameter.

[TOUTSTATE] The t_rcvudata() function was issued in the wrong sequence at the transport endpoint referenced by the fd parameter.

[TSYSERR] A system error occurred during execution of this function.

Related Information

Functions: fcntl(2), t_alloc(3), t_open(3), t_optmgmt(3), t_rcv(3), t_rcvuderr(3), t_sndudata(3)
t_rcvuderr

Purpose  Receives a unit data error indication

Library  XTI Library (libtli.a)

Synopsis  #include <xti.h>
          int t_rcvuderr(intfd, 
                          struct t_uderr *uderr);

Parameters  The t_sndudata() function can only be called in the T_IDLE transport provider state. The following table summarizes the relevance of input and output parameters before and after t_rcvuderr() is called:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>uderr-&gt;addr.maxlen</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>uderr-&gt;addr.len</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>uderr-&gt;addr.buf</td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td>uderr-&gt;opt.maxlen</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>uderr-&gt;opt.len</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>uderr-&gt;opt.buf</td>
<td>y</td>
<td>(y)</td>
</tr>
<tr>
<td>uderr-&gt;error</td>
<td>n</td>
<td>y</td>
</tr>
</tbody>
</table>

Notes to Table:

- y This is a meaningful parameter.
- n This is not a meaningful parameter.
- (y) The content of the object pointed to by y is meaningful.

- fd Specifies a file descriptor returned by the t_open() function that identifies the local transport endpoint on which the error occurred.
uderr

Points to a type t_uderr structure used to specify the protocol address, protocol options, and the nature of the error associated with the data unit sent through the transport endpoint specified by the fd parameter. The t_uderr structure has the following members:

struct netbuf addr

References a buffer for protocol address information associated with the erroneous data unit sent from the transport endpoint specified by the fd parameter. The type netbuf structure referenced by this member is defined in the xti.h include file and has the following members:

unsigned int maxlen

Specifies the maximum byte length of the data buffer.

unsigned int len

Specifies the actual byte length of data written to the buffer.

char *buf

Points to the buffer location.

struct netbuf opt

Specifies a buffer for protocol-specific parameters associated with the previously sent erroneous data unit.

long error

Specifies a protocol-specific error code associated with the previously sent erroneous data unit.

Before the t_rcvuderr() function is called the uderr->addr.maxlen and uderr->opt.maxlen parameters must be set to specify the maximum byte length of the protocol address buffer and the protocol options buffer, respectively, of the calling transport user.

When a transport user does not wish to identify the source of the previously sent data unit error, the uderr parameter may be specified as a null pointer. When this parameter is expressed as a null pointer, the data unit error indication is cleared, but no information is returned to buffers pointed to by this parameter.

Description

The t_rcvuderr() function is an XTI connectionless service function that is used to retrieve information about an error indication returned when a data unit was previously sent with a t_sndudata() call.
The `t_rcvuderr()` function should be called only after a [T_LOOK] error is returned in response to a `t_sndudata()` call. When `t_rcvuderr()` successfully executes, the error will be cleared. The `t_rcvuderr()` function uses type `t_uderr` and `netbuf` structures, which are defined in the `xti.h` include file.

**Return Value**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate the error.

**Errors**

If the `t_rcvuderr()` function fails, `t_errno` may be set to one of the following values:

- **[TBADF]** File descriptor `fd` does not refer to a valid transport endpoint.
- **[TNOUDERR]** No unit data error indication currently exists at the transport endpoint specified by the `fd` parameter.
- **[TBUFOVFLW]** The number of bytes allocated for the incoming protocol address or options information is not sufficient to store that information. Unit data error information was not returned to buffers pointed to by the `uderr` parameter.
- **[TSYSERR]** A system error occurred during execution of this function.

**Related Information**

Functions: `t_look(3), t_rcvudata(3), t_sndudata(3)`
t_snd

Purpose
Sends normal data or expedited data over a connection

Library
XTI Library (libtli.a)

Synopsis
#include <xti.h>

int t_snd(
int fd,
char *buf,
unsigned nbytes,
int flags);

Parameters
The t_snd() function can only be called in the T_DATAXFER and T_INREL transport provider states. The following table summarizes the relevance of input and output parameters before and after t_snd() is called:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>buf</td>
<td>y(y)</td>
<td>n</td>
</tr>
<tr>
<td>nbytes</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>flags</td>
<td>y</td>
<td>n</td>
</tr>
</tbody>
</table>

Notes to Table:
y This is a meaningful parameter.
n This is not a meaningful parameter.
y(y) The content of the object pointed to by y is meaningful.

fd Specifies a file descriptor returned by the t_open() function that identifies the local transport endpoint where an active connection exists.

buf Points to the data buffer from which data is to be sent.

nbytes Specifies the length in bytes of the send data buffer contents pointed to by the buf parameter.
**flags**
Points to an integer whose bits specify certain optional information. Corresponding values and symbolic names for these flag bits are defined in the `xti.h` include file. Flags specified by this function are:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_MORE</td>
<td>When set, this bit notifies the transport provider that sent data is a fragment of a Transport Service Data Unit (TSDU) or Expedited Transport Service Data Unit (ETSDU), and that more data will be sent on the same TSDU or ETSDU via the <code>t_snd()</code> function. The rest of the TSDU or ETSDU can be sent through further <code>t_snd()</code> function calls. Each time the T_MORE flag is set, another <code>t_snd()</code> call follows so that additional parts of the TSDUs or ETSDUs can be sent. When the final piece is sent, this flag bit is set to a value of 0 (zero). When the transport provider does not support TSDU or ETSDU data exchanges (refer to the <code>t_open()</code> and <code>t_getinfo()</code> functions) the state of this flag bit is meaningless.</td>
</tr>
<tr>
<td>T_EXPEDITED</td>
<td>When set, this bit notifies the transport provider that expedited data is sent. When the value of ETSDU data exceeds the value specified by <code>nbytes</code> parameter, this flag bit and the T_MORE flag bit should be set prior to the initial <code>t_snd()</code> call. Subsequent <code>t_snd()</code> calls used to send pieces of ETSDU must have both these flag bits set. When the final ETSDU is sent, the T_MORE flag bit is set to a value of 0 (zero).</td>
</tr>
</tbody>
</table>

**Description**
The `t_snd()` function is an XTI connection-oriented service function that is used to send normal or expedited data. The transport endpoint through which normal Transport Service Data Unit (TSDU) data or special Expedited TSDU (ETSDU) data is sent is specified by a file descriptor previously returned by the `t_open()` function.
The size of each TSDU or ETSDU must not exceed the size limits specified by info->tsdu or info->etsdu, respectively, returned by the t_open() or t_getinfo() functions. Failure to comply with specified size constraints results in return of a [TYSYSERR] protocol error. By default, the t_snd() function executes in the synchronous operating mode. In the synchronous operating mode t_snd() waits for data to be accepted by the transport provider, before returning control to the calling transport user.

When the transport endpoint specified by the file descriptor has been opened with the O_NONBLOCK flag set in the t_open() or fcntl() function, the t_snd() function executes in asynchronous mode. When data cannot be immediately accepted because flow control restrictions apply, control is immediately returned to the caller.

When the t_snd() function executes successfully, the number of bytes accepted by the transport provider is returned. It is possible that only part of the data may be accepted by a transport provider. When only partial data is accepted, the returned value is less than the number of bytes sent. If the nbytes parameter is specified as 0 (zero), and the underlying transport service does not support the sending of 0 octets, t_errno is set to [TBADDATA] and -1 is returned.

Notes

In asynchronous mode, when the number of bytes accepted by the transport provider is less than the number of bytes sent, the transport provider may be blocked because of flow-control restrictions.

Return Value

Upon successful completion, the t_snd() function returns the number of bytes of data accepted by the transport provider. Otherwise, -1 is returned and t_errno is set to indicate the error.

Errors

If the t_snd() function fails, t_errno may be set to one of the following values:

[TBADF] File descriptor fd does not refer to a valid transport endpoint.

[TBADDATA] Illegal amount of data. Zero octets is not supported.

[TBADFLAG] An invalid flags value was specified.
Asynchronous mode is indicated because O_NONBLOCK was set, but no data can currently be accepted by the transport provider because of flow-control restrictions.

An asynchronous event occurred on this transport endpoint.

The \texttt{t SND()} function was issued in the wrong sequence on the transport endpoint referenced by the \texttt{fd} parameter.

A system error occurred during execution of the \texttt{t SND()} function. A protocol error may not cause \texttt{t SND()} to fail until a subsequent access of the transport endpoint is made.

Related Information

Functions: \texttt{fcntl(2)}, \texttt{t_getinfo(3)}, \texttt{t.look(3)}, \texttt{t_open(3)}, \texttt{t_optmgmt(3)}, \texttt{t_rcv(3)}
t_snndis

**Purpose** Sends user-initiated disconnect request

**Library**
XTI Library (libtli.a)

**Synopsis**

```c
#include <xti.h>
int t_snndis(
    int fd,
    struct t_call *call);
```

**Parameters**
The `t_snndis()` function can be called in the following transport provider states: T_DATAXFER, T_OUTCON, T_OUTREL, T_INREL, and T_INCON (when the number of outstanding connections is greater than zero). The following table summarizes the relevance of input and output parameters before and after `t_snndis()` is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>call-&gt;addr.maxlen</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>call-&gt;addr.len</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>call-&gt;addr.buf</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>call-&gt;opt.maxlen</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>call-&gt;opt.len</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>call-&gt;opt.buf</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>call-&gt;udata.maxlen</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>call-&gt;udata.len</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>call-&gt;udata.buf</td>
<td>o(o)</td>
<td>n</td>
</tr>
<tr>
<td>call-&gt;sequence</td>
<td>o</td>
<td>n</td>
</tr>
</tbody>
</table>

**Notes to Table:**
- **y** This is a meaningful parameter.
- **n** This is not a meaningful parameter.
- **o** This an optional parameter.
- **(o)** The content of the object pointed to by y is optional.

`fd` Specifies a file descriptor returned by the `t_open()` function that identifies the transport endpoint at which the disconnect is wanted.
Points to a type `t_call` structure used to specify information associated with the disconnect at the transport endpoint specified by file descriptor `fd`. When the `call` parameter is set to the null pointer value, no data is sent to the remote transport provider user. The `t_call` structure has the following two members:

```c
struct netbuf udata
```

Specifies a buffer for user data that may be optionally sent to the remote transport user. The type `netbuf` structure referenced by this member is defined in the `xti.h` include file. This structure, which is used to explicitly define buffer parameters, has the following members:

- `unsigned int maxlen` 
  Specifies the maximum byte length of the data buffer.

- `unsigned int len` 
  Specifies the actual byte length of data written to the buffer.

- `char *buf` 
  Points to the buffer location.

```c
int sequence
```

Specifies the identity of the connection for which this disconnect request is intended and has meaning only when the transport provider is in the T_INCON state and is rejecting an incoming rejection request.

The `udata` parameters pointed to by the `call` parameter need only be used when data is sent with a disconnect request.

When data is sent with the disconnect request, the size of the data written to the buffer pointed to by `call->udata.buf` must not exceed the limits specified by `info->discon`, which is returned by the `t_open()` or `t_getinfo()` functions. Failure to comply with the specified size constraints may result in return of a `[T_SYSERR]` protocol error.

The `sequence` parameter is meaningful only if the transport user is rejecting an incoming connection request and needs to identify which incoming connection request to reject.
Description

The `t_snddis()` XTI connection-oriented function is used to initiate an abortive disconnect at an established transport endpoint. The transport endpoint is specified by a file descriptor returned by the `t_open()` function. The `t_snddis()` function uses type `t_call` and `netbuf` structures, which are defined in the `xti.h` include file.

Return Value

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate the error.

Errors

If the `t_snddis()` function fails, `t_errno` may be set to one of the following values:

- [TBADF] File descriptor `fd` does not refer to a valid transport endpoint.
- [TOUTSTATE] This function was issued in the wrong sequence at the transport endpoint referenced by the `fd` parameter.
- [TBADDATA] The amount of user data specified was not within the bounds allowed by the transport provider. Some outbound data queued for this endpoint may be lost.
- [TBADSEQ] An invalid sequence number was specified, or a null value was used for the `call` parameter when the connect request was rejected. Some outbound data queued for this endpoint may be lost.
- [TSYSERR] A system error occurred during execution of this function.

Related Information

Functions: `t_connect(3)`, `t_getinfo(3)`, `t_listen(3)`, `t_open(3)`
t_sndrel

**Purpose**
Initiates an endpoint connect orderly release

**Library**
XTI Library (libtli.a)

**Synopsis**
```c
#include <xti.h>

int t_sndrel(int fd);
```

**Parameters**
The `t_sndrel()` function can be called in the T_DATAXFER and T_INREL transport provider states only. The following table summarizes the relevance of input parameter data before and after `t_sndrel()` is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td><code>y</code></td>
<td><code>n</code></td>
</tr>
</tbody>
</table>

**Notes to Table**
- `y`: This is a meaningful parameter.
- `n`: This is not a meaningful parameter.

**Description**
The `t_sndrel()` XTI function is used in connection-oriented mode to initiate an orderly release at a transport endpoint specified by a file descriptor previously returned by the `t_open()` function.

After this orderly release is indicated, the transport user should not try to send more data through that transport endpoint; an attempt to send more data to a released transport endpoint may block continuously. However, a transport user may continue to receive data over the connection until an orderly release indication is received.
The **t_sndrel**() function should not be used unless the `servtype` type-of-service returned by the **t_open**() or **t_getinfo**() functions is **T_COTS_ORD** (supports connection-mode service with the optional orderly release facility).

**Return Value**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and **t_errno** is set to indicate an error.

**Errors**

If the **t_sndrel**() function fails, **t_errno** may be set to one of the following values:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBADF</td>
<td>File descriptor <code>fd</code> does not refer to a valid transport endpoint.</td>
</tr>
<tr>
<td>TFLOW</td>
<td>Asynchronous mode is indicated because O_NONBLOCK was set, but the transport provider cannot accept a release because of flow-control restrictions.</td>
</tr>
<tr>
<td>TLOOK</td>
<td>An asynchronous event has occurred on this transport endpoint and requires immediate attention.</td>
</tr>
<tr>
<td>TOUTSTATE</td>
<td>The <strong>t_sndrel</strong>() function was issued in the wrong sequence at the transport endpoint specified by the <code>fd</code> parameter.</td>
</tr>
<tr>
<td>TSYSERR</td>
<td>A system error occurred during execution of this function.</td>
</tr>
</tbody>
</table>

**Related Information**

Functions: **t_getinfo**(3), **t_open**(3), **t_rcvrel**(3)
t_sndudata

Purpose
Sends a data unit

Library
XTI Library (libtli.a)

Synopsis
#include <xti.h>
int t_sndudata(
  int fd,
  struct t_unitdata *unitdata);

Parameters
The t_sndudata() function can only be called in the T_IDLE transport provider state. The following table summarizes the relevance of input and output parameters before and after t_sndudata() is called:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;addr.maxlen</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;addr.len</td>
<td>y(y)</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;addr.buf</td>
<td>o(o)</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;opt.maxlen</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;opt.len</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;opt.buf</td>
<td>o(o)</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;udata.maxlen</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;udata.len</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>unitdata-&gt;udata.buf</td>
<td>y(y)</td>
<td>n</td>
</tr>
</tbody>
</table>

Notes to Table:
y This is a meaningful parameter.
y(y) The content of the object pointed to by y is meaningful.
o This is a meaningful but optional parameter.
o(o) The content of the object pointed to by o is meaningful.
n This is not a meaningful parameter.

fd
Specifies a file descriptor returned by the t_open() function that identifies the transport endpoint through which data is sent.
Functions

\texttt{t_sndudata(3)}

\texttt{unitdata} Points to a type \texttt{t_unitdata} structure used to specify a data unit being sent through the transport endpoint specified by the \texttt{fd} parameter. The \texttt{t_unitdata} structure has the following members:

\begin{verbatim}
struct netbuf \texttt{addr}
    References a buffer for protocol address information of the remote transport user. The type \texttt{netbuf} structure referenced by this member is defined in the \texttt{xti.h} include file and has the following members:

    \begin{verbatim}
    unsigned int \texttt{maxlen}
        Specifies a maximum byte length of the data buffer.

    unsigned int \texttt{len}
        Specifies the actual byte length of the data written to the buffer.

    char *\texttt{buf}
        Points to the buffer location.
    \end{verbatim}

\end{verbatim}

\begin{verbatim}
struct netbuf \texttt{opt}
    Specifies protocol-specific optional parameters.

struct netbuf \texttt{udata}
    Specifies the user data unit that is being sent to the remote transport user.
\end{verbatim}

The \texttt{unitdata->addr.maxlen}, \texttt{unitdata->opt.maxlen}, and \texttt{unitdata->udata.maxlen} parameters are not meaningful with the \texttt{t_sndudata()} function.

When optional data is not provided, the \texttt{opt.len} parameter should be set to the null value.

If the \texttt{udata.len} parameter is specified as 0 (zero), and the underlying transport service does not support the sending of 0 (zero) octets, \texttt{t_errno} is set to \texttt{[TBADDATA]} and -1 is returned.

\textbf{Description}

The \texttt{t_sndudata()} function is an XTI connectionless service function that is used to send a data unit to a remote transport user. By default, \texttt{t_sndudata()} executes in the synchronous operating mode. The \texttt{t sndudata()} function waits for the transport provider to accept the data before returning control to the calling transport user.
When the transport endpoint specified by the \textit{fd} parameter has been previously opened with the O\textunderscore NONBLOCK flag set in the \texttt{t\_open()} or \texttt{fcntl()} functions, the \texttt{t\_sndudata()} function executes in asynchronous mode. In asynchronous mode, when a data unit is not accepted control is immediately returned to the caller. The \texttt{t\_look()} function can be used to determine when flow control restrictions have been lifted.

**Return Value**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and \texttt{t\_errno} is set to indicate the error.

**Errors**

If the \texttt{t\_sndudata()} function fails, \texttt{t\_errno} may be set to one of the following values:

[TBADDATA] Illegal amount of data. Zero octets is not supported.

[TBADF] File descriptor \textit{fd} is not a valid transport endpoint.

[TFLOW] Asynchronous mode is indicated because O\textunderscore NONBLOCK was set, but the transport provider cannot accept the data because of flow-control restrictions.

[TLOOK] An asynchronous event has occurred on this transport endpoint and requires immediate attention.

[TOUTSTATE] The \texttt{t\_sendudata()} function was issued in the wrong sequence on the transport endpoint referenced by the \textit{fd} parameter.

[TSYSERR] A system error occurred during execution of this function. A protocol error may not cause the \texttt{t\_sndudata()} function to fail until a subsequent call is made to access the transport endpoint specified by the \textit{fd} parameter.

**Related Information**

Functions: \texttt{fcntl(2)}, \texttt{t\_alloc(3)}, \texttt{t\_open(3)}, \texttt{t\_rcvuderr(3)}, \texttt{t\_sndudata(3)}
t_sync

Purpose
Synchronizes transport library

Library
XTI Library (libtli.a)

Synopsis
#include <xti.h>

int t_sync(
    int fd);

Parameters
The t_sync() function can be called in any transport provider state except TUNINIT. The following table summarizes the relevance of input parameter data before and after t_sync() is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>y</td>
<td>n</td>
</tr>
</tbody>
</table>

Notes to Table

y This is a meaningful parameter.

n This is not a meaningful parameter.

fd Specifies a file descriptor returned by the t_open() function that identifies an active, uninitialized local transport endpoint.

Description

The t_sync() XTI utility service function is used to synchronize data structures managed by the transport library with information from the underlying transport provider.

The t_sync() function is used to convert an uninitialized file descriptor, previously returned by the open() or dup() functions, or returned as the result of fork() or exec() functions, to an initialized transport endpoint. When the file descriptor references a valid transport endpoint, necessary library data structures are allocated and updated.
The \texttt{t\_sync()} function also permits two cooperating processes to synchronize their interaction with a transport provider. When a process forks, for example, and an \texttt{exec()} function is issued, the child (new) process must call the \texttt{t\_sync()} function to build a private library data structure associated with the transport endpoint referenced by the \texttt{fd} parameter and to synchronize the library data structure with relevant transport provider information.

A transport provider treats multiple users of a transport endpoint as the same user. When more than one process is using the same transport endpoint, each should coordinate its activities so that operation does not conflict with the transport provider state at the transport endpoint specified by \texttt{fd}.

The \texttt{t\_sync()} function returns the current state of the transport provider (refer to the \texttt{t\_getstate()} function). Return of the current state of the transport provider permits the calling transport user to verify the transport provider state before issuing the next function call. This coordination is only valid among cooperating processes; it is possible that a process or an incoming event can change the transport provider state at the reference transport endpoint after \texttt{t\_sync()} is called.

When the transport provider at the transport endpoint referenced by the \texttt{fd} parameter is undergoing a change of state and the \texttt{t\_sync()} function is called, the \texttt{t\_sync()} process fails and returns a [TSTATECHNG] error.

\section*{Return Value}

Upon successful completion, the state of the transport provider at the transport endpoint specified by the \texttt{fd} parameter is returned. Otherwise, a value of -1 is returned and \texttt{t\_errno} is set to indicate the error. The returned state is one of the following:

- \texttt{[T\_UNBND]} Transport endpoint not bound to an address.
- \texttt{[T\_IDLE]} Transport endpoint is idle.
- \texttt{[T\_OUTCON]} Outgoing connection pending.
- \texttt{[T\_INCON]} Incoming connection pending.
- \texttt{[T\_DATAXFER]} Data transfer.
- \texttt{[T\_OUTREL]} Outgoing orderly release (waiting for an orderly release indication).
- \texttt{[T\_INREL]} Incoming orderly release (waiting for an orderly release request).
Errors

If the `t_sync()` function fails, `t_errno` may be set to one of the following values:

[TBADF] File descriptor `fd` is not a valid transport endpoint. This error may be returned when the `fd` parameter has been previously closed or an erroneous file-descriptor value may have been passed to the call.

[TSTATECHNG] The transport endpoint is undergoing a state change.

[TSYSERR] A system error occurred during execution of this function.

Related Information

Functions: `exec(2), fcntl(2), fork(2), open(2), t_getstate(3)`
t_unbind

**Purpose**  
Disables a transport endpoint

**Library**  
XTI Library (libtli.a)

**Synopsis**  
```c  
#include <xti.h>  
int t_unbind(  
    int fd) ;  
```

**Parameters**  
The `t_bind()` function can only be called in the T_IDLE transport provider state. The following table summarizes the relevance of input parameter data before and after `t_bind()` is called:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Before Call</th>
<th>After Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td><code>y</code></td>
<td><code>n</code></td>
</tr>
</tbody>
</table>

**Notes to Table:**
- `y` This is a meaningful parameter.
- `n` This is not a meaningful parameter.

`fd` Specifies a file descriptor returned by the `t_open()` function that identifies an active, previously bound local transport endpoint.

**Description**  
The `t_unbind()` XTI service function is used in connection-oriented and connectionless modes to disable the transport endpoint, specified by the file descriptor that was previously bound by a `t_bind()` call. When `t_bind()` completes, no further data destined for this transport endpoint or events are accepted by the transport provider.
Return Value
Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and \texttt{t_errno} is set to indicate the error.

Errors
If the \texttt{t_unbind()} function fails, \texttt{t_errno} is set to one of the following values:

- \([\text{TBA DF}]\) File descriptor \(fd\) is not a valid transport endpoint.
- \([\text{TOUTSTATE}]\) This function was issued in the wrong sequence.
- \([\text{TLOOK}]\) An asynchronous event occurred at the transport endpoint specified by the \(fd\) parameter.
- \([\text{TSYSERR}]\) A system error occurred during execution of this function.

Related Information
Functions: \texttt{t_bind(3)}
tcdrain

Purpose  Waits for output to complete

Library

Standard C Library (libc.a)

Synopsis

#include <termios.h>

int tcdrain(
        int filedes);

Parameters

filedes  Specifies an open file descriptor.

Description

The tcdrain() function waits until all output written to the object referred to by the
filedes parameter has been transmitted.

A process group is sent a SIGTTOU signal if the tcdrain() function is called from
one of its member processes. If the calling process is blocking or ignoring
SIGTTOU signals, the process is allowed to perform the operation and no signal is
sent.

The tcdrain() function, which suspends the calling process until the request is
completed, is redefined so that only the calling thread is suspended.

Notes

AES Support Level:  Full use

Example

To wait until all output has been transmitted, enter:

rc = tcdrain(stdout);
Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the `tcdrain()` function fails, errno may be set to one of the following values:

- [EBADF] The `filedes` parameter does not specify a valid file descriptor.
- [EINTR] A signal interrupted the `tcdrain()` function.
- [ENOTTY] The file associated with the `filedes` parameter is not a terminal.

Related Information

Functions: `tcflow(3)`, `tcflush(3)`, `tcsendbreak(3)`
tcflow

Purpose
Performs flow control functions

Library
Standard C Library (libc.a)

Synopsis
#include <termios.h>

int tcflow(
    int filedes,
    int action);

Parameters
filedes Specifies an open file descriptor.
action Specifies one of the following:

TCOFF
Suspend output.

TCOON
Restart suspended output.

TCIOFF
Transmit a STOP character, which is intended to cause the
terminal device to stop transmitting data to the system.

TCION
Transmit a START character, which is intended to cause the
terminal device to start transmitting data to the system.

Description
The tcflow() function suspends transmission or reception of data on the object
referred to by the filedes parameter, depending on the value of the action
parameter.

A process group is sent a SIGTTOU signal if the tcflow() function is called from
one of its member processes. If the calling process is blocking or ignoring
SIGTTOU signals, the process is allowed to perform the operation and no signal is
sent.
Notes

AES Support Level: Full use

Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Errors

If the tcflow() function fails, errno may be set to one of the following values:

- [EBADF] The filedes parameter does not specify a valid file descriptor.
- [EINVAL] The action parameter is not a supported value.
- [ENOTTY] The file associated with the filedes parameter is not a terminal.

Related Information

Functions: tcdrain(3), tcflush(3), tcsendbreak(3)

Files: termios(4)
tcflush

Purpose
Flushes nontransmitted output data or nonread input data

Library
Standard C Library (libc.a)

Synopsis
#include <termios.h>
int tcflush(
    int filedes,
    int queue_selector);

Parameters

Parameters

filedes
Specifies an open file descriptor associated with a terminal.

queue_selector
Specifies one of the following:

TCIFLUSH
Flush data received but not read.

TCOFLUSH
Flush data written but not transmitted.

TCIOFLUSH
Flush both data received but not read and data written but not transmitted.

Description
The tcflush() function discards any data written to the object referred to by the
filedes parameter, or data received but not read by the object referred to by filedes,
depending on the value of the queue_selector parameter.

A process group is sent a SIGTTOU signal if the tcflush() function is called from
one of its member processes. If the calling process is blocking or ignoring
SIGTTOU signals, the process is allowed to perform the operation and no signal is
sent.

Notes

AES Support Level: Full use
Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

Errors

If the `tcflush()` function fails, **errno** may be set to one of the following values:

- [EBADF]: The `filedes` parameter does not specify a valid file descriptor.
- [EINVAL]: The `queue_selector` parameter does not specify a proper value.
- [ENOTTY]: The file associated with the `filedes` parameter is not a terminal.

Related Information

Functions: `tcdrain(3)`, `tcflow(3)`, `tcsendbreak(3)`
Files: `termios(4)`
tcgetattr

**Purpose**
Gets the parameters associated with the terminal

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <termios.h>

int tcgetattr (
    int file_des,
    struct termios *termios_p);
```

**Parameters**
- `file_des` Specifies an open file descriptor associated with a terminal.
- `termios_p` Points to a `termios` structure.

**Description**
The `tcgetattr()` function gets the parameters associated with the object referenced by the `file_des` parameter and stores them in the `termios` structure referenced by the `termios_p` parameter.

If the device does not support split baud rates, the input baud rate stored in the `termios` structure will be 0 (zero).

The `tcgetattr()` function may be called from any process.

**Notes**

AES Support Level: Full use

**Return Values**
Upon successful completion, 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.
Errors

If the `tcgetattr()` function fails, `errno` may be set to one of the following values:

- `[EBADF]` The `file_des` parameter is not a valid file descriptor.
- `[ENOTTY]` The file associated with the `file_des` parameter is not a terminal.

Related Information

Functions: `tcsetattr(3)`
Files: `termios(4)`
tcgetpgrp

Purpose
Gets foreground process group ID

Library
Standard C Library (libc.a)

Synopsis
```
#include <sys/types.h>

pid_t tcgetpgrp(
    int file_des);
```

Parameters
```
file_des
```
Indicates the open file descriptor for the terminal special file.

Description
The `tcgetpgrp()` function returns the value of the process group ID of the foreground process group associated with the terminal. The function can be called from a background process; however, the information may be subsequently changed by the foreground process.

Notes
AES Support Level: Full use

Return Values
Upon successful completion, the process group ID of the foreground process is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.
Errors

If the tcgetpgrp( ) function fails, errno may be set to one of the following values:

[EBADF] The file_des parameter is not a valid file descriptor.

[ENOTTY] The calling process does not have a controlling terminal or the file is not the controlling terminal.

Related Information

Functions: setpgid(2), setsid(2), tcsetpgrp(3)
tcsendbreak

**Purpose**
Sends a break on an asynchronous serial data line

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <termios.h>

int tcsendbreak(
    int filedes,
    int duration);
```

**Parameters**
- `filedes` Specifies an open file descriptor.
- `duration` Specifies the number of milliseconds that zero-valued bits are transmitted. If the value of the `duration` parameter is 0 (zero), transmission of zero-valued bits is for 250 milliseconds. If `duration` is not 0, transmission of zero-valued bits is for `duration` milliseconds.

**Description**
If the terminal is using asynchronous serial data transmission, the `tcsendbreak()` function causes transmission of a continuous stream of zero-valued bits for a specific duration. If the terminal is not using asynchronous serial data transmission, the `tcsendbreak()` function returns without taking any action.

A process group is sent a SIGTTOU signal if the `tcsendbreak()` function is called from one of its member processes. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation and no signal is sent.

**Notes**
- **AES Support Level:** Full use
Return Values

Upon successful completion, a value of 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors

If the `tcsendbreak()` function fails, `errno` is set to one of the following values:

- [EBADF] The `filedes` parameter does not specify a valid open file descriptor.
- [ENOTTY] The file associated with the `filedes` parameter is not a terminal.

Related Information

Functions: `tcdrain(3)`, `tcflow(3)`, `tcflush(3)`
Files: `termios(4)`
tcsetattr

Purpose
Sets the parameters associated with the terminal

Library
Standard C Library (libc.a)

Synopsis
#include <termios.h>
int tcsetattr ( 
    int file_des, 
    int optional_actions, 
    struct termios *termios_p);

Parameters

file_des Specifies an open file descriptor associated with a terminal.
optional_actions Specifies the options defining how the parameters will be set.
termios_p Points to a termios structure containing the terminal parameters.

Description
The tcsetattr() function sets the parameters associated with the terminal referred to by the open file descriptor from the termios structure referenced by termios_p as follows:

• If optional_actions is TCSANOW, the change will occur immediately.
• If optional_actions is TCSADRAIN, the change will occur after all output written to file_des has been transmitted. This function should be used when changing parameters that affect output.
• If optional_actions is TCSAFLUSH, the change will occur after all output written to file_des has been transmitted, and all input so far received but not read will be discarded before the change is made.

If the output baud rate stored in the termios structure pointed to by the termios_p parameter is the zero baud rate, B0, the modem control lines will no longer be asserted. Normally, this will disconnect the line.

If the input baud rate stored in the termios structure pointed to by the termios_p parameter is zero, the input baud rate given to the hardware will be the same as the output baud rate stored in the termios structure.
Attempts to use the `tcsetattr()` function from a process which is a member of a background process group on a `file_des` associated with its controlling terminal causes the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation and no signal is sent.

Notes

**AES Support Level:** Full use

Return Values

Upon successful completion, 0 (zero) is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors

If the `tcsetattr()` function fails, `errno` may be set to one of the following values:

- **[EBADF]** The `file_des` parameter is not a valid file descriptor.
- **[EINVAL]** The `optional_actions` parameter is not a proper value, or an attempt was made to change an attribute represented in the `termios` structure to an unsupported value.
- **[ENOTTY]** The file associated with the `file_des` parameter is not a terminal.

Related Information

Functions: `cfgetispeed(3)`, `tcgetattr(3)`
tcsetpgrp

**Purpose**
Sets foreground process group ID

**Library**
Standard C Library (libc.a)

**Synopsis**
```c
#include <sys/types.h>

int tcsetpgrp(
    int filedes,
    pid_t pgrp_id);
```

**Parameters**
- **filedes**: Specifies an open file descriptor.
- **pgrp_id**: Specifies the process group identifier.

**Description**
If the process has a controlling terminal, the `tcsetpgrp()` function sets the foreground process group ID associated with the terminal to the value of the `pgrp_id` parameter. The file associated with the `filedes` parameter must be the controlling terminal of the calling process, and the controlling terminal must be currently associated with the session of the calling process. The value of the `pgrp_id` parameter must match a process group ID of a process in the same session as the calling process.

**Notes**

**AES Support Level**: Full use

**Return Values**
Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.
Errors

If the `tcsetpgrp()` function fails, `errno` may be set to one of the following values:

- **[EBADF]** The `filedes` parameter is not a valid file descriptor.
- **[EINVAL]** The `pgrp_id` parameter is invalid.
- **[ENOTTY]** The calling process does not have a controlling terminal, the file is not the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.
- **[EPERM]** The `pgrp_id` parameter is valid, but matches a process ID or process group ID of a process in another session.

Related Information

Functions: `tcgetpgrp(3)`
time

Purpose
Gets time

Library
Standard C Library (libc.a)

Synopsis
#include <time.h>

time_t time(
    time_t *tloc);

Parameters

tloc Points to the location where the return value is stored. When this parameter is a null pointer, no value is stored.

Description
The time() function returns the time in seconds since the epoch. The epoch is referenced to 00:00:00 CUT (Coordinated Universal Time) 1 Jan 1970.

Notes
AES Support Level: Full use

Return Values
Upon successful completion, the time() function returns the value of time in seconds since the epoch. Otherwise, the value ((time_t) - 1) is returned.

Related Information
Functions: clock(3), gettimeofday(2), stime(3)
times

Purpose
Gets process and child process times

Library
Standard C Library (libc.a)

Synopsis
#include <sys/types.h>
#include <sys/times.h>
time_t times(
    struct tms *buffer);

Parameters

buffer Points to type tms structure space where system time information is stored.

Description
The times() function fills the type tms structure space pointed to by the buffer parameter with time-accounting information. All time values reported by this function are in hardware-dependent clock ticks.

The times of a terminated child process are included in the tms_cutime and tms_cstime elements of the parent process when a wait() or waitpid() function returns the process ID of that terminated child.

The tms structure, which is defined in the sys/times.h header file, contains the following members:

time_t tms_utime
User time. The CPU time charged while executing user instructions of the calling process.

time_t tms_stime
System time. The CPU time charged during system execution on behalf of the calling process.
time_t  tms_cutime
    User time, children. The sum of the tms_utime and the tms_cutime
    times of the child processes.

time_t  tms_cstime
    System time, children. The sum of the tms_stime and the
    tms_cstime times of the child processes.

When a child process does not wait for its children, its child-process times are not
included in its times.

This information comes from the calling process and each of its terminated child
processes for which a wait() function has been executed.

Notes

AES Support Level: Full use

Return Values

Upon successful completion, the times() function returns the elapsed real time in
clock ticks since an arbitrary reference time in the past (for example, system start-
up time). This reference time does not change from one times() function to
another. The return value may overflow the possible range of type clock_t values.
When the times() function fails, a value of -1 is returned.

Related Information

Functions: exec(2), fork(2), getrusage(2), profi(2), stime(3), sysconf(3),
time(3), wait(2)

Commands: cc(1)
tmpfile

**Purpose**  Creates a temporary file

**Library**  Standard I/O Package (libc.a)

**Synopsis**

```c
#include <stdio.h>

FILE *tmpfile ( void );
```

**Description**

The `tmpfile()` function creates a temporary file and returns its `FILE` pointer. The file is opened for update. The temporary file is automatically deleted when the process using it terminates.

**Notes**

**AES Support Level:** Full use

**Return Values**

Upon successful completion, the `tmpfile()` function returns a pointer to the stream of the file that is created. Otherwise, it returns a null pointer and sets `errno` to indicate the error.

**Errors**

If the `tmpfile()` function fails, `errno` may be set to one of the following values:

- `[EMFILE]` OPEN_MAX file descriptors are currently open in the calling process.
- `[ENFILE]` Too many files are currently open in the system.
- `[ENOSPC]` The directory or file system that would contain the new file cannot be expanded.

**Related Information**

Functions: `fopen(3)`, `mktemp(3)`, `tmpnam(3)`, `unlink(2)`
tmpnam, tempnam

Purpose
Constructs the name for a temporary file

Library
Standard I/O Package (libc.a)

Synopsis
```
#include <stdio.h>

char *tmpnam (char *s);

char *tempnam (const char *directory, const char *prefix);
```

Parameters

- **s**: Specifies the address of an array of at least the number of bytes specified by L_tmpnam, a constant defined in the stdio.h header file.
- **directory**: Points to the pathname of the directory in which the file is to be created.
- **prefix**: Points to an initial character sequence with which the filename begins. The prefix parameter can be null, or it can point to a string of up to five characters to be used as the first characters of the temporary filename.

Description

The tmpnam() and tempnam() functions generate filenames for temporary files.

The tmpnam() function generates a filename using the pathname defined as P_tmpdir in the stdio.h header file.

Files created using this function reside in a directory intended for temporary use, and their names are unique. It is the user's responsibility to use the unlink() function to remove the file when it is no longer needed.

Between the time a filename is created and the file is opened, it is possible for some other process to create a file with the same name. This should not happen if that other process uses these functions or the mktemp() function, and if the filenames are chosen to make duplication by other means unlikely.
The `tmpnam()` function allows you to control the choice of a directory. If the `directory` parameter is null or points to a string that is not a pathname for an appropriate directory, the pathname defined as `P_tmpdir` in the `stdio.h` header file is used. If that pathname is not accessible, `/tmp` is used. You can bypass the selection of a pathname by providing an environment variable, `TMPDIR`, in the user's environment. The value of the `TMPDIR` variable is a pathname for the desired temporary file directory.

The `prefix` parameter can be used to specify a prefix of up to 5 characters for the temporary filename.

**Notes**

The `tmpnam()` function generates a different filename each time it is called. If it is called more than `TMP_MAX` times by a single process, it starts recycling previously used names.

AES Support Level: Trial use

**Return Values**

If the `s` parameter is null, the nonreentrant version of the `tmpnam()` function places its result into an internal static area and returns a pointer to that area. The next call to this function destroys the contents of the area. The reentrant version of the `tmpnam()` function always returns null if `s` is null.

If the `s` parameter is not null, it is assumed to be the address of an array of at least the number of bytes specified by `L_tmpnam`. `L_tmpnam` is a constant defined in the `stdio.h` header file. The `tmpnam()` function places its results into that array and returns the value of the `s` parameter.

Upon successful completion, the `tempnam()` function returns a pointer to the generated pathname, suitable for use in a subsequent call to the `free()` function. Otherwise, null is returned and `errno` is set to indicate the error.

**Errors**

If the `tempnam()` function fails, `errno` may be set to the following value:

`ENOOMEM` Insufficient storage space is available.

**Related Information**

Functions: `fopen(3)`, `free(3)`, `malloc(3)`, `mktemp(3)`, `open(2)`, `tmpfile(3)`, `unlink(2)`
truncatem, ftruncate

**Purpose**  
Changes file length

**Synopsis**  
```c
#include <sys/types.h>

int truncate (const char *path, off_t length);

int ftruncate (int filedes, off_t length);
```

**Parameters**
- **path**  
  Specifies the name of a file that is opened, truncated, and then closed. The *path* parameter must point to a pathname which names a regular file for which the calling process has write permission. If the *path* parameter refers to a symbolic link, the length of the file pointed to by the symbolic link will be truncated.

- **filedes**  
  Specifies the descriptor of a file that must be open for writing.

- **length**  
  Specifies the new length of the file in bytes.

**Description**

The `truncate()` and `ftruncate()` functions change the length of a file to the size in bytes specified by the *length* parameter. If the new length is less than the previous length, the `truncate()` and `ftruncate()` functions remove all data beyond *length* bytes from the specified file. All file data between the new End-of-File and the previous End-of-File is discarded. If the new length is greater than the previous length, new file data between the previous End-of-File and the new End-of-File will be added, consisting of all zeros.

Full blocks are returned to the file system so that they can be used again, and the file size is changed to the value of the *length* parameter.

The `truncate()` and `ftruncate()` functions have no effect on FIFO special files or directories. These functions do not modify the seek pointer of the file.
Upon successful completion, the `truncate()` and `ftruncate()` functions mark the `st_ctime` and `st_mtime` fields of the file for update. If the file is a regular file, the `ftruncate()` and `truncate()` functions clear the `S_ISUID` and `S_ISGID` attributes of the file.

If the file has enforced file locking enabled and there are file locks on the file, the `truncate()` or `ftruncate()` function fails.

**Notes**

**AES Support Level:** Trial use

**Return Values**

Upon successful completion, a value of 0 (zero) is returned. If the `truncate()` or `ftruncate()` function fails, it returns a value of -1, and `errno` is set to indicate the error.

**Errors**

If the `truncate()` or `ftruncate()` function fails, `errno` may be set to one of the following values:

- `[EINVAL]` The file is not a regular file.
- `[EISDIR]` The file is a directory.
- `[EAGAIN]` The write operation failed due to an enforced write lock on the file.
- `[EACCES]` Write access permission to the file was denied.
- `[EFBIG]` The new file size would exceed the process' file size limit or the maximum file size.
- `[EROFS]` The file resides on a read-only file system.
- `[EAGAIN]` The file has enforced mode file locking enabled and there are file locks on the file.
In addition, the `truncate()` function fails if errors occur that apply to any service requiring pathname resolution, or if one of the following are true:

- **[ENAMETOOLONG]** The size of the pathname exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- **[ENOENT]** A component of the specified pathname does not exist, or the `path` parameter points to an empty string.
- **[ENOTDIR]** A component of the path prefix is not a directory.

In addition, if the `ftruncate()` function fails, `errno` may be set to the following value:

- **[EBADF]** The `filedes` parameter is not a valid file descriptor open for writing.

**Related Information**

Functions: `chmod(2)`, `fcntl(2)`, `open(2)`
tsearch, tfind, tdelete, twalk

Purpose  Manages binary search trees

Library  Standard C Library (libc.a)

Synopsis  
#include <search.h>

void *tsearch(
    const void *key,
    const void *rootp,
    int (*compar)(const void *, const void *));

void *tfind(
    const void *key,
    const void **rootp,
    int (*compar)(const void *, const void *));

void *tdelete(
    const void *key,
    const void **rootp,
    int (*compar)(const void *, const void *));

void twalk( const void **rootp,
    void (*action)(const void *, const enunt VISIT, const int));

Parameters

key  Points to a key that specifies the entry to be searched in the binary tree.

rootp  Points to a variable that points to the root of the binary tree.

compar  Specifies the name (that you supply) of a comparison function (strcmp(), for example). This function is called with two parameters that point to the data undergoing comparison in the binary tree.

action  The name of a routine to be invoked at each node during a walk through the binary tree.
The tsearch(), tfind(), tdelete() and twalk() functions are used to operate on binary search trees. Comparisons are done with a function that you supply (strcmp(), for example). The address of the compare function is passed as the compar parameter in these functions. The compare function must be called with two parameters that point to entries in the binary tree undergoing comparison.

The tsearch() function is used to build and access a binary tree during a search. The key parameter is a pointer to an entry that is accessed or stored. When an entry in the binary tree compares with (is equal to) the value pointed to by the key parameter, a pointer to this entry in the binary tree is returned. Otherwise, the value pointed to by the key parameter is inserted into the binary tree in its proper place, and a pointer to the inserted key is returned. Only pointers are copied, so the calling routine must store the data.

The rootp parameter points to a variable that points to the root of a binary tree. When a null value is specified for the rootp parameter, an empty tree is specified; in this case, the variable pointed to by the rootp parameter is set to point to the entry, which is located at the root of a new tree.

As with the tsearch() function, the tfind() function searches for an entry in the binary tree, returning a pointer to that entry in the binary tree when a match with the key parameter occurs. However, when key is not matched, this function returns a null pointer.

The tdelete() function deletes a node from a binary search tree. Parameters for this function are used in the same way as for the tsearch() function. The variable pointed to by the rootp parameter is changed when the deleted node is the root of the binary tree. The tdelete() function returns a pointer to the parent of the deleted node.

The twalk() function traverses a binary search tree. The rootp parameter specifies the root of the binary tree to be traversed. Any node in a binary tree may be used as the root node for a traverse at the level below the specified root node. The action parameter is the name of a routine to be invoked at each node. This action routine is called with three parameters. The first parameter specifies the address of the visited node. The second parameter specifies a value from an enum data type, which is defined in the search.h include file as follows:

\[
\text{typedef}@\text{enum} \{ \text{preorder}, \text{postorder}, \text{endorder}, \text{leaf} \} \text{VISIT}
\]

The value of the second parameter in the action routine depends on whether this is the first (preorder), second (postorder), or third (endorder) time that the node has been visited during a depth-first left-to-right traversal of the tree, or whether the node is a leaf. (A leaf is a node that is not the parent of another node). The third parameter in the action routine is the level of the node in the binary tree; the root level of a binary tree is 0 (zero).
Notes

The comparison function need not compare every byte; consequently, arbitrary data may be contained in the searched keys in addition to the values undergoing comparison.

Pointers to keys and to roots of binary trees should be of type `pointer-to-element` and cast to type `pointer-to-character`. Although declared as type `pointer-to-character`, the value returned should be cast to type `pointer-to-element`.

AES Support Level: Trial use

Return Values

When the `key` parameter is matched with an entry in the binary tree, both the `tsearch()` and `tfind()` functions return a pointer to the matching entry in the binary tree. When `key` remains unmatched, the `tfind()` function returns a null pointer. The `tsearch()` function returns a pointer to the inserted entry.

A null pointer is returned by the `tsearch()` function when there is not enough space available in the binary tree to create a new node.

A null pointer is returned by the `tsearch()`, `tfind()`, and `tdelete()` functions when the `rootp` parameter is set to the null pointer value.

No value is returned by the `twalk()` function.

Errors

If the `tsearch()`, `tfind()`, `twalk()`, or `tdelete()` function fails, `errno` may be set to the following value:

 `[ENOMEM]` Insufficient storage space is available to add an entry to the binary tree.

Related Information

Functions: `bsearch(3)`, `hsearch(3)`, `lsearch(3)`
ttyname, isatty

Purpose

Gets the name of a terminal

Library

Standard C Library (libc.a)

Synopsis

char *ttyname(
    int file_descriptor );

int isatty(
    int file_descriptor );

int ttyname_r(
    int file_descriptor,
    char *buffer,
    int len );

Parameters

file_descriptor
    Specifies an open file descriptor.

buffer
    Points to a buffer in which the terminal name is stored.

len
    Specifies the length of the buffer pointed to by the buffer parameter.

Description

The ttyname() function gets the name of a terminal. It returns a pointer to a string containing the null-terminated pathname of the terminal device associated with the file_descriptor parameter.

The isatty() function determines if the device associated with the file_descriptor parameter is a terminal. If so, the isatty() function returns a value of 1. If the file descriptor is not associated with a terminal, a value of 0 (zero) is returned.

The ttyname_r() function is the reentrant version of the ttyname() function. Upon successful completion, the terminal name is stored as a null-terminated string in the buffer pointed to by the buffer parameter.
Notes

AES Support Level: Full use

Return Values

The `ttyname()` function returns a pointer to a string which is static data that is overwritten by each call. A null pointer is returned if the `file_descriptor` parameter does not describe a terminal device in the directory `/dev`.

The `isatty()` function returns a value of 1 if the specified file descriptor is associated with a terminal, and 0 (zero) otherwise.

The `ttyname_r()` function returns 0 (zero) if successful. Otherwise, -1 is returned.

Errors

If the `ttyname_r()` function fails, `errno` may be set to the following value:

[EINVAL] The `buffer` parameter is a null pointer or the `len` parameter was too short to store the string.

If the `isatty()` function fails, `errno` may be set to the following value:

[ENOTTY] The file associated with `filedes` is not a terminal.

Related Information

Functions: `ttyslot(3)`
ttyslot

**Purpose**  Finds the slot in the *utmp* file for the current user

**Library**  Standard C Library (*libc.a*)

**Synopsis**  
```c
int ttyslot ( void );
```

**Description**

The `ttyslot()` function returns the index of the current user's entry in the `/etc/utmp` file. The `ttyslot()` function scans the `/etc/utmp` file for the name of the terminal associated with the standard input, the standard output, or the error output file descriptors (0, 1, or 2).

**Return Values**

Upon successful completion, the `ttyslot()` function returns the index of the current user's entry in the `/etc/utmp` file. Otherwise, if an error is encountered while searching for the terminal name, or if none of the first three file descriptors (0, 1, and 2) is associated with a terminal device, 0 (zero) is returned.

**Related Information**

Functions: `getutent(3)`, `ttyname(3)`
ulimit

**Purpose**
Sets and gets user limits

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <ulimit.h>

long ulimit (  
    int command,  
    off_t new_limit,  
    ...  
);
```

**Parameters**
- `command` Specifies the form of control. The `command` parameter values follow:
  - **UL_GETFSIZE ()**
    Returns the process file size limit. The limit is in units of UBSIZE blocks (see the sys/param.h file) and is inherited by child processes. Files of any size can be read.
  - **UL_SETFSIZE ()**
    Sets the process file size limit for output operations to the value of the `new_limit` parameter, and returns the new file size limit. Any process can decrease this limit, but only a process with the SEC_LIMIT system privilege can increase the limit.
  - **GET_GETBREAK ()**
    Returns the maximum possible break value (described in the brk() and sbrk() functions).
- `new_limit` Specifies the new limit. The value of the `new_limit` parameter depends on the `command` parameter value that is used.
ulimit( ) function controls process limits.
With remote files, the ulimit( ) function values of the client node or local node are used.

Notes
The ulimit( ) function is implemented in terms of setrlimit( ); therefore, the two interfaces should not be used in the same program. The result of doing so is undefined.
AES Support Level: Trial use

Example
To increase the size of the stack segment by 2048 bytes, and set the rc variable to the new lowest valid stack address, enter:

\[
rc = \text{ulimit}(1006, \text{ulimit}(1005, 0) - 2048);
\]

Return Values
Upon successful completion, a nonnegative value is returned. If the ulimit( ) function fails, a value of -1 is returned and errno is set to indicate the error.

Errors
If the ulimit( ) function fails, the limit remains unchanged and errno may be set to one of the following values:

EPERM A process without appropriate system privilege attempts to increase the file size limit.

EINVAL The command parameter is invalid.

Related Information
Functions: brk(2), getrlimit(2), pathconf(3), write(2)
umask

**Purpose**
Sets and gets the value of the file creation mask

**Synopsis**
```c
#include <sys/types.h>
#include <sys/stat.h>

mode_t umask (mode_t cmask);
```

**Parameters**
- `cmask` Specifies the value of the file mode creation mask.

**Description**
The `umask()` function sets the file mode creation mask of the process to the value of the `cmask` parameter and returns the previous value of the mask. The `cmask` parameter is constructed by logically ORing file permission bits defined in the `sys/stat.h` header file.

Whenever a file is created (by the `open()`, `mkdir()`, or `mknod()` function), all file permission bits set in the file mode creation mask are cleared in the mode of the created file. This clearing allows users to restrict the default access to their files.

The mask is inherited by child processes.

**Notes**

AES Support Level: Full use

**Return Values**
Upon successful completion, the previous value of the file mode creation mask is returned.

**Related Information**
Functions: `chmod(2)`, `mkdir(2)`, `mknod(2)`, `open(2)`, `stat(2)`
Commands: `chmod(1)`, `mkdir(1)`, `sh(1)`, `umask(1)`
umount

Purpose
Unmounts a file system

Library
System V Compatibility Library (libsys5.a)

Synopsis
#include <sys/mount.h>

int umount(
    char *spec);

Parameters

spec  Points to the pathname of the special file or file system to be unmounted.

Description

The umount() function unmounts a previously-mounted file system contained on the block special file pointed to by the spec parameter. When the file system is unmounted, the directory mount point where the file system was mounted returns to its normal interpretation.

The umount() function can only be invoked by the superuser.

Notes

Two umount() functions are supported by OSF/1: the BSD umount() and the System V umount(). The default umount() function is the BSD umount(). To use the version of umount() documented on this reference page, you must link with the libsys5 library before you link with libc.

Return Value

The umount() function returns 0 (zero) if the file system was successfully unmounted. Otherwise, -1 is returned and errno is set to indicate the error.
Errors

If the `umount()` function fails, `errno` may be set to one of the following values:

- `[EPERM]` The effective user ID of the calling process is not root.
- `[ENOENT]` The `spec` parameter points to a pathname that does not exist.
- `[ENOTDIR]` A component of the path prefix of `spec` is not a directory.
- `[ENOTBLK]` The device identified by `spec` is not a block-special device.
- `[ENXIO]` The device identified by `spec` does not exist.
- `[EBUSY]` A file on the device pointed to by the `spec` parameter is busy.
- `[EINVAL]` The device pointed to by the `spec` parameter is not mounted.

Related Information

Commands: `mount(8)`
uname

Purpose      Gets the name of the current system

Synopsis    
    #include <sys/utsname.h>
    int uname (struct utsname *name);

Parameters
    name       Points to a utsname structure.

Description
    The uname() function stores information identifying the current system in the
    structure pointed to by the name parameter.

    The uname() function uses the utsname structure, which is defined in the
    sys/utsname.h file and contains the following members:

        char    sysname[SYS_NMLN];
        char    nodename[SYS_NMLN];
        char    release[SYS_NMLN];
        char    version[SYS_NMLN];
        char    machine[SYS_NMLN];

    The uname() function returns a null-terminated character string naming the
    current system in the sysname character array. The nodename array contains the
    name that the system is known by on a communications network. The release and
    version arrays further identify the system. The machine array identifies the CPU
    hardware being used.

Notes

    AES Support Level: Full use
Functions
uname(2)

Return Values

Upon successful completion, a nonnegative value is returned. Otherwise, -1 is returned and errno is set to indicate the error.

Errors

If the uname() function fails, errno may be set to the following value:

[EFAULT] The name parameter points outside of the process address space.

Related Information

Commands: uname(1)
ungetc, ungetwc

Purpose
Pushes a character back into input stream

Library
Standard I/O Package (libc.a)

Synopsis
#include <stdio.h>
int ungetc (int character,
FILE *stream);

Parameters
character Specifies a character.
stream Specifies the input stream.

Description
The ungetc() function inserts the character specified by the character parameter into the buffer associated with the input stream specified by the stream parameter. This causes the next call to the getc() function to return character. The ungetc() function returns character, and leaves the stream parameter file unchanged.

If the character parameter is EOF, the ungetc() function does not place anything in the buffer and a value of EOF is returned.

You can push one character back onto a stream, provided that something has been read from the stream or the setbuf() function has been called. The fseek() subroutine erases all memory of inserted characters.

The ungetc() function returns a value of EOF if it cannot insert the character.

Notes
When running with Japanese Language Support, the following function, stored in libc.a, is provided:

#include <stdio.h>
int ungetwc (int character,
FILE *stream);
The `ungetwc()` function inserts the `NLchar` specified by the `character` parameter into the buffer associated with the input stream. This causes the next call to the `getwc()` function to return the value of the `character` parameter.

**AES Support Level:** Full use

**Return Values**

The `ungetwc()` function returns a value of EOF if the character cannot be inserted.

**Related Information**

Functions: `fseek(3), getc(3), getwc(3), setbuf(3)`
unlink(2)

unlink

Purpose
Removes a directory entry

Synopsis
```c
int unlink (const char *path);
```

Parameters
- `path` Specifies the directory entry to be removed.

Description
The `unlink()` function removes the directory entry specified by the `path` parameter and, if the entry is a hard link, decrements the link count of the file referenced by the link.

When all links to a file are removed and no process has the file open or mapped, all resources associated with the file are reclaimed, and the file is no longer accessible. If one or more processes have the file open or mapped when the last link is removed, the link will be removed before the `unlink()` function returns, but the removal of the file contents is postponed until all open or map references to the file are removed. If the `path` parameter names a symbolic link, the symbolic link itself is removed.

Removing a hard link to a directory requires superuser privilege.

Upon successful completion, the `unlink()` function marks for update the `st_ctime` and `st_mtime` fields of the directory which contained the link. If the file's link count is not 0 (zero), the `st_ctime` field of the file is also marked for update.

Notes
AES Support Level: Full use

Return Values
Upon successful completion, a value of 0 (zero) is returned. If the `unlink()` function fails, a value of -1 is returned, the named file is not changed, and `errno` is set to indicate the error.
Errors

If the `unlink()` function fails, the named file is not unlinked and `errno` may be set to one of the following values:

- **[ENOENT]** The named file does not exist or the `path` parameter points to an empty string.
- **[EACCES]** Search permission is denied for a component of the path prefix, or write permission is denied on the directory containing the link to be removed.
- **[EPERM]** The named file is a directory, and the calling process does not have superuser privilege.
- **[EPERM]** The S_ISVTX flag is set on the directory containing the file to be deleted, and the caller is not the file owner.
- **[EBUSY]** The entry to be unlinked is the mount point for a mounted file system.
- **[EROFS]** The entry to be unlinked is part of a read-only file system.
- **[EFAULT]** The `path` parameter is an invalid address.
- **[ELOOP]** Too many links were encountered in translating `path`.
- **[ENAMETOOLONG]** The length of the `path` parameter exceeds `PATH_MAX` or a pathname component is longer than `NAME_MAX`.
- **[ENOTDIR]** A component of the path prefix is not a directory.

Related Information

Functions: `close(2)`, `link(2)`, `open(2)`, `rmdir(2)`

Commands: `rm(1)`
unload

Purpose
Unloads a previously loaded module

Library
Loader Library libld.a

Synopsis
#include <sys/types.h>
#include <loader.h>
int unload(
   ldr_module_t mod_id);

Parameters
mod_id
Specifies the identifier for the module to be unloaded. The module ID is returned when the module is first loaded.

Description
The unload() function unloads the specified module from the virtual address space of the calling process. The function unmaps the module’s regions and discards the loader data structures that describe the module.

The module is unloaded even if any references to it remain in other modules. The loader does not keep track of such dangling references or attempt to unsnap any invalidated links. These housekeeping tasks are the responsibility of the calling process. Attempts to refer to addresses in an unloaded module can result in indeterminate errors.

Notes
Once a module has been unloaded, its module ID is no longer valid.

Return Values
Upon successful completion, the unload() function returns a value of 0 (zero). If the unload fails, the function returns a value of -1 and errno is set to indicate the error.
Errors

If the `unload()` function fails, `errno` may be set to one of the following values:

- `[EINVAL]` The specified module ID cannot be unloaded or is not valid.
- `[EDUPPKG]` The loaded module exported a package which duplicated the package name of a module already loaded in the same process.

Related Information

Functions: `load(2), ldr_xunload(2)`
unlocked_getc, unlocked_getchar

**Purpose**
Gets a character from an input stream

**Library**
Standard C Library (libc.a)

**Synopsis**
```
#include <stdio.h>
int unlocked_getc(
    FILE * file);
int unlocked_getchar ( void );
```

**Parameters**

- **file**
  Specifies the input stream.

**Description**
The `unlocked_getc()` and `unlocked_getchar()` functions are functionally identical to the `getc()` and `getchar()` functions, except that `unlocked_getc()` and `unlocked_getchar()` may be safely used only within a scope that is protected by the `flockfile()` and `funlockfile()` functions used as a pair. The caller must ensure that the stream is locked before these functions are used.

**Return Values**
The integer constant EOF is returned at the end of the file or upon an error.

**Related Information**
Functions: `flockfile(3), funlockfile(3), getc(3)`
unlocked_putc, unlocked_putchar

Purpose  Writes a character to a stream

Library

Standard C Library (libc.a)

Synopsis

```c
#include <stdio.h>

int unlocked_putchar(
    char c,
    FILE *file);

int unlocked_putchar(
    char c);
```

Parameters

- `file` Specifies the stream.
- `c` Specifies the character to be written.

Description

The `unlocked_putchar()` functions are functionally identical to the `putc()` and `putchar()` functions, except that `unlocked_putchar()` may be safely used only within a scope that is protected by the `flockfile()` and `funlockfile()` functions used as a pair. The caller must ensure that the stream is locked before these functions are used.

Return Values

Upon successful completion, the value written is returned. Otherwise, the constant EOF is returned.

Related Information

Functions: `flockfile(3), funlockfile(3), putc(3)`
usleep

Purpose          Suspends execution for an interval

Library          Standard C Library (libc.a)

Synopsis         unsigned usleep(
                  unsigned mseconds);

Parameters

mseconds         The number of microseconds to suspend execution for.

Description

The usleep() function suspends the current process from execution for the number of microseconds specified by the mseconds parameter. Because of other activity in the system, or because of the time spent in processing the call, the actual suspension time may be longer than specified.

The usleep() function sets an interval timer and pauses until it occurs. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous timer, the process sleeps only until the signal would have occurred, and the signal is sent a short time later.

The usleep() function uses the setitimer() function. It requires eight system calls each time it is invoked. A similar but less compatible function can be obtained with a single select; it would not restart after signals, but would not interfere with other uses of setitimer().

Related Information

Functions: alarm(3), getitimer(2), sigaction(2), sigvec(2), sleep(3)
utime, utimes

Purpose
Sets file access and modification times

Synopsis

```c
#include <sys/time.h>
#include <utime.h>
#include <sys/types.h>

int utime(
    const char *path,
    struct utimbuf *times);

int utimes(
    const char *path,
    struct timeval times[2];
```

Parameters

- `path`: Points to the file. If the final component of the `path` parameter names a symbolic link, the link will be traversed and pathname resolution will continue.

- `times`: Points to a `utimbuf` structure for the `utime()` function, or to an array of `timeval` structures for the `utimes()` function.

Description

The `utimes()` function sets the access and modification times of the file pointed to by the `path` parameter to the value of the `times` parameter. The `utimes()` function allows time specifications accurate to the microsecond.

The `utime()` function also sets file access and modification times; however, each time is contained in a single integer and is accurate only to the nearest second.
For **utime()**, the *times* parameter is a pointer to a **utimbuf** structure, defined in the **utime.h** header file. The first structure member represents the date and time of last access, and the second member represents the date and time of last modification. The times in the **utimbuf** structure are measured in seconds since the epoch (00:00:00, January 1, 1970, Coordinated Universal Time (CUT)).

For **utimes()**, the *times* parameter is an array of **timeval** structures, as defined in the **sys/time.h** header file. The first array element represents the date and time of last access, and the second element represents the date and time of last modification. The times in the **timeval** structure are measured in seconds and microseconds since the epoch, although rounding towards the nearest second may occur.

If the *times* parameter is null, the access and modification times of the file are set to the current time. If the file is a remote file, the current time at the remote node, rather than the local node, is used. The effective user ID of the process must be the same as the owner of the file, or must have write access to the file or superuser privilege in order to use the call in this manner.

If the *times* parameter is not null, the access and modification times are set to the values contained in the designated structure, regardless of whether those times correlate with the current time. Only the owner of the file or a user with superuser privilege can use the call this way.

Upon successful completion, the **utime()** and **utimes()** functions mark the time of the last file status change, **st_ctime**, for update.

**Notes**

**AES Support Level:** Full use

**Return Values**

Upon successful completion, a value of 0 (zero) is returned. Otherwise, a value of -1 is returned, **errno** is set to indicate the error, and the file times will not be affected.

**Errors**

If the **utimes()** or **utime()** function fails, **errno** may be set to one of the following values:

- **[ENOENT]** The named file does not exist or the *path* parameter points to an empty string.
- **[EPERM]** The *times* parameter is not the null value and the calling process has write access to the file but neither owns the file nor has the appropriate system privilege.
[EACCES] Search permission is denied by a component of the path prefix; or the times parameter is null and effective user ID is neither the owner of the file nor has appropriate system privilege, and write access is denied.

[EROFS] The file system that contains the file is mounted read-only.

[EFAULT] The path parameter is an invalid address, or (for utimes()) either the path or times parameter is an invalid address.

[ELOOP] Too many links were encountered in translating path.

[ENAMETOOLONG] The length of the path parameter exceeds PATH_MAX or a pathname component is longer than NAME_MAX.

[ENOTDIR] A component of the path prefix is not a directory.

The utimes() function can also fail if additional errors occur.

Related Information

Functions: stat(2)
Purpose
Handles a variable-length parameter list

Library
Standard C Library (libc.a)

Synopsis
#include <stdarg.h>
va_alist
va_dcl
void va_start (  
    va_alist argp );
type va_arg (  
    va_alist argp,  
    type );
void va_end (  
    va_alist argp );

Parameters

argp
Specifies a variable that the varargs macros use to keep track of the current location in the parameter list. Do not modify this variable.

type
Specifies the type to which the expected argument will be converted when passed as an argument. In C, arguments that are char or short should be accessed as int; unsigned char or short are converted to unsigned int, and float arguments are converted to double. Different types can be mixed, but it is up to the routine to know what type of argument is expected, since it cannot be determined at runtime.
Description

The varargs set of macros allows you to write portable functions that accept a variable number of parameters. Subroutines that have variable-length parameter lists (such as the printf() function), but that do not use the varargs macros, are inherently nonportable because different systems use different parameter-passing conventions.

The varargs macros are as follows:

va_alist()  Defines the type of the variable used to traverse the list.
va_start()  Initializes argp to point to the beginning of the list. The va_start() macro will be invoked before any access to the unnamed arguments.
va_argp()   A variable that the varargs macros use to keep track of the current location in the parameter list. Do not modify this variable.
va_arg()    Returns the next parameter in the list pointed to by argp.
va_end()    Cleans up at the end.

Your function can traverse, or scan, the parameter list more than once. Start each traversal with a call to va_start() and end it with va_end().

Example

The following example is a possible implementation of the execl() function:

```c
#include <varargs.h>
define MAXargS 100
/*
 ** execl is called by
 ** execl(file, arg1, arg2, . . . , (char *) 0);
 */
excl(va_alist)
  va_dcl
  { va_alist ap;
    char *file;
    char *args[MAXargS];
    int argno = 0;
    va_start(ap);
    file = va_arg(ap, char *);
    while ((args[argno++] = va_arg(ap, char *)) != (char *) 0)
      /* Empty loop body */
    va_end(ap);
    return (execv(file, args));
  }
```
Notes

The calling routine is responsible for specifying the number of parameters because it is not always possible to determine this from the stack frame. For example, the `exec()` function is passed a null pointer to signal the end of the list. The `printf()` function determines the number of parameters from its `fmt` parameter.

AES Support Level: Temporary use

Related Information

Functions: `exec(2)`, `printf(3)`, `vprintf(3)`
vprintf, vfprintf, vsprintf

Purpose
Formats a varargs parameter list for output

Library
Standard I/O Package (libc.a)

Synopsis
```
#include <stdio.h>
#include <stdarg.h>

int vprintf (const char *format, va_list printarg);

int vfprintf (FILE *stream, const char *format, va_list printarg);

int vsprintf (char *string, const char *format, va_list printarg);
```

Parameters

- **format**: Specifies a character string that contains two types of objects:
  - Plain characters, which are copied to the output stream.
  - Conversion specifications, each of which causes zero or more items to be fetched from the varargs parameter lists.

- **printarg**: Specifies the arguments to be printed.

- **stream**: Specifies the output stream.

- **string**: Specifies the buffer to which output is printed.

Description

The vprintf(), vfprintf(), and vsprintf() functions format and write varargs parameter lists.
These functions are the same as the `printf()`, `fprintf()`, and `sprintf()` functions, respectively, except that they are not called with a variable number of parameters. Instead, they are called with a parameter list pointer as defined by `varargs`.

**Notes**

**AES Support Level:** Full use

**Example**

The following example demonstrates how the `vfprintf()` function can be used to write an error routine:

```c
#include <stdio.h>
#include <varargs.h>
/* error should be called with the syntax: */
/* error(routine_name, Format [, value, . . . ])); */
/*VARARGS0*/

void error(va_alist) va_dcl;
/* ** Note that the function name and Format arguments **
cannot be separately declared because of the **
definition of varargs. */
{ 
  va_list args;
  char *fmt;
  void fprintf() , vfprintf() , abort();
  va_start(args);
  /*
  ** Display the name of the function that called error
  */
  fprintf(stderr, "ERROR in %s: ", va_arg(args, char *));
  /*
  ** Display the remainder of the message
  */
  fmt = va_arg(args, char *);
  vfprintf(fmt, args);
  va_end(args);
  abort(); }

**Related Information**

Functions: `printf(3)`
wait, waitpid, wait3

Purpose  Waits for a child process to stop or terminate

Synopsis  
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait (  
    int *status_location);

pid_t waitpid (  
    pid_t process_id,  
    int *status_location,  
    int options);

#include <sys/resource.h>

pid_t wait3 (  
    union wait *status_location,  
    int options,  
    struct rusage *resource_usage);

Parameters  

status_location  Points to a location that is filled in with the child process termination status, as defined in the sys/wait.h header file.

process_id  Specifies the child process.

options  Modifies the behavior of the function.

resource_usage  Specifies the location of a structure to be filled in with resource utilization information for terminated child processes.

Description  
The wait() function suspends the calling process until it receives a signal that is to be caught, or until any one of the calling process’ child processes stops or terminates. The wait() function returns without waiting if a child process that has not been waited for has already stopped or terminated prior to the call.

The effect of the wait() function can be modified by the setting of the SIGCHLD signal. (See the sigaction() function for details.)
The `waitpid()` function behaves identically to the `wait()` function if the `process_id` parameter has a value of -1 and the `options` parameter has a value of 0 (zero). Otherwise, its behavior is modified by the values of the `process_id` and `options` parameters.

The `wait()`, `waitpid()`, and `wait3()` functions, which suspend the calling process until the request is completed, are redefined so that only the calling thread is suspended.

The `process_id` parameter allows the calling process to gather status from a specific set of child processes, according to the following rules:

- If the `process_id` parameter is equal to -1, status is requested for any child process. In this respect, the `waitpid()` function is equivalent to the `wait()` function.
- If the `process_id` parameter is greater than 0 (zero), it specifies the process ID of a single child process for which status is requested.
- If the `process_id` parameter is equal to 0 (zero), status is requested for any child process whose process group ID is equal to that of the calling process.
- If the `process_id` parameter is less than -1, status is requested for any child process whose process group ID is equal to the absolute value of the `process_id` parameter.

The `waitpid()` function will only return the status of a child process from this set.

The `options` parameter to both the `waitpid()` and `wait3()` functions modifies the behavior of the function. Two values are defined, WNOHANG and WUNTRACED, which can be combined by specifying their bitwise-inclusive OR. The WNOHANG option prevents the calling process from being suspended even if there are child processes to wait for. In this case, 0 (zero) is returned indicating that there are no child processes that have stopped or terminated. If the WUNTRACED option is set, the call also returns information when child processes of the current process are stopped because they received a SIGTTIN, SIGTTOU, SIGSTOP, or SIGTSTOP signal.

If the `wait()`, `waitpid()`, or `wait3()` function returns because the status of a child process is available, they return the process ID of the child process. In this case, if the `status_location` parameter is not null, information will be stored in the location pointed to by `status_location`. The value stored at the location pointed to by `status_location` is 0 (zero) if and only if the status returned is from a terminated child process that returned 0 (zero) from the `main()` routine, or passed 0 (zero) as the `status` parameter to the `_exit()` or `exit()` function. Regardless of its value, this information can be interpreted using the following macros, which are defined in the `sys/wait.h` header file and evaluate to integral expressions; the `status_value` parameter is the integer value pointed to by `status_location`. 

1–924
Functions

wait(2)

WIFEXITED(status_value)
   Evaluates to a nonzero value if status was returned for a child process that terminated normally.

WEXITSTATUS(status_value)
   If the value of WIFEXITED(status_value) is nonzero, this macro evaluates to the low-order 8 bits of the status parameter that the child process passed to the _exit() or exit() functions, or the value the child process returned from the main() routine.

WIFSIGNALED(status_value)
   Evaluates to nonzero value if status was returned for a child process that terminated due to the receipt of a signal that was not caught.

WTERMSIG(status_value)
   If the value of WIFSIGNALED(status_value) is nonzero, this macro evaluates to the number of the signal that caused the termination of the child process.

WIFSTOPPED(status_value)
   Evaluates to a nonzero value if status was returned for a child process that is currently stopped.

WSTOPSIG(status_value)
   If the value of WIFSTOPPED(status_value) is nonzero, this macro evaluates to the number of the signal that caused the child process to stop.

If the information stored at the location pointed to by the status_location parameter was stored there by a call to the waitpid() or wait3() function that specified the WUNTRACED flag, exactly one of the WIFEXITED(*status_location), WIFSIGNALED(*status_location), and WIFSTOPPED(*status_location) macros will evaluate to a nonzero value. If the information stored at the location pointed to by the status_location function was stored there by a call to waitpid() or wait3() that did not specify the WUNTRACED flag or by a call to the wait() function, exactly one of the WIFEXITED(*status_location) and WIFSIGNALED(*status_location) macros will evaluate to a nonzero value.

The wait3() function is provided for compatibility with BSD systems. A program that calls wait3() must be compiled with the _BSD switch defined. In this case, the parameter to the macros described above should be the w_status member of the union pointed to by status_location. The wait3() function also provides a resource_usage parameter that points to a location in which resource usage information for the child process is stored, as defined in the sys/resource.h function.
OSF/1 Programmer's Reference

wait(2)

If a parent process terminates without waiting for all of its child processes to terminate, the remaining child processes will be assigned a parent process ID equal to the process ID of init.

Notes

If a program that calls the wait() function is compiled with the _BSD switch defined and linked with the libbsd compatibility library, the status_location parameter is of type union wait * rather than int *, as described above for the wait3() function.

AES Support Level: Full use (wait(), waitpid())

Return Values

If the wait(), waitpid(), or wait3() function returns because the status of a child process is available, the process ID of the child is returned to the calling process. If they return because a signal was caught by the calling process, -1 is returned and errno is set to [EINTR].

If the WNOHANG option was specified, and there are no stopped or exited child processes, the waitpid() and wait3() functions return a value of 0 (zero). Otherwise, -1 is returned and errno is set to indicate the error.

Errors

If the wait(), waitpid(), or wait3() function fails, errno may be set to one of the following values:

[ECHILD] The calling process has no existing unwaited-for child processes.

[EINTR] The function was terminated by receipt of a signal.

[EFAULT] The status_location or resource_usage parameter points to a location outside of the address space of the process.

The waitpid() function fails if one or both of the following are true:

[ECHILD] The process or process group ID specified by the process_id parameter does not exist or is not a child process of the calling process.
The `waitpid()` and `wait3()` functions fail if the following is true:

[EINVAL] The value of the `options` parameter is not valid.

Related Information

Functions: `exec(2)`, `exit(2)`, `fork(2)`, `pause(3)`, `ptrace(2)`, `getrusage(2)`, `sigaction(2)`
wcstombs

**Purpose**
Converts a wide character string into a multibyte character string

**Library**
Standard C Library (libc.a)

**Synopsis**

```c
#include <stdlib.h>
size_t wcstombs(
    char *s,
    const wchar_t *pwcs,
    size_t n);
```

**Parameters**

- **s**
  Points to the location where the converted multibyte character string is stored.

- **pwcs**
  Points to the wide-character string to be converted.

- **n**
  Specifies the number of bytes to be converted.

**Description**

The `wcstombs()` function converts a wide character string into a multibyte character string and stores the converted string in a location pointed to by the `s` parameter. The `wcstombs()` function stops storing characters supplied to the output array when it encounters a null character. The `wcstombs()` function only stores the number of bytes specified by the `n` parameter as the output string. When copying between objects that overlap, the behavior of `wcstombs()` is undefined.

The behavior of the `wcstombs()` function is affected by the LC_CTYPE category of the current locale. In environments that use shift-state dependent encoding, the array pointed to by `s` begins in its initial shift state.

Conversion terminates when the wide-character null is encountered or when the number of bytes expressed by the value of the `n` parameter (or `n-1`) has been stored at the location pointed to by the `s` parameter. When the amount of space available at the location pointed to by `s` only permits a partial multibyte character to be stored, `n-1` bytes are used because only valid (complete) multibyte characters are allowed.
Notes

AES Support Level: Full use

Return Values

When the `wcstombs()` function encounters a wide character code that does not correspond to a valid multibyte character, the value `(size_t)-1` is returned and `errno` is set to indicate the error. Otherwise, `wcstombs()` returns the number of bytes stored, not including a null terminator. (When the return value is `n`, the output array is not null terminated.)

Errors

If the `wcstombs()` fails, `errno` may be set to the following value:

- `[EINVAL]` The array pointed to by the `pwcs` parameter contains an entry that does not correspond with a valid multibyte character.

Related Information

Functions: `mblen(3), mbstowcs(3), mbtowc(3), wctomb(3)`
wctomb

Purpose
Converts a wide character into a multibyte character

Library
Standard C Library (libc.a)

Synopsis
#include <stdlib.h>

int wctomb(
    char *s,
    wchar_t wchar) ;

Parameters
s Points to the location where the conversion is stored.
wchar Specifies the wide character to be converted.

Description
The wctomb() function converts a wide character into a multibyte character. The wctomb() function stores no more than MB_CUR_MAX bytes.

The behavior of the wctomb() function is affected by the LC_CTYPE category of the current locale. In environments with shift-state dependent encoding, calls to the wctomb() function with the wchar parameter set to 0 (zero) put the function in its initial shift state. Subsequent calls with the wchar parameter set to nonzero values alter the state of the function as necessary. Changing the LC_CTYPE category of the locale causes the shift state of the function to be unreliable.

The implementation behaves as though no other function calls the wctomb() function.

Notes
AES Support Level: Full use
Return Values

When the s parameter is not a null pointer, the wctomb() function returns a value determined as follows:

- If the wchar parameter corresponds to a valid multibyte character, the wctomb() function returns the number of bytes in the multibyte character.
- If the wchar parameter does not correspond to a valid multibyte character, the wctomb() function returns -1 and sets errno to indicate the error.

When the s parameter is a null pointer, the return value depends on the environment in the following way:

- In environments where encoding is not state dependent, wctomb() returns 0 (zero).
- In environments where encoding is state dependent, wctomb() returns a nonzero value.

In no case is the returned value greater than the value of the MB_CUR_MAX macro.

Errors

If the wctomb() function fails, errno may be set to the following value:

[EINVAL] The wchar parameter does not correspond to a valid multibyte character.

Related Information

Functions: mblen(3), mbstowcs(3), mbtowc(3), wcstombs(3)
write, writev

Purpose

Writes to a file

Synopsis

```c
int write(
    int filedes ,
    const char *buffer,
    unsigned int nbytes );

#include <sys/types.h>
#include <sys/uio.h>

int writev(
    int filedes ,
    struct iovec *iov,
    int iov_count );
```

Parameters

- `filedes` Identifies the object to which the data is to be written.
- `buffer` Identifies the buffer containing the data to be written.
- `nbytes` Specifies the number of bytes to write.
- `iov` Points to an array of `iovec` structures, which identifies the buffers containing the data to be written. The `iovec` structure is defined in the `sys/uio.h` header file and contains the following members:
  ```c
  caddr_t iov_base;
  int iov_len;
  ```
- `iov_count` Specifies the number of `iovec` structures pointed to by the `iov` parameter.

Description

The `write()` function attempts to write `nbytes` of data to the file associated with the `filedes` parameter from the buffer pointed to by the `buffer` parameter.
If the nbyte parameter is 0 (zero), the write() function returns 0 (zero) and has no other results if the file is a regular file.

The writev() function performs the same action as the write() function, but gathers the output data from the iov_count buffers specified by the array of iovec structures pointed to by the iov parameter. Each iovec entry specifies the base address and length of an area in memory from which data should be written. The writev() function always writes a complete area before proceeding to the next.

The write() and writev() functions, which suspend the calling process until the request is completed, are redefined so that only the calling thread is suspended.

With regular files and devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. If this incremented file pointer is greater than the length of the file, the length of the file is set to this file offset. Upon return from the write() function, the file pointer increments by the number of bytes actually written.

With devices incapable of seeking, writing always takes place starting at the current position. The value of a file pointer associated with such a device is undefined.

Fewer bytes than requested can be written if there is not enough room to satisfy the request. In this case the number of bytes written is returned. The next attempt to write a nonzero number of bytes fails (except as noted in the following text). The limit reached can be either the ulimit() or the end of the physical medium. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes returns 20. The next write of a nonzero number of bytes will give a failure return (except as noted below).

Upon successful completion, the write() function returns the number of bytes actually written to the file associated with the fildes parameter. This number is never be greater than the nbyte parameter.

If the O_APPEND flag of the file status is set, the file offset is set to the end of the file prior to each write.

If the O_SYNC flag of the file status flags is set and the fildes parameter refers to a regular file, a successful write() function does not return until the data is delivered to the underlying hardware (as described in the open() function).

Write requests to a pipe (or FIFO) are handled the same as a regular file with the following exceptions:

- There is no file offset associated with a pipe; hence each write() request appends to the end of the pipe.
- If the size of the write() request is less than or equal to the value of the PIPE_BUF system variable, the write() function is guaranteed to be atomic. The data is not interleaved with data from other processes doing writes on the
same pipe. Writes of greater than PIPE_BUF bytes can have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not O_NONBLOCK or O_NDELAY are set.

- If neither O_NONBLOCK nor O_NDELAY are set, a write() request to a full pipe causes the process to block until enough space becomes available to handle the entire request.

- If the O_NONBLOCK or O_NDELAY flag is set, write() requests are handled differently in the following ways: the write() function does block the process; write() requests for PIPE_BUF or fewer bytes either succeed completely and return nbyte, or return -1 and set errno to [EAGAIN]. A write() request for greater than PIPE_BUF bytes either transfers what it can and returns the number of bytes written, or transfers no data and returns -1 with errno set to [EAGAIN]. Also, if a request is greater than PIPE_BUF bytes and all data previously written to the pipe has been read, write() transfers at least PIPE_BUF bytes.

When attempting to write to a regular file with enforcement mode record locking enabled, and all or part of the region to be written is currently locked by another process:

- If O_NDELAY and O_NONBLOCK are clear (the default), the calling process blocks until all the blocking locks are removed, or the write() function is terminated by a signal.

- If O_NDELAY or O_NONBLOCK is set, then the write() function returns -1 and sets errno to [EAGAIN].

Upon successful completion, the write() function marks the st_ctime and st_mtime fields of the file for update, and clears its set-user ID and set-group ID attributes if the file is a regular file.

The fcntl() function provides more information about record locks.

The behavior of an interrupted write() function depends on how the handler for the arriving signal was installed:

- If a write() function is interrupted by a signal before it writes any data, it returns -1 with errno set to [EINTR].

- If a write() function is interrupted by a signal after it successfully writes some data, it returns the number of bytes written. A write() request to a pipe or FIFO never returns with errno set to [EINTR] if it has transferred any data and nbyte is less than or equal to PIPE_BUF.
If the handler was installed with an indication that functions should not be restarted, the `write()` function returns a value of -1 and sets `errno` to [EINTR] (even if some data was already written).

If the handler was installed with an indication that functions should be restarted:

- If no data was written when the interrupt was handled, the `write()` function does not return a value (it is restarted).
- If data was written when the interrupt was handled, the `write()` function returns the amount of data already written.

Notes

AES Support Level: Full use (`write()`)

Return Values

Upon successful completion, the `write()` and `writev()` functions return the number of bytes that were actually written. Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors

If the `write()` or `writev()` function fails, `errno` may be set to one of the following values:

- **[EBADF]** The `filedes` parameter does not specify a valid file descriptor open for writing.
- **[EINVAL]** The file position pointer associated with the `filedes` parameter was negative.
- **[EINVAL]** The `iov_count` parameter value was not between 1 and 16, inclusive.
- **[EINVAL]** One of the `iov_len` values in the `iov` array was negative or the sum overflowed a 32-bit integer.
- **[EFAULT]** The `buffer` parameter or part of the `iov` parameter points to a location outside of the allocated address space of the process.
- **[EPIPE]** An attempt was made to write to a pipe or FIFO that is not opened for reading by any process. A SIGPIPE signal is sent to the process.
- **[EPERM]** An attempt was made to write to a socket or type `SOCK_STREAM` that is not connected to a peer socket.
write(2)

[EAGAIN] The O_NONBLOCK flag is set on this file and the process would be delayed in the write operation.

[EAGAIN] An enforcement mode record lock is outstanding in the portion of the file that is to be written, and O_NDELAY or O_NONBLOCK is set.

[ENOLCK] Enforced record locking is enabled and LOCK_MAX regions are already locked in the system.

[EDEADLK] Enforced record locking is enabled, O_NDELAY is clear, and a deadlock condition is detected.

[EFBIG] An attempt was made to write a file that exceeds the maximum file size.

[ENOSPC] No free space is left on the file system containing the file.

[EINTR] A signal was caught during the write() operation, and the signal handler was installed with an indication that functions are not to be restarted.

[EIO] The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned.

Related Information

Functions: open(2), fcntl(2), fcntl(2), lseek(2), open(2), pipe(2), poll(2), select(2), ulimit(3)
wsprintf

Purpose
Prints formatted output

Library
Standard I/O Package (libc.a)

Synopsis
#include <stdio.h>

int wsprintf (wchar_t *string, char *format [, value, ... ]);

Parameters

string Specifies a wchar_t string.
format Specifies a character string that contains conversion specifications, each of which causes zero or more items to be fetched from the value parameter list. If there are not enough items for format in the value parameter list, the results are unpredictable. If more values remain after the entire format has been processed, they are ignored.
value Specifies the input to the format parameter.

Description
The wsprintf() function is provided when Japanese Language Support is installed on your system.

The wsprintf() function converts, formats, and stores its value parameters, under control of the format parameter, into consecutive wchar_t characters starting at the address specified by the string parameter. The wsprintf() function places a ' ' (null character) at the end; It is your responsibility to ensure that enough storage space is available to contain the formatted string. The field width unit is specified as the number of wchar_t characters.

The wsprintf() function is the same as the sprintf() function, except that the wsprintf function uses a wchar_t string.
wsprintf(3)

Return Values

Upon successful completion, the wsprintf() function returns the number of display characters in the output string rather than the number of bytes in the string. The wsprintf() function uses strings that can contain 2-byte wchars. The value returned by wsprintf does not include the final ‘ ’ character. If an output error occurs, a negative value is returned.

Related Information

Functions: conv(3), ecvt(3), printf(3), putc(3), putwc(3), scanf(3)
wsscanf

Purpose
Converting formatted input

Library
Standard I/O Package (libc.a)

Synopsis
#include <stdio.h>

int wsscanf (wchar_t *string,
             char *format [, pointer, ... ]);

Parameters

string Specifies a wchar_t string.

format Contains conversion specifications used to interpret the input. If there are insufficient arguments for the format parameter, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated as always but are otherwise ignored.

pointer Specifies where to store the interpreted data.

Description
The wsscanf() function is provided when Japanese Language Support is installed on your system.

The wsscanf() function reads character data, interprets it according to a format, and stores the converted results into specified memory locations. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored.

This function is the same as the scanf() function, except that the wsscanf() function reads its input from the wchar_t string specified by the string parameter.
Return Values

The `wsscanf()` function returns the number of successfully matched and assigned input items. This number can be 0 (zero) if there was an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, only EOF is returned.

Related Information

Functions: `atof(3)`, `atoi(3)`, `getc(3)`, `getwc(3)`, `printf(3)`, `scanf(3)`
**Functions**

**wstring(3)**

- wstrcat, wstrchr, wstrcmp, wstrcpy, wstrcspn,
  wstrdup, wstrlen, wstrncat, wstrnclmp,
  wstrncpy, wstrpbrk, wstrrcchr, wstrspn,
  wstrtok

**Purpose**
Performs operations on wide character strings

**Library**
Standard C Library (libc.a)

**Synopsis**

```c
#include <wstring.h>

wchar_t *wstrcat (wchar_t *wsl, wchar_t *ws2);
wchar_t *wstrncat (wchar_t *wsl, wchar_t *ws2, int n);

int wstrcmp (wchar_t *wsl, wchar_t *ws2);

int wstrncpy (wchar_t *wsl, wchar_t *ws2, int n);

size_t wstrlen (wchar_t *ws);
wchar_t *wstrchr (wchar_t *ws, int n);
```

1-941
wstring(3)

```c
wchar_t *wstrrrchr (    
    wchar_t *ws,    
    int n );

wchar_t *wstrpbrk (    
    wchar_t *ws1,    
    wchar_t ws2;    

size_t wstrspn (    
    wchar_t *ws1,    
    wchar_t ws2 );    

size_t wstrencspn (    
    wchar_t *ws1,    
    wchar_t ws2 );    

wchar_t *wstrtok (    
    wchar_t *ws1,    
    wchar_t ws2 );    

wchar_t *wstrdup (    
    wchar_t *ws1 );
```

**Parameters**

`ws, ws1, ws2` Pointers to strings of type `wchar_t` (arrays of wide characters terminated by a `wchar_t` null character).

**Description**

The `wstring` functions copy, compare, and append strings in memory, and determine such things as location, size, and existence of strings in memory. For these functions, a string is an array of wide characters, terminated by a null character. The `wstring` functions parallel the `string` functions, but operate on strings of type `wchar_t` rather than on type `char`, except as noted below.

These functions require their parameters to be explicitly converted to type `wchar_t`, so they should be used on input that will be scanned many times for each time it is converted.

The `ws1, ws2, and ws` parameters point to strings of type `wchar_t`.

The `wstrcat()`, `wstrcat()`, `wstrcpy()`, and `wstrncpy()` functions all alter the `ws1` parameter. They do not check for overflow of the array pointed to by `ws1`. All string movement is performed character by character and starts at the left. Overlapping moves toward the left work as expected, but overlapping moves toward the right may give unexpected results. All of these functions are declared in the `wstring.h` header file.
The `wstrcat()` function appends a copy of the `ws2` string to the end of the `ws1` string. The `wstrcat()` function returns a pointer to the null-terminated result.

The `wstrcat()` function copies, at most, `n wchar_t`s in the `ws2` parameter to the end of the string pointed to by the `ws1` parameter. Copying stops before `n wchar_t`s if a null character is encountered in the string pointed to by the `ws2` parameter. The `wstrcat()` function returns a pointer to the null-terminated result.

The `wstrncat()` function copies, at most, `n wchar_t`s in the `ws2` parameter to the end of the string pointed to by the `ws1` parameter. Copying stops before `n wchar_t`s if a null character is encountered in the string pointed to by the `ws2` parameter. The `wstrncat()` function returns a pointer to the null-terminated result.

The `wstrncpy()` function copies the `ws2` string to the `ws1` string. Copying stops when the `wchar_t` null character is copied. The `wstrncpy()` function returns the value of the `ws1` parameter.

The `wstrncpy()` function copies the value of the `n` parameter of `wchar_t`s from the `ws2` string to the `ws1` string. If `ws2` is less than `n wchar_t`s long, then `wstrncpy()` pads `ws1` with trailing null characters to fill `n wchar_t`s. If `ws2` is `n` or more `wchar_t`s long, then only the first `n wchar_t`s are copied; the result is not terminated with a null character. The `wstrncpy()` function returns the value of the `ws1` parameter.

The `wstrlen()` function returns the number of `wchar_t`s in the string pointed to by the `ws` parameter, not including the terminating `wchar_t` null character.

The `wstrchr()` function returns a pointer to the first occurrence of the `wchar_t` specified by the `n` parameter in the `ws` string. A null pointer is returned if the `wchar_t` specified by `n` does not occur in the `ws` string. The `wchar_t` null character that terminates a string is considered to be part of the `ws` string.

The `w strrchr()` function returns a pointer to the last occurrence of the `wchar_t` specified by the `n` parameter in the `ws` string. A null pointer is returned if the `wchar_t` does not occur in the `ws` string. The `wchar_t` null character that terminates a string is considered to be part of the `wchar_t` string.

The `wstrpbrk()` function returns a pointer to the first occurrence in the `ws1` string of any code point from the `ws2` string. A null pointer is returned if no character matches.

The `wstrspn()` function returns the length of the initial segment of the `ws1` string that consists entirely of code points from the `ws2` string.
The \texttt{wstrcspn()} function returns the length of the initial segment of the \texttt{ws}1 string that consists entirely of code points \textit{not} from the \texttt{ws}2 string.

The \texttt{wstrtok()} function returns a pointer to an occurrence of a text token in the \texttt{ws}1 string. The \texttt{ws}2 parameter specifies a set of code points as token delimiters. If the \texttt{ws}1 parameter is anything other than null, then the \texttt{wstrtok()} function reads the string pointed to by the \texttt{ws}1 parameter until it finds one of the delimiter code points specified by the \texttt{ws}2 parameter. It then stores a \texttt{wchar\_t} null character in the \texttt{wchar\_t} string, replacing the delimiter code point, and returns a pointer to the first \texttt{wchar\_t} of the text token. The \texttt{wstrtok()} function keeps track of its position in the \texttt{wchar\_t} string so that subsequent calls with a null \texttt{ws}1 parameter step through the \texttt{wchar\_t} string. The delimiters specified by the \texttt{ws}2 parameter can be changed for subsequent calls to \texttt{wstrtok()}. When no tokens remain in the \texttt{wchar\_t} string pointed to by the \texttt{ws}1 parameter, the \texttt{wstrtok()} function returns a null pointer.

The \texttt{wstrdup()} function returns a pointer to a \texttt{wchar\_t} string that is a duplicate of the \texttt{wchar\_t} string to which the \texttt{ws}1 parameter points. Space for the new string is allocated using the \texttt{malloc()} function. When a new string cannot be created, a null pointer is returned.

\section*{Related Information}

Functions: \texttt{malloc(3)}, \texttt{string(3)}
Chapter 2

Files

This chapter contains reference pages for OSF/1 files. The reference pages from the \texttt{man4} and \texttt{man7} directories are sorted alphabetically in this chapter.
ar

Purpose Archive (library) file format

Synopsis #include <ar.h>

Description

The ar archive command combines several files into one. Archives are used mainly as libraries to be searched by the ld link editor.

A file produced by ar has a magic string at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout are described in the ar.h include file.

Each file begins on an even boundary. A newline character is inserted between files if necessary; nevertheless, the size given reflects the actual size of the file exclusive of padding.

There is no provision for empty areas in an archive file.

The encoding of the header is portable across machines. If an archive contains printable files, the archive itself is printable.

Related Information

Commands: ar(1), ld(1), nm(1)
core

Purpose  Specifies the format of the memory image file

Synopsis  #include <sys/param.h>

Description
The OSF/1 kernel writes out a memory image of a terminated process when an
error occurs. The most common errors are memory violations, illegal instructions,
bus errors, and user-generated quit signals. The memory image is called core and
is written in the process' working directory (provided it can be; normal access
controls apply).

The maximum size of a core file is limited by the setrlimit( ) function. Files which
would be larger than the limit are not created. Using the C shell, the core file size
can be limited with the limit command.

The core file consists of pertinent information about the stat of the process that
terminated. The exact layout of the core file is machine dependent, and is not
described here.

In general, the user debuggers are sufficient to deal with core images.

Related Information
Commands: csh(1)
Functions: sigvec(2), setrlimit(2)
ctab

Purpose
Locale character classification, case conversion, and collating input file

Description
A locale character classification, case conversion and collating input file consists of records separated by newline characters. Each record consists of one character or collation element in the locale, where a collation element is a sequence of two or more characters that collate as a single unit. These files are not directly accessed by user programs: the ctab command reads them to produce binary files loaded by the setlocale() function.

The ordering of the records determines the order of the locale’s characters. Records marked with the translate or ignore indicator (see KEYWORDS) do not reflect this ordering. The ordering of characters in a locale may also be referred to as their collation weights.

Several characters may have the same primary collation weights but different secondary weights. In French, the plain and accented versions of a’s all sort to the same primary location. If there is a tie between a plain and accented character, however, a secondary sort is applied. A group of characters with the same primary collation value are said to belong to the same equivalence class. If a character is not part of an equivalence class, it has identical primary and secondary collation weights.

This primary and secondary collation weight information is used in applications, such as grep, which use ctab information to determine string sequence.

The ctab input file describes the collating weights for an assumed code set and a particular language. If a character is encountered which does not appear in the ctab file corresponding to the current locale, the character’s collation weight will be based on its relative position in the current code set.

Records in the locale ctab input files have fields separated by a separator character (By default, this separator is a : (colon), but the user can change this; see KEYWORDS). The records have the following fields:

subject character
The subject character field is actually the collating element, which may be comprised of more than one character. If the subject character is a multicharacter collating element, the first character in the element must also be defined as a subject character elsewhere in the input file. If the character or collating element is followed by the equivalence class character, which is a ^ (circumflex) by default, it is given the same primary collating weight as the character.
represented by the preceding record. The secondary collation weight is unique. Characters can be specified using octal escape sequences consisting of a \ (backslash) followed by one or more octal digits. Any backslash not followed by an octal digit is an escape character. The subject character field must be terminated by a separator character even if there are no other fields in the record.

case conversion
The case conversion field specifies the character that is the inverse case of the character in the first field. For example, if the first field is p, the second field is P. If the third field, the character classification field (see below), contains an l or L (for lowercase), the second field specifies the uppercase equivalent of the subject character. If the character classification field contains a u or U (for uppercase), the case conversion field specifies the lowercase equivalent of the subject character. Any character with a nonempty case conversion field can specify the corresponding uppercase or lowercase letter. Characters classified as alphabetic do not require a corresponding case; that is, the second field can be empty. The second field currently is not used for SJIS characters when Japanese Language Support is installed.

character classification
The character classification field values assume the following classes and values:

- **u** or **U**: Uppercase letter
- **l** or **L**: Lowercase letter
- **a** or **A**: Alphabetic character
- **n** or **N**: Digits
- **x** or **X**: Hexadecimal digits
- **p** or **P**: Punctuation characters
- **s** or **S**: Whitespace characters
- **c** or **C**: Control characters
- **g** or **G**: Graphic
- **-**: No type
Characters can belong to more than one character class, subject to certain rules. The difference between graphic and printable characters is that the set of graphic characters does not include the space character, but the set of printable characters does include the space character. The ASCII code set is predefined as follows:

- A through Z (Uppercase letters)
- a through z (Lowercase letters)
- A through Z, and a through z (Alphabetic characters)
- 0 through 9 (Digits)
- Alphabetic characters and digits (Alphanumeric characters)
- 0 through 9, A through F, and a through f (Hexadecimal digits)

Any character below the Space character and the Delete character (Control characters)

- Space, formfeed, newline, carriage-return, horizontal tab, and vertical tab (Whitespace characters)

Any character except the above (Punctuation characters)

Characters not defined as alphabetic are automatically defined as punctuation.

Keywords

A line beginning with the word "option" serves to change one or more of the default conditions or metacharacters built into the collating table. The word "option" is followed by one or more keyword/value pairs. Keywords and values are separated by tab or space characters. The following keywords are recognized:

- **comment**
  
  Uses the assigned value as the comment character. The default value is the # (number sign). Anything on a line that follows the comment character is ignored.

- **sep**
  
  Uses the assigned value as the field separator character. The default value is a : (colon). Tabs or spaces can surround fields or separators.
ignore

Uses the assigned value as the ignore character indicator. The default value is the @ (at sign). A character marked with the ignore indicator is ignored for collation purposes.

repeat

Uses the assigned value as the equivalence class indicator. The default value is the ^ (circumflex) character. A character marked with the equivalence class indicator has the same primary collation value as the preceding character.

trans

Uses the assigned character as the translate indicator. The default value is the | (vertical bar). A collation element marked with the translate indicator is translated to the collation element(s) following the indicator. For example, to treat the German eszet (ß) element as the two characters ss, the first field of the line would be:

\337|ss:

The unique collation weight is used in regular expressions (see grep). Characters being translated cannot be followed by an equivalence character. The subject character cannot be contained in its own substitution collation element(s) (not o|oe). The translation mechanism completes in one pass: none of the characters in the substitution collation elements can in turn be the subject of further translation, so the following example is illegal:

q|r:
x|pq:

Characters being translated have no primary collating weight of their own, but have a unique collation weight, which is based on the order of the input line of the input file.

Examples

The following line is interpreted as a field containing a backslash and a colon followed by a field separator:

\\::

Files
ctab(4)
Here are the first and last three lines of a sample C.ctab file:

\000:
\001:
\002:

}:
~:
\177::c

Files

/usr/lib/nls/loc/<locale>
    Binary character classification, case conversion and collating output file for locale <locale>.

/etc/nls/loc/<locale>
    Binary locale classification, case conversion and collating output file. This is only used as a default during single-user mode operation.

Related Information

Commands: ctab(1)
Functions: setlocale(3)
"Using Internationalization Features" in the OSF/1 User's Guide
dir

**Purpose**  Format of directories

**Synopsis**  
#include <sys/types.h>
#include <dirent.h>

**Description**
A directory behaves like an ordinary file except that no user may write into a
directory. The fact that a file is a directory is indicated by a bit in the flag word of
its inode entry; see the fs reference page.

The POSIX standard way of returning directory entries is in directory entry
structures, which are of variable length. Each directory entry has a struct direct at
the front of it, containing its inode number, the length of the entry, and the length
of the name contained in the entry. These are followed by the name padded to a
4-byte boundary with null bytes. All names are guaranteed null terminated. The
maximum length of a name in a directory is _D_NAME_MAX.

By convention, the first two entries in each directory are for . (dot) and .. (dot-dot).
The first is an entry for the directory itself. The second is for the parent directory.
The meaning of .. (dot-dot) is modified for the root directory (/) of the master file
system, where .. (dot-dot) has the same meaning as . (dot).

**Related Information**
Functions: opendir(3)
Files: fs(4)
Purpose
Disk pack label

Synopsis
#include <sys/disklabel.h>

Description
Each disk or disk pack on a system may contain a disk label which provides
detailed information about the geometry of the disk and the partitions into which
the disk is divided. It should be initialized when the disk is formatted, and may be
changed later with the disklabel program. This information is used by the system
disk driver and by the bootstrap program to determine how to program the drive
and where to find the file systems on the disk partitions. Additional information is
used by the file system in order to use the disk most efficiently and to locate
important file system information. The description of each partition contains an
identifier for the partition type (standard file system, swap area, etc.). The file
system updates the in-core copy of the label if it contains incomplete information
about the file system.

The label is located in sector number LABELSECTOR of the drive, usually sector 0
(zero) where it may be found without any information about the disk geometry. It
is at an offset LABELOFFSET from the beginning of the sector, to allow room for
the initial bootstrap. The disk sector containing the label is normally made read-
only so that it is not accidentally overwritten by pack-to-pack copies or swap
operations; the DIOCWLABEL ioctl, which is done as needed by the disklabel
program, allows modification of the label sector.

A copy of the in-core label for a disk can be obtained with the DIOCGDINFO
ioctl; this works with a file descriptor for a block or character (raw) device for any
partition of the disk. The in-core copy of the label is set by the DIOCSDINFO
ioctl. The offset of a partition cannot generally be changed, nor made smaller
while it is open. One exception is that any change is allowed if no label was found
on the disk, and the driver was able to construct only a skeletal label without
partition information. Finally, the DIOCWDINFO ioctl operation sets the in-core
label and then updates the on-disk label; there must be an existing label on the disk
for this operation to succeed. Thus, the initial label for a disk or disk pack must be
installed by writing to the raw disk. All of these operations are normally done
using the disklabel program.
Related Information

Files: `disktab(4)`

Commands: `disklabel(8)`
disktab

**Purpose**
Disk description file

**Synopsis**
#include <disktab.h>

**Description**
The `disktab` database describes disk geometries and disk partition characteristics. It is used to initialize the disk label on the disk. The format is patterned after the `termcap` terminal database. Entries in a `disktab` file consist of a number of : (colon) separated fields. The first entry for each disk gives the names which are known for the disk, separated by | (vertical bar) characters. The last name given should be a long name fully identifying the disk.

The following list indicates the normal values stored for each disk entry:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ty</td>
<td>str</td>
<td>Type of disk (e.g. removable, winchester)</td>
</tr>
<tr>
<td>dt</td>
<td>str</td>
<td>Type of controller (e.g. SMD, ESDI, floppy)</td>
</tr>
<tr>
<td>ns</td>
<td>num</td>
<td>Number of sectors per track</td>
</tr>
<tr>
<td>nt</td>
<td>num</td>
<td>Number of tracks per cylinder</td>
</tr>
<tr>
<td>nc</td>
<td>num</td>
<td>Total number of cylinders on the disk</td>
</tr>
<tr>
<td>sc</td>
<td>num</td>
<td>Number of sectors per cylinder, nc*nt default</td>
</tr>
<tr>
<td>su</td>
<td>num</td>
<td>Number of sectors per unit, sc*nc default</td>
</tr>
<tr>
<td>se</td>
<td>num</td>
<td>Sector size in bytes, DEV_BSIZE default</td>
</tr>
<tr>
<td>sf</td>
<td>bool</td>
<td>Controller supports bad144-style bad sector forwarding</td>
</tr>
<tr>
<td>rm</td>
<td>num</td>
<td>Rotation speed, rpm, default 3600</td>
</tr>
<tr>
<td>sk</td>
<td>num</td>
<td>Sector skew per track, default 0</td>
</tr>
<tr>
<td>cs</td>
<td>num</td>
<td>Sector skew per cylinder, default 0</td>
</tr>
<tr>
<td>hs</td>
<td>num</td>
<td>Headswitch time, usec, default 0</td>
</tr>
<tr>
<td>ts</td>
<td>num</td>
<td>One-cylinder seek time, usec, default 0</td>
</tr>
<tr>
<td>il</td>
<td>num</td>
<td>Sector interleave (n:1), default 1</td>
</tr>
<tr>
<td>d[0-4]</td>
<td>num</td>
<td>Drive-type-dependent parameters</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>bs</td>
<td>num</td>
<td>Boot block size, default BBSIZE</td>
</tr>
<tr>
<td>sb</td>
<td>num</td>
<td>Superblock size, default SBSIZE</td>
</tr>
<tr>
<td>ba</td>
<td>num</td>
<td>Block size for partition ‘a’ (bytes)</td>
</tr>
<tr>
<td>bd</td>
<td>num</td>
<td>Block size for partition ‘d’ (bytes)</td>
</tr>
<tr>
<td>be</td>
<td>num</td>
<td>Block size for partition ‘e’ (bytes)</td>
</tr>
<tr>
<td>bf</td>
<td>num</td>
<td>Block size for partition ‘f’ (bytes)</td>
</tr>
<tr>
<td>bg</td>
<td>num</td>
<td>Block size for partition ‘g’ (bytes)</td>
</tr>
<tr>
<td>bh</td>
<td>num</td>
<td>Block size for partition ‘h’ (bytes)</td>
</tr>
<tr>
<td>fa</td>
<td>num</td>
<td>Fragment size for partition ‘a’ (bytes)</td>
</tr>
<tr>
<td>fd</td>
<td>num</td>
<td>Fragment size for partition ‘d’ (bytes)</td>
</tr>
<tr>
<td>fe</td>
<td>num</td>
<td>Fragment size for partition ‘e’ (bytes)</td>
</tr>
<tr>
<td>ff</td>
<td>num</td>
<td>Fragment size for partition ‘f’ (bytes)</td>
</tr>
<tr>
<td>fg</td>
<td>num</td>
<td>Fragment size for partition ‘g’ (bytes)</td>
</tr>
<tr>
<td>fh</td>
<td>num</td>
<td>Fragment size for partition ‘h’ (bytes)</td>
</tr>
<tr>
<td>oa</td>
<td>num</td>
<td>Offset of partition ‘a’ in sectors</td>
</tr>
<tr>
<td>ob</td>
<td>num</td>
<td>Offset of partition ‘b’ in sectors</td>
</tr>
<tr>
<td>oc</td>
<td>num</td>
<td>Offset of partition ‘c’ in sectors</td>
</tr>
<tr>
<td>od</td>
<td>num</td>
<td>Offset of partition ‘d’ in sectors</td>
</tr>
<tr>
<td>oe</td>
<td>num</td>
<td>Offset of partition ‘e’ in sectors</td>
</tr>
<tr>
<td>of</td>
<td>num</td>
<td>Offset of partition ‘f’ in sectors</td>
</tr>
<tr>
<td>og</td>
<td>num</td>
<td>Offset of partition ‘g’ in sectors</td>
</tr>
<tr>
<td>oh</td>
<td>num</td>
<td>Offset of partition ‘h’ in sectors</td>
</tr>
<tr>
<td>pa</td>
<td>num</td>
<td>Size of partition ‘a’ in sectors</td>
</tr>
<tr>
<td>pb</td>
<td>num</td>
<td>Size of partition ‘b’ in sectors</td>
</tr>
<tr>
<td>pc</td>
<td>num</td>
<td>Size of partition ‘c’ in sectors</td>
</tr>
<tr>
<td>pd</td>
<td>num</td>
<td>Size of partition ‘d’ in sectors</td>
</tr>
<tr>
<td>pe</td>
<td>num</td>
<td>Size of partition ‘e’ in sectors</td>
</tr>
<tr>
<td>pf</td>
<td>num</td>
<td>Size of partition ‘f’ in sectors</td>
</tr>
<tr>
<td>pg</td>
<td>num</td>
<td>Size of partition ‘g’ in sectors</td>
</tr>
<tr>
<td>ph</td>
<td>num</td>
<td>Size of partition ‘h’ in sectors</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>ta</td>
<td>str</td>
<td>Partition type of partition ‘a’ (4.2BSD file system, swap, etc.)</td>
</tr>
<tr>
<td>tb</td>
<td>str</td>
<td>Partition type of partition ‘b’</td>
</tr>
<tr>
<td>tc</td>
<td>str</td>
<td>Partition type of partition ‘c’</td>
</tr>
<tr>
<td>td</td>
<td>str</td>
<td>Partition type of partition ‘d’</td>
</tr>
<tr>
<td>te</td>
<td>str</td>
<td>Partition type of partition ‘e’</td>
</tr>
<tr>
<td>tf</td>
<td>str</td>
<td>Partition type of partition ‘f’</td>
</tr>
<tr>
<td>tg</td>
<td>str</td>
<td>Partition type of partition ‘g’</td>
</tr>
<tr>
<td>th</td>
<td>str</td>
<td>Partition type of partition ‘h’</td>
</tr>
</tbody>
</table>

**Example**

The following is an example `disktab` entry:
```
rz22|RZ22|DEC RZ22 Winchester: \
   :ty=winchester:dt=SCSI:ns#33:nt#4:nc#776: \
   :pa#32768:ba#8192:fa#1024: \
   :pb#69664:bb#8192:fb#1024: \
   :pc#102432:bc#8192:fc#1024:
```

**Files**

`/etc/disktab`

**Related Information**

Functions: `getdiskbyname(3)`

Files: `disklabel(4)`

Commands: `disklabel(8), newfs(8)`
Purpose  Locale country convention tables

Description
A locale country convention table consists of newline-separated records of the form name=value, where name is the name of a variable and value is its value. These name=value pairs specify configuration information and tailor input and output forms of dates, times, and monetary sums according to national or local requirements. If a symbolic constant value contains spaces, the value appears in quotes. Spaces cannot separate the equal sign from the variable or value which follows it.

Variables

AMPMSTR  AM/PM string with two colon-separated fields for suffixes to AM and PM time strings (for example, AMPMSTR=AM:PM).
CUR_SYM  The currency symbol (for example, CUR_SYM=$).
DEC_PNT  The radix character, or character that separates whole and fractional quantities (for example, DEC_PNT=,).
FRAC_DIG  The number of fractional digits.
GROUPING  The number of digits in each group separated by the THOUS_SEP character.
INT_CUR_SYM  International currency symbol.
INT_FRAC  International currency fraction digits.
MON_DEC_PNT  Currency radix character (for example, MON_DEC_PNT=,).
MON_GRP  The number of digits in each group separated by the MON_THOUS character in currency digits.
MON_THOUS  Currency thousands separator.
NEG_SGN  Currency minus sign.
NLDATE, NLSDATE, NLLDATE, NLDATIM, NLTIME
These variables are conversion specifications for date, short form of
date, long form of date, date and time, and time strings, respectively.
Their values cannot begin with an asterisk. Consult the strftime() function for a description of valid conversion specification
elements.

NLLDAY The full (long) names for the days of the week, beginning with
Sunday.

NLLMONTH The full (long) names of the months of the year, beginning with
January.

NLSDAY The short names of the days of the week. Names should be the same
length, and contain five or fewer characters. The specification starts
with the short name for Sunday.

NLSMONTH The short names of the months of the year. Names should be the
same length and contain five or fewer characters. The specification
starts with the short name for January.

NLTIME The conversion specification for the format of the time.

NLTMISC Miscellaneous strings needed for input of date and time
specifications. The default value is:

\texttt{at:each:every:on:through:am:pm:zulu}.

NLTSTRS The relative or informal names needed for input of date and time
specifications to the \texttt{remind} and \texttt{at} commands. The default value is:


NLTUNITS The singular and plural forms for all names of units of time, used for
input of date specifications to the \texttt{at} command. The default value is:

\texttt{minute:minutes:hour:hours:day:days:week:weeks:month:months:year:years:min:mins}.

NLYEAR The specification of eras to be used for display of the year relative
to the Japanese emperor calendar. It consists of colon-separated
elements of the format YYYYMMDD, name, where YYYY represents
the starting year of the era (year 1) and MMDD represents the
month and day, and name is the name of the era. If more than one
element is given, they must be in reverse order. Years B.C. must be
specified with a leading minus sign. There is no default. Example:
\texttt{NLYEAR=19890108,Heisei:19261225,Showa:}.

NOSTR The allowed forms for negative responses. A leading or trailing
colon, or two adjacent colons, indicate a null response. The order in
which the possible responses are listed has no significance.
N_CS_PRE Set to 1 if the currency symbol precedes the value for a negative formatted monetary quantity and to 0 (zero) if it succeeds the value.

N_SEP_SP Set to 1 if the currency symbol is separated by a space from the value for a negative formatted monetary quantity and to 0 (zero) if not.

N_SGN_POS Set to a value indicating the positioning of the positive sign for a negative formatted monetary quantity.

POS_SGN The positive sign.

P_CS_PRE Set to 1 if the currency symbol precedes the value for a nonnegative formatted monetary quantity and to 0 (zero) if it succeeds it.

P_SEP_SP Set to 1 if the currency symbol is separated by a space from the value for a nonnegative formatted monetary quantity and to 0 (zero) if not.

P_SGN_POS Set to a value indicating the positioning of the positive sign for a nonnegative formatted monetary quantity.

THOUS_SEP Separator for thousands place in decimal notation, where that place is determined by the GROUPING variable.

YESSTR The allowed forms for positive responses. A leading or trailing colon, or two adjacent colons, indicate a null response. The order in which the possible responses are listed has no significance.

Files

/usr/lib/nls/locale/<locale>.en
Country conversion file for locale <locale>.

/etc/nls/locale/<locale>.en
Locale country conversion file for locale <locale>. This file is only used as a default during single-user mode operation.

Related Information

Functions: strftime(3)

"Using Internationalization Features" in the OSF/1 User's Guide
exports

Purpose
Defines remote mount points for NFS mount requests

Description
The exports file specifies remote mount points for the NFS compatible mount protocol per the NFS server specification.

Each line in the file specifies one remote mount point. The first field is the mount point directory path followed optionally by export options and specific hosts separated by white space. Only the first entry for a given local file system may specify the export options, since these are handled on a "per local file system" basis. If no specific hosts are specified, the mount point is exported to all hosts.

The export options are as follows:

-root=<uid> Specifies how to map root's uid (default -2).
-r Synonymous with -root, in an effort to be backward compatible with older export file formats.
-ro Specifies that the file system should be exported read-only (default read/write).
-o Synonymous for -ro in an effort to be backward compatible with older export file formats.

Example
Given that /usr, /u and /u2 are local file system mount points, the following are valid entries in the /etc/exports file:

/usr -root=0 rickers snowwhite.cis.uoguelph.ca
/usr/local 131.104.48.16
/u -root=5
/u2 -ro

These entries specify that /usr is exported to hosts rickers and snowwhite.cis.uoguelph.ca with root mapped to root, /usr/local is exported to host 131.104.48.16 with root mapped to root, /u is exported to all hosts with root mapped to user ID 5, and /u2 is exported to all hosts read-only with root mapped to -2.

Note that /usr/local -root=5 would have been incorrect, since /usr and /usr/local reside in the same local file system.
Files

/etc/exports

Related Information

Commands: mountd(8), nfsd(8), showmount(8)
fd, stdin, stdout, stderr

**Purpose**  
File descriptors

**Description**  
The /dev/fd/0 through /dev/fd/# files refer to file descriptors which can be accessed through the file system. If the file descriptor is open and the mode the file is being opened with is a subset of the mode of the existing descriptor, the call:

```c
fd = open("/dev/fd/0", mode);
```

and the call:

```c
fd = fcntl(0, F_DUPFD, 0);
```

are equivalent.

Opening the /dev/stdin, /dev/stdout and /dev/stderr files is equivalent to the following calls:

```c
fd = fcntl(STDIN_FILENO, F_DUPFD, 0);
fd = fcntl(STDOUT_FILENO, F_DUPFD, 0);
fd = fcntl(STDERR_FILENO, F_DUPFD, 0);
```

Flags to the `open()` function other than O_RDONLY, O_WRONLY and O_RDWR are ignored.

**Files**

- **/dev/fd/#**  
  File descriptor files, where # (number sign) represents the file descriptor number.

- **/dev/stdin**  
  Special file for the standard input device.

- **/dev/stdout**  
  Special file for the standard output device.

- **/dev/stderr**  
  Special file for the standard error device.

**Related Information**

Files: tty(7)
fs, inode

**Purpose**  Specifies the format of the file system volume

**Synopsis**

```c
#include <sys/types.h>
#include <sys/fs.h>
#include <sys/inode.h>
```

**Description**

Every file system storage volume (disk, nine-track tape, for instance) has a common format for certain vital information. Each such volume is divided into a certain number of blocks. The block size is a parameter of the file system. Sectors beginning at BBLOCK and continuing for BBSIZE are used to contain a label and for some hardware primary and secondary bootstrapping programs.

Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A file system is described by its superblock, which in turn describes the cylinder groups. The superblock is critical data and is replicated in each cylinder group to protect against loss of data. This is done at file system creation time and the critical superblock data does not change, so the copies need not be referenced further until necessary.

Addresses stored in inodes are capable of addressing fragments of blocks. File system blocks of at most MAXBSIZE size can be optionally broken into 2, 4, or 8 pieces, each of which is addressable; these pieces may be DEV_BSIZE, or some multiple of a DEV_BSIZE unit.

Large files consist exclusively of large data blocks. To avoid wasted disk space, the last data block of a small file is allocated only as many fragments of a large block as are necessary. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determined from information in the inode, using the `blksize(fs, ip, lbn)` macro.

The file system records space availability at the fragment level; to determine block availability, aligned fragments are examined.

The root inode is the root of the file system. Inode 0 (zero) can’t be used for normal purposes and, historically, bad blocks were linked to inode 1. Thus, the root inode is 2 (inode 1 is no longer used for this purpose, but numerous dump tapes make this assumption).
Some fields to the fs structure are as follows:

**fs_minfree**  Gives the minimum acceptable percentage of file system blocks that may be free. If the freelist drops below this level only the superuser may continue to allocate blocks. The fs_minfree field may be set to 0 (zero) if no reserve of free blocks is deemed necessary. However, severe performance degradations will be observed if the file system is run at greater than 90% full; thus the default value of the fs_minfree field is 10%.

Empirically the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 8, thus the default fragment size is an eighth of the block size.

**fs_optim**  Specifies whether the file system should try to minimize the time spent allocating blocks, or if it should attempt to minimize the space fragmentation on the disk. If the value of fs_minfree is less than 10%, then the file system defaults to optimizing for space to avoid running out of full sized blocks. If the value of fs_minfree is greater than or equal to 10%, fragmentation is unlikely to be problematic, and the file system defaults to optimizing for time.

*Cylinder group related limits:* Each cylinder keeps track of the availability of blocks at different positions of rotation, so that sequential blocks can be laid out with minimum rotational latency. With the default of 8 distinguished rotational positions, the resolution of the summary information is 2 milliseconds for a typical 3600 rpm drive.

**fs_rotdelay**  Gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. The fs_rotdelay field is used in determining the rotationally optimal layout for disk blocks within a file; the default value for fs_rotdelay is 2 milliseconds.

Each file system has a statically allocated number of inodes. An inode is allocated for each NBPI bytes of disk space. The inode allocation strategy is extremely conservative.

MINBSIZE is the smallest allowable block size. With a MINBSIZE of 4096 it is possible to create files of size $2^{32}$ with only two levels of indirection. MINBSIZE must be big enough to hold a cylinder group block, thus changes to struct cg must keep its size within MINBSIZE. Note that superblocks are never more than size SBSIZE.
The pathname on which the file system is mounted is maintained in `fs_fsmnt`. `MAXMNTLEN` defines the amount of space allocated in the superblock for this name. The limit on the amount of summary information per file system is defined by `MAXCSBUFS`. For a 4096 byte block size, it is currently parameterized for a maximum of two million cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from `fs_csaddr` (size `fs_cssize`) in addition to the superblock.

**Superblock for a file system**: The size of the rotational layout tables is limited by the fact that the superblock is of size `SB SIZE`. The size of these tables is inversely proportional to the block size of the file system. The size of the tables is increased when sector sizes are not powers of two, as this increases the number of cylinders included before the rotational pattern repeats (`fs_cpc`). The size of the rotational layout tables is derived from the number of bytes remaining in `(struct fs)`.

The number of blocks of data per cylinder group is limited because cylinder groups are at most one block. The inode and free block tables must fit into a single block after deducting space for the cylinder group structure `struct cg`.

**Inode**: The inode is the focus of all file activity in the UNIX file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is ‘named’ by its device/i-number pair.

**Notes**

`sizeof (struct csum)` must be a power of two in order for the `fs_cs` macro to work.
**Purpose**  
Group file

**Description**  
The /etc/group database contains the following information for each group:
- Group name
- Encrypted password
- Numerical group ID
- A comma-separated list of all users allowed in the group

The /etc/group file is an ASCII file, with the fields separated by colons. Each group is separated from the next by a new line. If the password field is null, no password is demanded.

Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group IDs to names.

**Files**

/etc/group

**Related Information**

Functions: setgroups(2), initgroups(3), crypt(3)

Commands: passwd(1)

Files: passwd(5)
icmp

Purpose
Internet Control Message Protocol

Synopsis
```
#include <sys/socket.h>
#include <netinet/in.h>
s = socket(AF_INET, SOCK_RAW, proto);
```

Description
The Internet Control Message Protocol (ICMP) is the error and control message protocol used by the Internet Protocol (IP) and the Internet Protocol family. It may be accessed through a raw socket for network monitoring and diagnostic functions. The proto parameter to the socket call to create an ICMP socket is obtained from the `getprotobyname()` function. ICMP sockets are connectionless, and are normally used with the `sendto()` and `recvfrom()` functions. The `connect()` function may also be used to fix the destination for future packets, in which case the `read()` or `recv()` and `write()` or `send()` functions may be used.

Outgoing packets automatically have an IP header prepended to them (based on the destination address). Incoming packets are received with the IP header and options intact.

Errors
If a socket operation fails, `errno` may be set to one of the following values:

- `[EISCONN]` The socket is already connected. This error occurs when trying to establish connection on a socket or when trying to send a datagram with the destination address specified.

- `[ENOTCONN]` The destination address of a datagram was not specified, and the socket has not been connected.

- `[ENOBUFS]` The system ran out of memory for an internal data structure.

- `[EADDRNOTAVAIL]` An attempt was made to create a socket with a network address for which no network interface exists.
Related Information

Functions: send(2), recv(2)
Files: netintro(7), inet(7), ip(7)
idp

Purpose  Xerox Internet Datagram Protocol

Synopsis  
```
#include <sys/socket.h>
#include <netns/ns.h>
#include <netns/idp.h>

s = socket(AF_NS, SOCK_DGRAM, 0);
```

Description

The Xerox Internet Datagram Protocol (IDP) is a simple, unreliable datagram protocol which is used to support the SOCK_DGRAM abstraction for the Internet protocol family. IDP sockets are connectionless, and are normally used with the `sendto()` and `recvfrom()` functions. The `connect()` function may also be used to fix the destination for future packets, in which case the `recv()` or `read()` and `send()` or `write()` functions may be used.

Xerox protocols are built vertically on top of IDP. Thus, IDP address formats are identical to those used by the Sequenced Packet Protocol (SPP). Note that the IDP port space is the same as the SPP port space; that is, an IDP port may be connected to an SPP port, with certain options enabled as described below. In addition, broadcast packets may be sent (assuming the underlying network supports this) by using a reserved broadcast address; this address is network interface dependent.

The following socket options are available:

**SO_HEADERS_ON_INPUT**

When set, the first 30 bytes of any data returned from a `read()` or `recv()` function will be the initial 30 bytes of the IDP packet, as described by:

```
struct idp {
    u_short idp_sum;
    u_short idp_len;
    u_char idp_tc;
    u_char idp_pt;
    struct ns_addr idp_dna;
    struct ns_addr idp_sna;
};
```

This allows the user to determine the packet type, and whether the packet was a multicast packet or directed specifically at the local host. When requested, gives the current state of the option as either NSP_RAWIN or 0 (zero).
SO_HEADERS_ON_OUTPUT
When set, the first 30 bytes of any data sent will be the initial 30 bytes of the IDP packet. This allows the user to determine the packet type, and whether the packet should be a multicast packet or directed specifically at the local host. You can also misrepresent the sender of the packet. When requested, gives the current state of the option as either NSP_RAWOUT or 0 (zero).

SO_DEFAULT_HEADERS
The user provides the kernel an IDP header, from which it gleans the packet type. When requested, the kernel will provide an IDP header, showing the default packet type, and local and foreign addresses, if connected.

SO_ALL_PACKETS
When set, this option defeats automatic processing of error packets, and sequence protocol packets.

SO_SEQNO
When requested, this option returns a sequence number which is not likely to be repeated until the machine crashes or a very long time has passed. It is useful in constructing packet exchange protocol packets.

Errors
If a socket operation fails, errno may be set to one of the following values:

[EISCONN] The socket is already connected. This error occurs when trying to establish connection on a socket or when trying to send a datagram with the destination address specified.

[ENOTCONN] The destination address of a datagram was not specified, and the socket has not been connected.

[ENOBUFS] The system ran out of memory for an internal data structure.

[EADDRINUSE] An attempt was made to create a socket with a network address for which no network interface exists.

[EADDRNOTAVAIL] An attempt was made to create a socket with a network address for which no network interface exists.
Related Information

Functions: `send(2), recv(2)`

Files: `netintro(7), ns(7)`
inet

Purpose
Internet Protocol family

Synopsis
#include <sys/types.h>
#include <netinet/in.h>

Description
The Internet Protocol family is a collection of protocols layered atop the Internet Protocol (IP) transport layer, and utilizing the Internet address format. The Internet family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types; the SOCK_RAW interface provides access to the IP protocol.

Internet addresses are 4-byte quantities, stored in network standard format (on the VAX and other machines, these are word and byte reversed). The netinet/in.h include file defines this address as a discriminated union.

Sockets bound to the Internet protocol family utilize an addressing structure sockaddr_in, whose format is dependent on whether _SOCKADDR_LEN has been defined prior to including the netinet/in.h header file. If _SOCKADDR_LEN is defined, the sockaddr_in structure takes 4.4BSD behavior, with a separate field for specifying the length of the address; otherwise, the default 4.3BSD behavior is used.

Sockets may be created with the local address INADDR_ANY to effect wildcard matching on incoming messages. The address in a connect() or sendto() call may be given as INADDR_ANY to mean "this host." The distinguished address INADDR_BROADCAST is allowed as a shorthand for the broadcast address on the primary network if the first network configured supports broadcast.

The Internet protocol family is comprised of the IP transport protocol, Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). TCP is used to support the SOCK_STREAM abstraction while UDP is used to support the SOCK_DGRAM abstraction. A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The ICMP message protocol is accessible from a raw socket.

The 32-bit Internet address contains both network and host parts. It is frequency-encoded; the most-significant bit is clear in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses use the high-order 16 bits as the network field, and Class C addresses have a 24-bit network part. Sites with a cluster of local networks and a connection to the DARPA Internet may chose to use a single network number for the cluster; this is done by using subnet...
addressing. The local (host) portion of the address is further subdivided into subnet and host parts. Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry.

Subnet addressing is enabled and examined by the following \texttt{ioctl()} commands on a datagram socket in the Internet domain; they have the same form as the \texttt{SIOCIFADDR} command (see the reference page for the \texttt{netintro} function).

\texttt{SIOCSIFNETMASK}
   Set interface network mask. The network mask defines the network part of the address; if it contains more of the address than the address type would indicate, then subnets are in use.

\texttt{SIOCGIFNETMASK}
   Get interface network mask.

\textbf{Notes}

The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

\textbf{Related Information}

Functions: \texttt{ioctl(2), socket(2)}

Files: \texttt{netintro(7), tcp(7), udp(7), ip(7), icmp(7)}

\textit{OSF/1 Network Applications Programmer's Guide}

\textit{OSF/1 Network and Communications Administrator's Guide}

\textit{OSF/1 System and Network Administrator's Reference}
#ip

## Purpose
Internet Protocol

## Synopsis
```c
#include <sys/socket.h>
#include <netinet/in.h>
s = socket(AF_INET, SOCK_RAW, proto);
```

## Description
The Internet Protocol (IP) is the transport layer protocol used by the Internet Protocol family. Options may be set at the IP level when using higher-level protocols that are based on IP (such as the Transmission Control Protocol (TCP) and the User Datagram Package (UDP)). It may also be accessed through a raw socket when developing new protocols, or special purpose applications.

IP_OPTIONS is used to provide IP options to be transmitted in the IP header of each outgoing packet. Options are set with the `setsockopt()` function and examined with the `getsockopt()` function. The format of IP options to be sent is that specified by the IP specification, with one exception: the list of addresses for Source Route options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.

Raw IP sockets are connectionless, and are normally used with the `sendto()` and `recvfrom()` calls, though the `connect()` call may also be used to fix the destination for future packets, in which case the `read()` or `recv()` and `write()` or `send()` functions may be used.

If `proto` is 0 (zero), the default protocol IPPROTO_RAW is used for outgoing packets, and only incoming packets destined for that protocol are received. If `proto` is nonzero, that protocol number will be used on outgoing packets and to filter incoming packets.

Outgoing packets automatically have an IP header prepended to them (based on the destination address and the protocol number the socket is created with), unless the IP_HDRINCL option is set. IP_HDRINCL specifies whether the IP header is provided by the sent packet. Incoming packets are received with IP header and options intact.
Errors

If a socket operation fails, `errno` may be set to one of the following values:

[EISCONN] The socket is already connected. This error occurs when trying to establish connection on a socket or when trying to send a datagram with the destination address specified.

[ENOTCONN] The destination address of a datagram was not specified, and the socket has not been connected.

[ENOBUFF] The system ran out of memory for an internal data structure.

[EADDRNOT] An attempt was made to create a socket with a network address for which no network interface exists.

The following errors specific to IP may occur when setting or getting IP options:

[EINVAL] An unknown socket option name was given.

[EINVAL] The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

Related Information

Functions: `getsockopt(2), send(2), recv(2)`

Files: `netintro(7), icmp(7), inet(7)`
Purpose
Software loopback network interface

Synopsis
pseudo-device loop

Description
The loopback interface is a software loopback mechanism which is used for performance analysis, software testing, and/or local communication. As with other network interfaces, the loopback interface must have network addresses assigned for each address family with which it is to be used. These addresses may be set or changed with the SIOCSIFADDR ioctl. The loopback interface should be the last interface configured, as protocols may use the order of configuration as an indication of priority. The loopback should never be configured first unless no hardware interfaces exist.

Notes
Previous versions of the UNIX system enabled the loopback interface automatically, using a nonstandard Internet address (127.1). Use of that address is now discouraged; a reserved host address for the local network should be used instead.

Errors

lon: can't handle afn
The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

Related Information
Files: netintro(7), inet(7), ns(7)
lvm

Purpose  Logical Volume Manager (LVM) programming interface

Synopsis  #include <lvm/lvm.h>

Description
The Logical Volume Manager (LVM) implements virtual disks, called logical volumes, and uses physical disks, called physical volumes, to store the actual data. The programming interface to the LVM is provided through a number of LVM ioctl commands. These commands perform functions like creating logical and physical volumes, removing logical and physical volumes, and so on. Basically, there are four groupings of the LVM ioctl commands: those that deal with volume groups, those that deal with logical volumes, those that deal with physical volumes, and those that perform miscellaneous functions. The following table illustrates these groupings:

Volume Group
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVM_ACTIVATEVG</td>
<td>Activate volume group</td>
</tr>
<tr>
<td>LVM_CREATEVG</td>
<td>Create volume group</td>
</tr>
<tr>
<td>LVM_DEACTIVATEVG</td>
<td>Deactivate volume group</td>
</tr>
<tr>
<td>LVM_QUERYVG</td>
<td>Query volume group (retrieve information)</td>
</tr>
<tr>
<td>LVM_SETVGID</td>
<td>Set volume group ID</td>
</tr>
</tbody>
</table>

Logical Volume
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVM_CHANGELV</td>
<td>Change logical volume attributes</td>
</tr>
<tr>
<td>LVM_CREATELV</td>
<td>Create logical volume</td>
</tr>
<tr>
<td>LVM_DELETELV</td>
<td>Delete logical volume from volume group</td>
</tr>
<tr>
<td>LVM_EXTENDLV</td>
<td>Extend logical volume (adds extents)</td>
</tr>
<tr>
<td>LVM_QUERYLV</td>
<td>Query logical volume (retrieve information)</td>
</tr>
<tr>
<td>LVM_QUERYLVMAP</td>
<td>Query logical volume physical extent map</td>
</tr>
<tr>
<td>LVM_REALLOCLV</td>
<td>Move physical extents between logical volumes</td>
</tr>
<tr>
<td>LVM_REDUCELV</td>
<td>Reduce logical volume (reduce extents)</td>
</tr>
<tr>
<td>LVM_RESYNCLV</td>
<td>Resynchronize logical volume</td>
</tr>
</tbody>
</table>
**Physical Volume**

- **LVM_ATTACHpv**
  Attach physical volume to volume group
- **LVM_CHANGEpv**
  Change physical volume attributes
- **LVM_DELETEpv**
  Delete physical volume from volume group
- **LVM_INSTALLpv**
  Install physical volume to volume group
- **LVM_QUERYpv**
  Query physical volume (retrieve information)
- **LVM_QUERYpvMAP**
  Query map of physical extents on physical volume
- **LVM_QUERYpvPATH**
  Query physical volume using physical identifier as the pathname
- **LVM_QUERYpvS**
  Query multiple physical volumes (retrieve information)
- **LVM_REMOVEpv**
  Remove a physical volume from the volume group
- **LVM_RESYNCPv**
  Resynchronize physical volume

**Miscellaneous**

- **LVM_OPTIONGET**
  Obtain current raw device I/O options, as set by the LVM_OPTIONSET command
- **LVM_OPTIONSET**
  Set I/O options for the raw logical volume device
- **LVM_RESYNCLnx**
  Initiates resynchronization for physical extents

The following alphabetic listing of the LVM ioctl commands first gives synopses and descriptions of the LVM ioctl commands, then provides descriptions of the command parameters, and finally provides a list of returned errors.

For detailed information on the LVM, see *The Design of the OSF/1 Operating System*.

```c
ioctl(fd, LVM_ACTIVATEVG, &flags)
int fd;
int flags;
```

This command brings the specified volume group online. This involves reconciliation of the VGDA’s on all attached physical volumes, and recovery of active mirrors. Depending on whether the **LVM_ALL_PVS_REQUIRED** or **LVM_NONMISSING_PVS_REQUIRED** flags are set in flags, it may fail if some of the physical volumes in the volume group are missing (**LVM_ALL_PVS_REQUIRED** flag set).
This command attaches the specified physical volume to the specified volume group. This operation is analogous to a `mount()` command: the named device is opened, and the LVM maintains a reference to it. The `LVM_ATTACHPV` command reads the LVM record to determine the `vg_id` and the `pvnum`. This command fails if the volume is a member of another volume group.

This command changes the attributes of a logical volume in a specified volume group. It updates the specified logical volume's LVM data structures and logical volume entry in the descriptor area.

You can use this command on a logical volume device. In this case, the `minor_num` is ignored, and the command applies to that device.

This command changes the attributes of a physical volume. You can use `LVM_CHANGEPV` to change the maximum number of defects (maxdefects) that can be relocated on this physical volume. You can also use this command to disallow or re-allow allocation of extents on the physical volume. Allocation from a physical volume should be disallowed if the physical extents of that physical volume are to be migrated to another physical volume.
Note that if you change `maxdefects` to a number lower than what has already been relocated on the physical volume, `LVM_CHANGEPV` will reset `maxdefects` to the relocated number of defects.

```c
ioctl(fd, LVM_CREATELV, &lv_statuslv)
int fd;
struct lv_statuslv {
    ushort_t minor_num;
    ushort_t maxlxs;
    ushort_t lv_flags;
    ushort_t sched_strat;
    ushort_t maxmirrors;
} lv_statuslv;
```

This command creates a logical volume in a specified volume group. It uses the supplied information to update a previously unused entry in the logical volume list. The index into the list of logical volume entries corresponds to the minor number (`minor_num`) of the logical volume. `LVM_CREATELV` does not do extent allocation. The `LVM_EXTENDLV ioctl` must be used to allocate extents for the new logical volume.

```c
ioctl(fd, LVM_CREATEVG, &lv_createvg)
int fd;
struct lv_createvg {
    char *path;
    lv_uniqueID_t vg_id;
    ushort_t pv_flags;
    ushort_t maxlvs;
    ushort_t maxpvs;
    ushort_t maxpxs;
    ulong_t pxsize;
    ulong_t pxspace;
    ushort_t maxdefects;
} lv_createvg;
struct lv_uniqueID {
    ulong_t id1;
    ulong_t id2;
};
typedef struct lv_uniqueID lv_uniqueID_t;
```

This command creates a volume group and installs the first physical volume. It initializes the in-memory VGDA for the volume group.
ioctl(fd, LVM_DEACTIVATEVG, 0)

int fd;

This command takes a specified volume (fd) group offline. All logical volumes in this volume group must be closed. The argument (0) is ignored.

ioctl(fd, LVM_DELETELV, minor_num)

int fd;
int minor_num;

ioctl(fd, LVM_DELETEPV, pv_key)

int fd;
int pv_key;

This command deletes a physical volume from a specified volume group (fd). The physical volume must not contain any extents of a logical volume for it to be deleted. If the physical volume contains any extents of a logical volume, an error code is returned. In this case, you must delete logical volumes or relocate the extents that reside on this physical volume. For an empty physical volume, LVM_DELETEPV removes the entries for this physical volume from the LVM data structures and from the descriptor area, and initializes the descriptor area on the physical volume being deleted.

ioctl(fd, LVM_EXTENDLV, &lv_lvsize)

int fd;
struct lv_lvsize {
    ushort_t minor_num;
    ulong_t size;
    lxmap_t *extents;
} lv_lvsize;
struct lxmap {
    ushort_t lx_num;
    ushort_t pv_key;
    ushort_t px_num;
    ushort_t status;
};
typedef struct lxmap lxmap_t;

This command adds extents to a given logical volume. It allocates physical extents for the specified logical volume at the physical volume and physical extent specified as input via the extent list pointer. It updates the LVM data structures and the descriptor area.
ioctl(fd, LVM_INSTALLPV, &lv_installpv)

int fd;
struct lv_installpv {
    char *path;
    ulong_t pxspace;
    ushort_t pv_flags;
    ushort_t maxdefects;
} lv_installpv;

This command installs a physical volume into a specified volume group. To do this, LVM_INSTALLPV adds the physical volume specification to the in-memory VGDA for the volume group, and then updates all active physical volumes in the volume group. This command fails if the physical volume is already a member of another volume group.

ioctl(fd, LVM_OPTIONGET, &lv_option)

int fd;
struct lv_option lv_option;

ioctl(fd, LVM_OPTIONSET, &lv_option)

int fd;
struct lv_option {
    ushort_t opt_avoid;
    ushort_t opt_options;
} lv_option;

This command sets the I/O options for the raw logical volume device. The raw device is capable of avoiding specified mirrors on read operations, set through the opt_avoid field. This allows a program to access a specific copy of a mirrored logical volume.

The opt_options field allows the program to temporarily (until the device is closed) specify that all writes are to be verified (LVM_VERIFY) or that defect relocation is not to be performed (LVM_NORELOC). To set these options permanently, or for the block device, see LVM_CHANGELV. The raw I/O options are cleared when the raw device is first opened, and never have an effect on block device operations.

The LVM_OPTIONGET command obtains the current raw device I/O options, as set by the LVM_OPTIONSET command. These functions apply only to open devices, are only valid against the logical volume devices, not the control device.
This command obtains information about a particular logical volume from the specified volume group. It verifies that the logical volume is valid and returns the information requested for its volume group to the buffer supplied. You can use this command on a file descriptor corresponding to a logical volume device. In this case, the command ignores the minor_num field. Structure fields are output fields unless marked otherwise.

This command obtains information from the specified volume group about the space and extents allocated to a particular logical volume. It verifies that the logical volume is valid and returns the information requested for its volume group to the buffer supplied. The allocation map must be large enough to accommodate the extent map from the logical volume. This information is available from LVM_QUERYLV.

You can use this command on a file descriptor corresponding to a logical volume device. In this case, the minor_num field is ignored.
ioctl(fd, LVM_QUERYPV, &lv_querypv)
int fd;
struct lv_querypv {
    ushort_t pv_key;
    ushort_t pv_flags;
    ushort_t px_count;
    ushort_t px_free;
    ulong_t px_space;
    dev_t pv_rdev;
    ushort_t maxdefects;
    ushort_t bbpool_len;
} lv_querypv;

This command retrieves information about a specified physical volume. It verifies that the physical volume is valid and writes the requested information to the buffer supplied.

ioctl(fd, LVM_QUERYPVMAP, &lv_querypvmap)
int fd;
struct lv_querypvmap {
    ushort_t pv_key;
    ushort_t numpxs;
    pxmap_t *map;
} lv_querypvmap;
struct pxmap {
    ushort_t lv_minor;
    ushort_t lv_extent;
    ushort_t status;
};

typedef struct pxmap pxmap_t;

This command returns the map of physical extents on the specified physical volume. This mapping indicates the logical volume and logical extent to which each corresponds. A physical extent which is not currently assigned to a logical volume will be indicated by an lv_minor value of 0 (zero).
ioctl(fd, LVM_QUERYPVSPATH, &lv_querypvpath)
   int fd;
   struct lv_querypvpath {
      char  *path;
      ushort_t pv_key;
      ushort_t pv_flags;
      ushort_t px_count;
      ushort_t px_free;
      ulong_t px_space;
      dev_t  pv_rdev;
      ushort_t maxdefects;
      ushort_t bbpool_len;
   } lv_querypvpath;

This command is identical to LVM_QUERYPV, except that it takes a pathname (path) as the physical volume identifier. Also, it returns the pv_key rather than taking it as input.

ioctl(fd, LVM_QUERYPVXS, &lv_querypvxs)
   int fd;
   struct lv_querypvxs {
      ushort_t numpvs;
      ushort_t *pv_keys;
   } lv_querypvxs;

This command retrieves the physical volume list from the volume group. It requires the number of volumes in the volume group as input (as obtained from LVM_QUERYVG) and returns the pv_key for each.

ioctl(fd, LVM_QUERYVG, &lv_queryvg)
   int fd;
   struct lv_queryvg {
      lv_uniqueID_t vg_id;
      ushort_t maxlvs;
      ushort_t maxpvs;
      ushort_t maxpxs;
      ulong_t pxxsize;
      ushort_t freepxs;
      ushort_t cur_lvs;
      ushort_t cur_pvs;
      ushort_t status;
   } lv_queryvg;

This command retrieves information about a specified volume group. It verifies that the specified volume group is valid and writes the information requested to the buffer supplied.
ioctl(fd, LVM_REALLOCLV, &lv_realloclv)

```c
int fd;
struct lv_realloclv {
    ushort_t sourcelv;
    ushort_t destlv;
    ulong_t size;
    lxmap_t *extents;
} lv_realloclv;
```

This command atomically removes physical extents from one logical volume \((fd)\) and assigns them to another \((destlv)\). The logical extent number of each physical extent is preserved. If the destination logical volume already has space allocated for the indicated logical extents, the new extents will be marked as stale by the reallocation.

ioctl(fd, LVM_REDUCELV, &lv_lvsize)

```c
int fd;
struct lv_lvsize {
    ushort_t minor_num;
    ulong_t size;
    lxmap_t *extents;
} lv_lvsize;
```

```c
struct lxmap {
    ushort_t lx_num;
    ushort_t pv_key;
    ushort_t px_num;
    ushort_t status;
};
typedef struct lxmap lxmap_t;
```

This command removes extents from a specified logical volume. It deallocates a logical extent for the specified logical volume at the physical volume. The extents to be removed are specified as input via the extent list pointer \((*extents)\). It updates the LVM data structures and the descriptor area.

You can use **LVM_REDUCELV** on a file descriptor corresponding to a logical volume device. In this case, the \(minor\_num\) field is ignored.

ioctl(fd, LVM_REMOVEPV, &pv_key)

```c
int fd;
int pv_key;
```

This command temporarily removes a physical volume from the volume group by closing the physical volume device. If the volume
group is active, the physical volume state is changed to "missing". This command is effectively the inverse of LVM_ATTACHPV.

```c
ioctl(fd, LVM_RESYNCLV, &minor_num)
```

```c
int fd;
int minor_num;
```

This command resynchronizes a logical volume. As a result, every logical extent in the specified logical volume (minor_num), that has a physical extent in the LVM_PXSTALE state, will be updated from a mirror copy. If successful, then the corresponding physical extent's LVM_PXSTALE state is cleared.

You can use this command on a file descriptor corresponding to a logical volume device. In this case, the minor_num argument is ignored.

```c
ioctl(fd, LVM_RESYNCLX, &lv_resynclx)
```

```c
int fd;
struct lv_resynclx {
    ushort_t minor_num;
    ushort_t lx_num;
} lv_resynclx;
```

For each physical extent of a logical extent, if the physical extent is in the LVM_PXSTALE state, this command initiates mirror resynchronization for that physical extent. When the command is done, these extents will be in the LVM_ACTIVE state.

```c
ioctl(fd, LVM_RESYNCPV, &pv_key)
```

```c
int fd;
int pv_key;
```

This command resynchronizes a physical volume. For each physical extent on the physical volume that is in the LVM_PXSTALE state, this command resynchronizes the corresponding logical extent.

```c
ioctl(fd, LVM_SETVGID, &lv_setvgid)
```

```c
int fd;
struct lv_setvgid {
    lv_uniqueID_t vg_id;
} lv_setvgid;
```

```c
struct lv_uniqueID {
    ulong_t id1;
    ulong_t id2;
};
```

```c
typedef struct lv_uniqueID lv_uniqueID_t;
```
This command sets the volume group ID for the volume group implied by the file descriptor. It fails if the volume group already has a volume group ID and attached physical volumes. It is a necessary precursor to the `LVM_ATTACHPV ioctl`. If the unique ID passed in is 0 (zero), it is stored. The LVM ioctl commands use the following parameters:

**Parameters**

- **allocmap**
  Allocation map for logical volume.

- **cur_lvs**
  Allowed Values: 0 (zero) to 255
  Current number of logical volumes in this volume group.

- **cur_pvs**
  Allowed Values: 0 (zero) to LVM_MAXPVS
  Current number of physical volumes in this volume group.

- **currentsize**
  Allowed Values: 0 (zero) to LVM_MAXLXS
  Current size for logical volume.

- **extents**
  Pointer to the extent array. **flags**
  Allowed Values:
  
  - LVM_ACTIVATE_LVS
    Allow logical volume opens.
  
  - LVM_AUTO_RESYNC
    Automatically resynchronize returned volumes.
  
  - LVM_ALL_PVS_REQUIRED
    Activate fails if any physical volumes are missing.
  
  - LVM_NONMISSING_PVS_REQUIRED
    Activate fails if any physical volumes are missing which were not previously known as missing.

- **freepxs**
  Allowed Values: 0 (zero) to LVM_MAXPXS
  Current number of free extents.

- **lv_extent**
  Allowed Values: 0 (zero) to MAXLXS
  Logical extent number on volume.
\textit{lv\_flags} \quad \text{Allowed Values: Logical OR of the following constants:}

- \texttt{LVM\_LVDEFINED}
  \begin{itemize}
  \item Logical volume entry defined.
  \end{itemize}

- \texttt{LVM\_DISABLED}
  \begin{itemize}
  \item Logical volume unavailable for use.
  \end{itemize}

- \texttt{LVM\_NORELOC}
  \begin{itemize}
  \item New bad blocks are not relocated.
  \end{itemize}

- \texttt{LVM\_RDONLY}
  \begin{itemize}
  \item Read-only logical volume; no writes permitted.
  \end{itemize}

- \texttt{LVM\_STRICT}
  \begin{itemize}
  \item Allocate mirrors on different physical volumes.
  \end{itemize}

- \texttt{LVM\_VERIFY}
  \begin{itemize}
  \item Verify all writes to the logical volume.
  \end{itemize}

- \texttt{LVM\_NOMWC}
  \begin{itemize}
  \item Do not perform mirror write consistency for this logical volume.
  \end{itemize}

\textit{lv\_minor} \quad \text{Allowed Values: 0 (zero) to LVM\_MAXLVS}

Logical volume minor number. \textit{lv\_uniqueID} The unique ID should be set to a globally unique number.

\textit{lx\_num} \quad \text{Allowed Values: 0 (zero) to LVM\_MAXLXS}

Logical extent number to perform command on.

\textit{map} \quad \text{Pointer to the physical extent map.}

\textit{maxdefects} \quad \text{Allowed Values: 0 (zero) to bbpool\_len}

Maximum number of software-relocated defects.

\textit{maxlvs} \quad \text{Allowed Values: 0 (zero) to LVM\_MAXLVS}

Maximum number of logical volumes this volume group will contain.

\textit{maxlxs} \quad \text{Allowed Values: 0 (zero) to LVM\_MAXLXS}

New maximum size for logical volume, count of logical extents.

\textit{maxmirrors} \quad \text{Allowed Values: LVM\_MAXCOPIES}

Maximum number of mirrors allowed for this logical volume.

\textit{maxpvs} \quad \text{Allowed Values: 0 (zero) to LVM\_MAXPVS}

Maximum number of physical volumes this volume group will contain.

\textit{maxpxs} \quad \text{Allowed Values: 0 (zero) to LVM\_MAXPX}

Maximum number of physical extents any physical volumes in this volume group will contain.
**minor_num**  
Allowed Values: 1 to LVM_MAXLVS (or 255)  
Logical volume minor number.

**path**  
Allowed Values: PATH_MAX chars max  
NULL terminated physical volume pathname.

**numlxs**  
Allowed Values: 0 (zero) to LVM_MAXLXS  
Current number of logical extents.

**opt_avoid**  
Allowed Values: 0 (zero) to LVM_MIRAVOID  
Mirrors avoided during raw reads.

**opt_options**  
Allowed Values: Logical OR of the following constants:  
LVM_NORELOC  
No bad block relocation performed.  
LVM_VERIFY  
Verify all writes.

**pv_flags**  
Allowed Values: Logical OR of the following constants:  
LVM_PVNOALLOC  
No extent allocation allowed from this physical volume.  
LVM_PVRORELOC  
No new defects relocated on this physical volume.

**pv_flags**  
Allowed Values: Logical OR of the following constants:  
LVM_PVMISSING  
Physical volume is missing from the volume group.  
LVM_NOTATTACHED  
Physical volume is not attached to a volume group.

**pv_key**  
Allowed Values: Internally defined  
Physical volume identifier assigned by driver.

**numpxs**  
Allowed Values: 0 (zero) to LVM_MAXPXS  
Total number of physical extents on this physical volume.
pv_rdev  Device number (major,minor) currently used to access this physical volume. Not valid if physical volume is not attached.

px_num  Allowed Values: 0 (zero) to LVM_MAXPXS
Physical extent number to add or remove.

pxsize  Allowed Values: 1MB to 256MB
Physical extent size for all extents in this volume group (in bytes). Must be a power of 2.

pxspace  Allowed Values: 1MB to 256MB
Actual space allocated for each extent (in bytes). This must be the same or larger than pxsize.

px_count  Allowed Values: 0 (zero) to LVM_MAXPXS
Maximum number of physical extents this physical volume will ever contain.

px_free  Allowed Values: 0 (zero) to LVM_MAXPXS
Current number of free physical extents on this physical volume.

px_space  Allowed Values: 1MB to 256MB
Actual space allocated for each extent (in bytes). This must be the same or larger than pxsize.

sched_strat  Allowed Values:
LVM_SEQUENTIAL
  Write mirror copies sequentially.
LVM_PARALLEL
  Write mirror copies in parallel.

size  Allowed Values: 1 to LVM_MAXPXS
Number of extents to add or remove.

status  Allowed Values: Any This parameter is ignored. status Allowed Values: LVM_PXSTALE
Physical extent is stale.

status  Allowed Values: Logical OR of the following constants:
LVM_PXSTALE
  Physical extent is stale (does not contain valid data).
LVM_PXMISSING
  Physical extent is on a missing physical volume.
Errors

On failure, the LVM ioctl commands return the following:

**LVM_ACTIVATEVG**

- `[ENODEV]` No valid volume group descriptor areas (VGDA) were found on any physical volume.
- `[ENODEV]` Could not find a valid volume group status area (VGSA).
- `[ENOENT]` Quorum was lost while attempting to update the volume group status area.
- `[EEXIST]` LVM_ALL_PVS_REQUIRED was specified and at least one physical volume was missing.
- `[ENOMEM]` Insufficient kernel memory to complete request.
- `[ENXIO]` Quorum does not exist.
- `[EIO]` I/O error while reading the bad block directory.
- `[EINVAL]` There is an invalid physical extent in the VGDA's extent map.
- `[ENOTDIR]` LVM_NONMISSING_PVS_REQUIRED was specified and a "nonmissing" physical volume has not been attached.

**LVM_ATTACHPV**

- `[EFAULT]` The path parameter does not refer to a valid memory address.
- `[ENXIO]` The physical volume is a member of another volume group.
- `[ENOENT]` A component of the path parameter does not exist.
- `[ENOTDIR]` A component of the path parameter prefix is not a directory.
- `[ENXIO]` The path parameter refers to a device that does not exist, or is not configured into the kernel.
- `[ENOTBLK]` The path parameter designates a file that is not a block device.
- `[EACCESS]` A component of the path parameter was not accessible.
- `[ELOOP]` Too many symbolic links were encountered while looking up the path.
[ENAMETOOLONG] The path parameter is too long, or a component exceeds the maximum allowable size.

[EEXIST] A physical volume with the same physical volume number is already attached to this volume group.

[ENOTTY] Inappropriate ioctl for device; the command was attempted on a logical volume device rather than the control device.

[ENODEV] The physical volume is not a member of any volume group.

[EXDEV] The physical volume is not a member of the specified volume group.

[ENOMEM] Insufficient kernel memory to complete request.

[EIO] I/O error while reading the bad block directory or the volume group descriptor area.

**LVM_CHANGELV**

[EINVAL] The minor_num parameter is invalid.

[EINVAL] The maxmirrors parameter was not in the range (0, LVM_MAXCOPIES-1).

[EINVAL] The lv_flags parameter contains an unrecognized flag.

[EROFS] The volume group is not activated.

[ENODEV] The minor_num parameter refers to a nonexistent logical volume.

[EINVAL] The sched_strat parameter was not one of LVM_PARALLEL or LVM_SEQUENTIAL. [EBUSY] The maxlxs or maxmirrors parameter is smaller than the current allocation for the logical volume. Must deallocate before changing the logical size.

[ENOMEM] Insufficient kernel memory to complete request.

[EFAULT] The parameter does not refer to a valid memory address.

**LVM_CHANGEPV**

[EINVAL] The pv_flags parameter contains unrecognized flags.

[ENXIO] The pv_key parameter references a nonexisting physical volume.

[EBUSY] There are more existing defects than could be supported with the max_defects parameter.

[EFAULT] The parameter does not refer to a valid memory address.

[ENOTTY] Inappropriate ioctl for device; the command was attempted on a logical volume device rather than the control device.
LVM_CREATELV

[EINVAL] The minor_num parameter is 0 (zero).
[EDOM] The minor_num parameter is greater than the maximum number of logical volumes in the volume group.
[EEXIST] The minor_num parameter refers to an already existing logical volume.
[ENOMEM] Insufficient kernel memory to satisfy the request.
[EROFS] The volume group is not activated.
[ENODEV] The minor_num parameter refers to a nonexistent logical volume.
[EINVAL] The sched_strat parameter was not one of LVM_PARALLEL or LVM_SEQUENTIAL.
[EBUSY] The maxlxs or maxmirrors parameter is smaller than the current allocation for the logical volume. Must deallocate before changing the logical size.
[ENOTTY] Inappropriate ioctl for device; the command was attempted on a logical volume device rather than the control device.

LVM_CREATEVG

[EINVAL] Invalid parameter structure; some field within the structure contained an invalid value. Specific checks are made for; 0 (zero) volume group ID, the maxlvs parameter greater than LVM_MAXLVS, the maxpvs parameter greater than MAXPVS, the maxpxs parameter greater than LVM_MAXPXS, 1MB <= psize <= 256MB, psize <= pxspace, the pxspace parameter is a multiple DEV_BSIZE, the pv_flags parameter is valid.
[EEXIST] The volume group already exists.
[ENOMEM] Insufficient kernel memory to complete request.
[ENOSPC] Insufficient space on the volume for the volume group reserved area (VGRA).
[ENOENT] The file specified by the path parameter does not exist.
[ENODEV] The path parameter does not specify a valid physical volume.
[EPERM] Permission denied on open of the path parameter.
[EIO] Unable to read the physical volume.
[ENOTBLK] The path parameter designates a file that is not a block device.
The physical volume has no driver configured.
The parameter does not refer to a valid memory address.
Inappropriate ioctl for device; the command was attempted on a logical volume device rather than the control device.

Inappropriate ioctl for device; the command was attempted on a logical volume device rather than the control device.

The minor_num parameter was less than or equal to zero.
The volume group is not activated.
The minor_num parameter refers to a nonexistent logical volume.
The indicated logical volume is open.
Inappropriate ioctl for device; the command was attempted on a logical volume device rather than the control device.

Inappropriate ioctl for device; the command was attempted on a logical volume device rather than the control device.

The minor_num parameter was less than or equal to zero.
The volume group is not activated.
The minor_num parameter refers to a nonexistent logical volume.
The indicated logical volume is open.
Inappropriate ioctl for device; the command was attempted on a logical volume device rather than the control device.

The paramerter does not refer to a valid memory address.
An extent described by the extent array is already in use.
The specified logical volume does not exist.

The volume group is not active.
Unable to allocate memory.
The device is not a valid physical volume.
Write permission denied on the device.
A component of the path parameter was not accessible.
Unable to read the physical volume.
The path parameter designates a file that is not a block device.
Write permission denied on the device.
The parameter does not refer to a valid memory address.
Inappropriate ioctl for device; the command was attempted on a logical volume device rather than the control device.
LVM_OPTIONSET/LVM_OPTIONGET

[EINVAL] The opt_avoid parameter out of range (LVM_OPTIONSET only).

[EINVAL] The opt_options parameter included invalid bit values (LVM_OPTIONSET only).

[EFAULT] The parameter does not refer to a valid memory address.

[ENOTTY] Inappropriate ioctl for device; the command was attempted on the control device.

LVM_QUERYLV

[EINVAL] The minor_num parameter is 0 (zero).

[ENXIO] The volume group is not activated.

[EFAULT] The parameter does not refer to a valid memory address.

LVM_QUERYLVMAP

[EFAULT] The parameter does not refer to a valid memory address.

LVM_QUERYPV

[EFAULT] The parameter does not refer to a valid memory address.

[ENODEV] The specified pv_key parameter does not correspond to physical volume attached to this volume group, that is, no such device.

[ENOTTY] Inappropriate ioctl for device; the command was attempted on the control device.

LVM_QUERYPVMAP

[EFAULT] The parameter does not refer to a valid memory address.

[ENODEV] The specified pv_key parameter does not correspond to physical volume attached to this volume group, that is, no such device.

[ENOTTY] Inappropriate ioctl for device; the command was attempted on the control device.

LVM_QUERYPATH

[EFAULT] The parameter does not refer to a valid memory address.

[ENOENT] A component of the path parameter does not exist.

[ENOTDIR] A component of the path parameter prefix is not a directory.

[ENXIO] The path parameter refers to a device that does not exist, or is not configured into the kernel.

[ENOTBLK] The path parameter designates a file that is not a block device.

[EACCES] A component of the path parameter was not accessible.
Too many symbolic links were encountered while looking up the path.

The path parameter is too long, or a component exceeds the maximum allowable size.

The specified path parameter does not correspond to physical volume attached to this volume group, that is, no such device.

Inappropriate ioctl for device; the command was attempted on a logical volume device.

The parameter does not refer to a valid memory address.

Inappropriate ioctl for device; the command was attempted on a logical volume device.

The parameter does not refer to a valid memory address.

The parameter does not refer to a valid memory address.

Inappropriate ioctl for device; the command was attempted on a logical volume device.

Inappropriate ioctl for device; the command was attempted on a logical volume device rather than the control device.

Function: ioctl(2)
msqid_ds

**Purpose**
Defines a message queue

**Synopsis**
```
#include <sys/msg.h>

struct msqid_ds{
    struct ipc_perm msg_perm;
    struct msg *msg_first;
    struct msg *msg_last;
    u_short msg_cbytes;
    u_short msg_qnum;
    u_short msg_qbytes;
    u_short msg_lspid;
    u_short msg_lrpid;
    time_t msg_stime;
    time_t msg_rtime;
    time_t msg_ctime;
};
```

**Description**
The `msqid_ds` structure defines a message queue associated with a message queue ID. There is one queue per message queue ID. Collectively, the queues are stored as an array, with message queue IDs serving as an index into the array.

A message queue is implemented as a linked list of messages, with `msg_first` and `msg_last` pointing to the first and last messages on the queue.

The IPC permissions for the message queue are implemented in a separate, but associated, `ipc_perm` structure.

A message queue is created indirectly via the `msgget()` call. If `msgget()` is called with a non-existent message queue ID, the kernel allocates a new `msqid_ds` structure, initializes it, and returns the message queue ID that is to be associated with the message queue.

**Fields**

- **msg_perm**  The `ipc_perm` structure that defines permissions for message operations. See NOTES.
- **msg_first**  A pointer to the first message on the queue.
- **msg_last**  A pointer to the last message on the queue.
The current number of bytes on the queue.

msg_qnum
The number of messages currently on the queue.

msg_qbytes
The maximum number of bytes allowed on the queue.

msg_lspid
The process ID of the last process that called msgsnd() for the queue.

msg_lrpid
The process ID of the last process that called msgrcv() for the queue.

msg_stime
The time of the last msgsnd() operation.

msg_rtime
The time of the last msgrcv() operation.

msg_ctime
The time of the last msgctl() operation that changed a member of the msqid_ds structure.

The msg_perm field identifies the associated ipc_perm structure that defines the permissions for operations on the message queue. The ipc_perm structure (from the sys/ipc.h header file) is shown here.

```
struct ipc_perm {
    ushort uid;  /* owner’s user id */
    ushort gid;  /* owner’s group id */
    ushort cuid; /* creator’s user id */
    ushort cgid; /* creator’s group id */
    ushort mode; /* access modes */
    ushort seq;  /* slot usage sequence number */
    key_t key;   /* key */
};
```

The mode field is a 9-bit field that contains the permissions for message operations. The first three bits identify owner permissions; the second three bits identify group permissions; and the last three bits identify other permissions. In each group, the first bit indicates read permission; the second bit indicates write permission; and the third bit is not used.

**Related Information**

Functions: msgctl(2), msgget(2), msgrcv(2), msgsnd(2)
Introduction to socket networking facilities

#include <sys/socket.h>
#include <net/route.h>
#include <net/if.h>

This section is a general introduction to the networking facilities available in the system. Documentation in this part of Section 7 is broken up into three areas: **protocol families** (domains), **protocols**, and **network interfaces**.

All network protocols are associated with a specific **protocol family**. A protocol family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. A protocol family may support multiple methods of addressing, though the current protocol implementations do not. A protocol family is normally comprised of a number of protocols, one per socket type. It is not required that a protocol family support all socket types. A protocol family may contain multiple protocols supporting the same socket abstraction.

A protocol supports one of the socket abstractions detailed in the reference page for the **socket( )** function. A specific protocol may be accessed either by creating a socket of the appropriate type and protocol family, or by requesting the protocol explicitly when creating a socket. Protocols normally accept only one type of address format, usually determined by the addressing structure inherent in the design of the protocol family and network architecture. Certain semantics of the basic socket abstractions are protocol specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide nonstandard facilities or extensions to a mechanism. For example, a protocol supporting the SOCK_STREAM abstraction may allow more than one byte of out-of-band data to be transmitted per out-of-band message.

A network interface is similar to a device interface. Network interfaces comprise the lowest layer of the networking subsystem, interacting with the actual transport hardware. An interface may support one or more protocol families, address formats, or both. The **SYNOPSIS** section of each network interface entry gives a sample specification of the related drivers for use in providing a system description to the **config** program. The **ERRORS** section lists messages which may appear on the console and/or in the system error log, /var/log/messages (see the **syslogd** function), due to errors in device operation.
The system currently supports the DARPA Internet protocols and the Xerox Network Systems protocols. Raw socket interfaces are provided to the IP layer of the DARPA Internet, and to the IDP of Xerox NS. Consult the appropriate manual pages in this section for more information regarding the support for each protocol family.

Addressing

Associated with each protocol family is an address format. All network address adhere to a general structure, called a sockaddr. However, each protocol imposes finer and more specific structure, generally renaming the variant.

Both the 4.3BSD and 4.4BSD sockaddr structures are supported by OSF/1. The default sockaddr structure is the 4.3BSD structure, which is as follows:

```c
struct sockaddr {
    u_short sa_family;
    char sa_data[14];
};
```

If the compile-time option _SOCKADDR_LEN is defined before the sys/socket.h header file is included, however, the 4.4BSD sockaddr structure is defined, which is as follows:

```c
struct sockaddr {
    u_char sa_len;
    u_char sa_family;
    char sa_data[14];
};
```

The 4.4BSD sockaddr structure provides for a sa_len field, which contains the total length of the structure. Unlike the 4.3BSD sockaddr structure, this length may exceed 16 bytes.

The following address values for sa_family are known to the system (and additional formats are defined for possible future implementation):

```c
#define AF_UNIX 1 /* local to host (pipes, portals) */
#define AF_INET 2 /* internetwork: UDP, TCP, etc. */
#define AF_NS 6 /* Xerox NS protocols */
```

Routing

The UNIX operating system provides packet routing facilities. The kernel maintains a routing information database, which is used in selecting the appropriate network interface when transmitting packets.
A user process (or possibly multiple cooperating processes) maintains this database by sending messages over a special kind of socket. This supplants fixed size ioctl’s used in earlier releases.

This facility is described in the files reference page for the route function.

Interfaces

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, lo, do not.

The following ioctl calls may be used to manipulate network interfaces. The ioctl is made on a socket (typically of type SOCK_DGRAM) in the desired domain. Most of the requests supported in earlier releases take an ifreq structure as its parameter. This structure has the following form:

```c
struct ifreq {
    #define IFNAMSIZ 16
    char ifr_name[IFNAMSIZ]; /* if name, e.g. "en0" */
    union {
        struct sockaddr ifru_addr;
        struct sockaddr ifru_dstaddr;
        struct sockaddr ifru_broadaddr;
        short ifru_flags;
        int ifru_metric;
        caddr_t ifru_data;
    } ifr_ifru;
    #define ifr_addr ifr_ifru.ifru_addr /* address */
    #define ifr_dstaddr ifr_ifru.ifru_dstaddr /* other end of p-to-p link */
    #define ifr_broadaddr ifr_ifru.ifru_broadaddr /* broadcast address */
    #define ifr_flags ifr_ifru.ifru_flags /* flags */
    #define ifr_metric ifr_ifru.ifru_metric /* metric */
    #define ifr_data ifr_ifru.ifru_data /* for use by interface */
};
```
Calls which are now deprecated are:

**SIOCSIFADDR**  
Set interface address for protocol family. Following the address assignment, the “initialization” routine for the interface is called.

**SIOCSIFDSTADDR**  
Set point to point address for protocol family and interface.

**SIOCSIFBRDADDR**  
Set broadcast address for protocol family and interface.

All *ioctl* requests to obtain addresses and requests both to set and retrieve other data are still fully supported and use the *ifreq* structure:

**SIOCGIFADDR**  
Get interface address for protocol family.

**SIOCGIFDSTADDR**  
Get point to point address for protocol family and interface.

**SIOCGIFBRDADDR**  
Get broadcast address for protocol family and interface.

**SIOCSIFFLAGS**  
Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified; some interfaces may be reset so that incoming packets are no longer received. When marked up again, the interface is reinitialized.

**SIOCGIFFLAGS**  
Get interface flags.

**SIOCSIFMETRIC**  
Set interface routing metric. The metric is used only by user-level routers.

**SIOCGIFMETRIC**  
Get interface metric.
There are three requests that make use of a new structure:

SIOCAIFADDR

An interface may have more than one address associated with it in some protocols. This request provides a means to add additional addresses (or modify characteristics of the primary address if the default address for the address family is specified). Rather than making separate calls to set destination addresses, broadcast addresses, or network masks (now an integral feature of multiple protocols) a separate structure is used to specify all three facets simultaneously:

```c
struct ifaliasreq {
    char ifra_name[IFNAMSIZ]; /* if name, e.g. "en0" */
    struct sockaddr ifra_addr;
    struct sockaddr ifra_broadaddr;
    struct sockaddr ifra_mask;
};
```

One would use a slightly tailored version of this struct are specific to each family (replacing each sockaddr by one of the family-specific type). Where the sockaddr itself is larger than the default size, one needs to modify the ioctl identifier itself to include the total size.

SIOCDIFADDR

This request deletes the specified address from the list associated with an interface. It uses the if_aliasreq structure to permit protocols to allow multiple masks or destination addresses, and it adopts the convention that specification of the default address means to delete the first address for the interface belonging to the address family in which the original socket was opened.
SIOCGIFCONF

Get interface configuration list. This request takes an \texttt{ifconf} structure (see below) as a value-result parameter. The \texttt{ifc_len} field should be initially set to the size of the buffer pointed to by \texttt{ifc_buf}. On return it contains the length, in bytes, of the configuration list.

\begin{verbatim}
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */

struct ifconf {
    int ifc_len; /* size of associated buffer */
    union {
        caddr_t ifcu_buf;
        struct ifreq *ifcu_req;
    } ifc_ifcu;
#define ifc_buf ifc_ifcu.ifcu_buf /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req /* array of structures returned */
};
\end{verbatim}

\section*{Related Information}

Functions: \texttt{socket(2)}, \texttt{ioctl(2)}

Files: \texttt{config(8)}, \texttt{routed(8)}
ns

**Purpose**  Xerox Network Systems protocol family

**Synopsis**

options NS
options NSIP
pseudo-device ns

**Description**

The NS protocol family is a collection of protocols layered atop the Internet Datagram Protocol (IDP) transport layer, and using the Xerox NS address formats. The NS family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, SOCK_SEQPACKET, and SOCK_RAW socket types. The SOCK_RAW interface is a debugging tool, allowing you to trace all packets entering (or with toggling kernel variable, additionally leaving) the local host.

**Addressing**

The NS addresses are 12-byte quantities, consisting of a 4-byte network number, a 6-byte host number and a 2-byte port number, all stored in network standard format. (On the VAX and other machines, these are word and byte reversed; on a Sun machine, they are not reversed). The `netns/ns.h` include file defines the NS address as a structure containing unions (for quicker comparisons).

Both the 4.3BSD and 4.4BSD `sockaddr_ns` structures are supported by OSF/1. The default `sockaddr_ns` structure is the 4.3BSD structure, which is as follows:

```c
struct sockaddr_ns {
    u_short      sns_family;
    struct ns_addr  sns_addr;
    char         sns_zero[2];
};
```

If the compile-time option `_SOCKADDR_LEN` is defined before the `netns/ns.h` header file is included, however, the 4.4BSD `sockaddr` structure is defined, which is as follows:

```c
struct sockaddr_ns {
    u_char         sns_len;
    u_char         sns_family;
    struct ns_addr  sns_addr;
    char         sns_zero[2];
};
```
The 4.4BSD `sockaddr_in` structure provides for a `sms_len` field, which contains the total length of the structure.

The `ns_addr` field is composed as follows:

```c
union ns_host {
    u_char    c_host[6];
    u_short   s_host[3];
};

union ns_net {
    u_char    c_net[4];
    u_short   s_net[2];
};

struct ns_addr {
    union ns_net    x_net;
    union ns_host   x_host;
    u_short x_port;
};
```

Sockets may be created with an address of all zeros to effect "wildcard" matching on incoming messages. The local port address specified in a `bind(2)` call is restricted to be greater than `NSPORT_RESERVED (=3000, in netns/ns.h )` unless the creating process is running as the superuser, providing a space of protected port numbers.

Protocols

The NS protocol family supported by the operating system is comprised of the Internet Datagram Protocol (IDP) `idp(4)`, Error Protocol (available through IDP), and Sequenced Packet Protocol (SPP) `spp(4)`. SPP is used to support the SOCK_STREAM and SOCK_SEQPACKET abstraction, while IDP is used to support the SOCK_DGRAM abstraction. The error protocol is responded to by the kernel to handle and report errors in protocol processing; it is, however, not easily accessible to user programs.

Related Information

Functions: `gethostbyname(3)`, `getnetent(3)`, `getprotoent(3)`, `getservent(3)`, `ns(3)`

Files: `netintro(7)`, `spp(7)`, `idp(7)`, `nsip(7)`
nsip

Purpose  Software network interface encapsulating NS packets in IP packets

Synopsis  options NSIP
#include <netns/ns_if.h>

Description

The nsip interface is a software mechanism which may be used to transmit Xerox NS packets through otherwise uncooperative networks. It functions by prepending an IP header, and resubmitting the packet through the UNIX IP machinery.

The superuser can advise the operating system of a willing partner by naming an IP address to be associated with an NS address. Presently, only specific host pairs are allowed, and for each host pair, an artificial point-to-point interface is constructed. At some future date, IP broadcast addresses or hosts may be paired with NS networks or hosts.

Specifically, a socket option of SO_NSIP_ROUTE is set on a socket of family AF_NS, type SOCK_DGRAM, passing the following structure:

```c
struct nsip_req {
    struct sockaddr rq_ns; /* must be ns format destination */
    struct sockaddr rq_ip; /* must be ip format gateway */
    short rq_flags;
};
```

Errors

nsip: can’t handle afn

The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

Related Information

Files: netintro(4), ns(4)
null

Purpose  Data sink

Description  Data written on a null special file are discarded.

 Reads from a null special file always return 0 (zero) bytes.

Examples  

 To read a device and discard all the data using the dd command:

    dd if=/dev/rz1c of=/dev/null

 To create a zero length file using the cat command:

    cat > foo < /dev/null

Files

/dev/null
OSF/ROSE

Purpose
Object file format for output from OSF/1 translators

Synopsis
#include <mach_o_format.h>
#include <mach_o_header.h>
#include <mach_o_vals.h>
#include <machine/mach_o_types.h>

Description

The OSF/ROSE format is the object file format for output files produced by the OSF/1 translators (that is, the assembler, the compilers, and the linker).

An OSF/ROSE object file consists of the following:

- A fixed-length file header.
- A list of variable-length load commands, in no particular order. One of the load commands is the load command map, which contains the offsets of the other load commands in the list. Individual load commands are referenced as indexes into the load command map.
- A variable number of sections, in no particular order. The sections contain the object file's program data and meta data. Each section must have an appropriate load command to serve as its header.

An object file must begin with the file header, followed immediately by the list of load commands with the load command map. The sections follow the load commands.

The OSF/ROSE format incorporates an implicit hierarchy whereby entries in the load command map point to individual load commands, which in turn point to their associated sections.

The OSF/ROSE format is designed to support the features of the OSF/1 program loader, including regions and packages. However, it is also a general-purpose extensible format that can be adapted to other loading schemes.

Program Data and Meta Data

The OSF/ROSE format distinguishes between program data and nonprogram, or meta, data. Program data is contained only in program sections, called regions.
By definition, all other sections contain meta data. Sections containing meta data can be read in or mapped anywhere in virtual memory. Examples of meta data include the lists of symbols exported by various regions and the lists of external symbols imported (referenced) by various regions. The OSF/ROSE format distinguishes the different types of meta data by providing a separate load command for each type.

Data Representation

Most of the meta data in an OSF/ROSE object file is represented and aligned in a natural way for C on the machine that is to run the program. This allows the loader to use the most efficient accessing code. Most fields are either long or short integers declared with types specific to OSF/ROSE. On most 32-bit word machines, these long and short integers correspond to four bytes and two bytes respectively.

To handle those situations where it is necessary to read object files created for different types of machines, OSF/ROSE defines a single data representation field in the file header. Each value of the field corresponds to a particular combination of attributes (for example, byte order, word size, compiler-specific packing of structures, and so on.). The implementation of a single data representation field allows cross tools to use data conversion routines.

Note that currently there is no provision for assigning unique data representation field values. The byte order field in the file header is probably sufficient for most machines.

In order for the data and machine representation fields to be read correctly in all cases, they must be in the OSF/ROSE canonical form. These fields are stored in the object file header, which is treated differently from the rest of the file. The header fields are always in network byte order (big-endian), are aligned in the natural way for 32-bit word machines, and are two or four 8-bit bytes long.

All character strings in the meta data of an OSF/ROSE object file are null-terminated and consist of single-byte or multibyte characters. Note that wide characters are not supported. The OSF/ROSE tools are required to use only single-byte-character processing, since the only comparisons are for equality.

File Header

The OSF/ROSE object file header contains two types of information. One type of information describes the data representation and indicates how the file can be used. For example, it describes the data representation of the meta data, machine- and vendor-specific information, and version numbers. This information is used by the loader and linker to determine whether they are able to handle a given object file. The other type of information is used in reading in the load commands. (The list of load commands follows the file header.)
A given **OSF/ROSE** file header can exist in either a canonical or a native form.

The **canonical form**, also known as the "raw" form, describes the file representation of the header. As stored in an object file, the header must be in the canonical form. The canonical form allows the header to be interpreted on any system.

The **native form** describes the memory representation of the header. When accessed in memory (for example, by the linker), the header must be in native form.

To support the reading and writing of the canonical form on different machines, **OSF/1** provides the `decode_mach_o_hdr()` and `encode_mach_o_hdr()` functions. These functions allow file headers to be converted between the **OSF/ROSE** canonical form and a given machine's native form. For example, to produce an executable object file, the linker fills in the header fields in native form and reserves space for the header in the object file. It then calls `encode_mach_o_hdr()` to convert the header to canonical form and write it to the reserved space.

The file header includes version information to support backward compatibility with previous versions of the **OSF/ROSE** format. The header structure can be modified only by extending it -- that is, only by adding fields at the end. This requirement ensures that a conversion routine can always read in the number of bytes that a particular version requires and expect to get all the information available for that version of the header. In other words, the conversion routines can convert between any two versions that they recognize.

When the `decode_mach_o_hdr()` routine reads the header from a file, it returns the version of the header from the file (canonical form) rather than the version from the memory structure being filled in (native form). It is the caller's responsibility to check the version, if version checking is required.

The native form of the file header is described by the following structure declaration from the machine-independent `mach_o_header.h` file. If a program needs to access an object file header, it must call `decode_mach_o_hdr(2)` to copy the contents of the canonical (raw) form of the header into this structure so they
can be interpreted in a straightforward way. (The canonical form of the header is defined by the raw_mh_header_t structure declaration in the machine/mach_o_header_md.h file.)

```c
typedef struct mo_header_t {
    mo_long_t  moh_magic;
    mo_short_t moh_major_version;
    mo_short_t moh_minor_version;
    mo_short_t moh_header_version;
    mo_short_t moh_max_page_size;
    mo_short_t moh_byte_order;
    mo_short_t moh_data_rep_id;
    mo_cpu_type_t moh_cpu_type;
    mo_cpu_subtype_t moh_cpu_subtype;
    mo_vendor_type_t moh_vendor_type;
    mo_long_t   moh_flags;
    mo_offset_t moh_load_map_cmd_off;
    mo_offset_t moh_first_cmd_off;
    mo_long_t   moh_sizeofcmds;
    mo_long_t   moh_n_load_cmds;
    mo_long_t   moh_reserved[2];
} mo_header_t;
```

The header fields are defined as follows:

- **moh_magic** The magic number for the OSF/ROSE header. The magic number is a 32-bit number that is the same on all machines. The values of this number as it appears when read directly on various machines are defined in the mach_o_vals.h file. The way the value looks when read directly on the current machine is defined in the mach_o_types.h header file as OUR_MOH_MAGIC. The value as it appears in the native (translated) version of the header is defined in the mach_o_header.h file as MOH_MAGIC.

- **moh_major_version** The major version number for the header structure. The most recently defined value is MOH_MAJOR_VERSION (defined in the mach_o_vals.h file).

- **moh_minor_version** The minor version number for the header structure. The most recently defined value is MOH_MINOR_VERSION (defined in the mach_o_header.h file).
moh_header_version
The header version. The most recently defined value is
MOH_HEADER_VERSION (defined in the mach_o_header.h
file).

moh_max_page_size
The maximum page size assumed by the linker, in bytes. In
executable files this refers to the virtual memory alignment of
program regions. In some executable files the regions must be
loaded at specified virtual addresses or at addresses that are at
specified offsets from other regions. Regions also have to be loaded
on page boundaries. If the system's page size is either larger than
moh_max_page_size or is not an even divisor of
moh_max_page_size, it may not be possible to load a region both on
a page boundary and in the proper relation to other regions.
The safest course of action is for the linker to make
moh_max_page_size equal to the system's page size, since some
loaders insist on this when loading regions with relative addressing
requirements.
This field should be irrelevant in files whose regions have no
relative addressing requirements.

moh_byte_order
The byte order for the target machine. Values are defined in the
mach_o_vals.h file. The value used on the current machine is
defined in the mach_o_types.h file as OUR_BYTE_ORDER.

moh_data_rep_id
The data representation used on the target machine. Values are
defined in the mach_o_vals.h file. The value used on the current
machine is defined in the mach_o_types.h file as
OUR_DATA_REP_ID.

moh_cpu_type
The type of machine on which the code will run. Values are defined
in the mach_o_vals.h file. The value used on the current machine is
defined in the mach_o_types.h file as OUR_CPU_TYPE.

moh_cpu_subtype
The vendor variation of the basic CPU type. This field is provided
to handle the case where two (or more) machines with the same
basic CPU type are so different that executable files compiled for
one machine should not be executed on the other. A restriction of
this type should be enforced in the file header so that linkers and loaders can prevent incompatible files from being executed. Values are defined in the `mach_o_vals.h` file. The value used on the current machine is defined in the `mach_o_types.h` file as `OUR_CPU_SUBTYPE`.

`moh_vendor_type`

The vendor type of the format variation used in this file. Vendor types are used to differentiate files with incompatible formats, such as different symbol or relocation information, when the incompatibility is introduced to provide an alternative rather than a replacement. (Replacements affect the version number.) It is the responsibility of the party defining the vendor type to ensure that the necessary software knows about it. The most significant half of the possible values are reserved for machine-dependent vendor types, and the least significant half for machine-independent types. Values for machine-independent vendor types are defined in the `mach_o_vals.h` file. The value used on the current machine is defined in the `mach_o_vals.h` file as `OUR_VENDOR_TYPE`.

`moh_flags`

Characteristics of the object file, indicating how the object file may be used. These values are not mutually exclusive. Possible values are as follows:

- **MOH_RELOCATABLE_F**
  - The object file has loader relocation.

- **MOH_LINKABLE_F**
  - The object file has linker relocation.

- **MOH_EXECABLE_F**
  - The object file has `crt0` and can be exec’d.

- **MOH_EXECUTABLE_F**
  - The object file can be loaded for execution.

- **MOH_UNRESOLVED_F**
  - The object file has unresolved references to imported symbols.

`moh_load_map_cmd_off`

The offset of the load command map in bytes from the beginning of the file.
The offset of the first load command in bytes from the beginning of the file.

The number of bytes occupied by all the load commands, including the load command map.

The number of load commands.

Reserved for future use.

Load Commands

In the OSF/ROSE format, load commands are used for section headers, loader directives, and information that is too short to have its own section. The load commands must be grouped together and must immediately follow the file header (although not necessarily on the next available byte). A variable number of load commands can be specified, in no particular order. Individual load commands are of variable length.

All load commands begin with a standard header, which includes type information and the file offset and length of the associated section, if there is one. The file offset and length fields are included even if there is no associated section.

The load command header is described by the following structure declaration from the mach_o_format.h file:

define struct ldc_header_t {
    mo_long_t  ldci_cmd_type;
    mo_long_t  ldci_cmd_size;
    mo_offset_t ldci_section_off;
    mo_long_t  ldci_section_len;
} ldc_header_t;

The load command header fields are defined as follows:

The load command type. The most significant half of the type field values are machine or vendor dependent and are unique only for a given CPU type. The name space for these values is determined by the object file header’s moh_cpu_type field. The least significant half of the type field values are reserved for machine- and vendor-
independent load commands. These values should have unique meanings, but there is no mechanism for enforcing uniqueness. The currently defined machine- and vendor-independent types (from the `mach_o_format.h` file) are as follows:

**LDC_UNDEFINED**
- An undefined load command. Used when logically deleting a load command entry from the load command map.

**LDC_CMD_MAP**
- Load command for the load command map.

**LDC_INTERPRETER**
- Load command for the program interpreter (no section). Only one interpreter load command is allowed. If this load command is used, it must be the first entry in the load command map so that further processing of the load commands can be avoided if an interpreter is to be called instead.

**LDC_STRINGS**
- Load command for a strings section.

**LDC_REGION**
- Load command for a region section (part of the program).

**LDC_RELOC**
- Load command for a relocation information section.

**LDC_PACKAGE**
- Load command for an import or export package list (no section).

**LDC_SYMBOLS**
- Load command for a symbols section.

**LDC_ENTRY**
- Load command for the program main entry point (no section).

**LDC_FUNC_TABLE**
- Load command for a function table (no section).

**LDC_GEN_INFO**
- Load command for general information (no section).

`ldci_cmd_size` The size of the load command in bytes.
ldci_section_off
The offset of the associated section from the beginning of the file. If there is no associated section, this field is set to 0 (zero).

ldci_section_len
The length of the associated section in bytes. If there is no associated section, this field is set to 0 (zero).

Data Types Specific to OSF/ROSE

The OSF/ROSE format implements several special data types for use with the format declarations. The typedefs for the machine-dependent base types are defined in the mach_o_types.h file. The machine-independent typedefs are defined in the mach_o_format.h file and are listed below:

typedef mo_long_t mo_lcid_t;
The mo_lcid_t typedef identifies a load command entry (index) in the load command map.

typedef struct mo_addr_t {
    mo_lcid_t  adr_lcid;
    mo_offset_t  adr_sctoff;
} mo_addr_t;
The mo_addr_t typedef describes an address in terms of an offset within a section. It specifies the section as a load command index and contains the offset in bytes of the address within the associated section.

typedef struct mo_index_t {
    mo_lcid_t  adx_lcid;
    mo_long_t  adx_index;
} mo_index_t;
The mo_index_t typedef identifies an element within an array-type section. It specifies the section as a load command index and contains the index of the element within the associated section.

typedef struct mo_rel_addr_t {
    mo_lcid_t  adrl_lcid;
    mo_offset_t  adrl_reloff;
} mo_rel_addr_t;
The mo_rel_addr_t typedef describes an address that is relative to a (region) section. It specifies the section as a load command index and contains the offset in bytes of the address from the beginning of the associated section. The address is not within the section and the offset is not negative. This type of address is used at load time to position a region at a fixed offset relative to another region.
The Load Command Map

Direct file offsets can be specified only in load commands. Each section has its own load command, which serves as a header for the section. Any reference from one section to another is made indirectly, via the target section's load command. As an aid in determining the addresses of the load commands, the OSF/ROSE format provides a special load command called the load command map. As a load command, it is stored with the other load commands.

The load command map is an array that contains the offsets of all the other load commands. Each load command is represented by an index into the array. Load command indexes already assigned will not change when new load commands or sections are added to the file or when old load commands are deleted.

Internal address references (that is, references within the object file) reflect the load command map/load command/section hierarchy. They are specified in terms of a load command map index and an offset within a section. The index indicates an entry within the load command map array which in turn provides the offset of a load command. The load command identifies the section to which the offset applies. The OSF/ROSE format specifies a special data type (mo_addr_t) to represent internal address references.

To logically delete a load command, change its entry in the load command map to LCM_INVALID_ENTRY and change the ldci_cmd_type field in the load command's header to undefined (LDC_UNDEFINED).

While the OSF/ROSE format allows load commands to be logically deleted by marking their entries in the load command map as invalid, the load commands for nonabsolute regions must not be logically deleted since they may be referenced by other sections.

The OSF/1 linker and loader are not required to check for undefined load commands. Therefore, only those load command map entries that are not used by the linker and loader can be made obsolete.

The load command map load command identifies the strings section used by the other load commands.

The load command map is described by the following structure declaration from the mach_o_format.h file:

```c
typedef struct load_cmd_map_command_t {
    ldc_header_t ldc_header;
    mo_lcid_t lcm_ld_cmd_strings;
    mo_long_t lcm_nentries;
    mo_offset_t lcm_map [1];
} load_cmd_map_command_t;
```
The load command map fields are defined as follows:

- **ldc_header**: The load command header for the load command map. In this structure, the ldci_cmd_type field must be set to LDC_CMD_MAP. The section fields must be set to 0 (zero) since the load command map cannot have an associated section.

- **lcm_ld_cmd_strings**: The load command map index (entry) of the strings section that contains the strings used by the load commands.

- **lcm_nentries**: The number of load command entries in the load command map, including any invalid entries that are followed by valid entries.

- **lcm_map**: The variable-length array that contains the file offsets (type mo_offset_t) of the load commands.

### Regions

A **region** is an object file section that contains a piece of the program. Any nonregion section is considered meta data. Meta data sections can be read in or mapped anywhere in virtual memory because their file offsets are contained only in their associated load commands. The loader manages nonregion sections differently from region sections.

When loaded, a region exists as a virtually contiguous range of bytes in process address space. To fully support this format, a loader should be able to load an arbitrary number of regions (rather than just text, data, and bss).

A region's attributes (such as address, size, protection, and type) are described in its associated region load command. All object files, linkable and executable, use the same region load command structure.
The region load command is described by the following structure declaration from the `mach_o_format.h` file:

```c
typedef struct region_command_t {
    ldc_header_t ldc_header;
    mo_addr_t regc_region_name;
    union {
        mo_vm_addr_t vm_addr;
        mo_rel_addr_t rel_addr;
    } regc_addr;
    mo_long_t regc_vm_size;
    mo_long_t regc_flags;
    mo_lcid_t regc_reloc_addr;
    mo_long_t regc_addralign;
    mo_short_t regc_usage_type;
    mo_short_t regc_initprot;
} region_command_t;
```

The region load command fields are defined as follows:

- **ldc_header**: The load command header for the region load command. In this structure, the `ldci_cmd_type` field must be set to `LDC_REGION`. The section offset and length fields are normally filled in, but they can be set to 0 (zero) for a bss (uninitialized data) region.

- **regc_region_name**: The name of the region, specified as an address within the strings section used by the load commands.

- **regc_addr**: The address to use for the region. This field is defined by a union and is used in conjunction with `regc_flags`. If `regc_flags` is `REG_ABS_ADDR_F`, `regc_addr.vm_addr` specifies the absolute address to use. If `regc_flags` is `REG_REL_ADDR_F`, `regc_addr.rel_addr` specifies the address in terms of the region that this region is relative to. If neither of those flags is specified, this field is not used (that is, the loader chooses the address).

- **regc_vm_size**: The amount of memory to allocate for the region, in bytes. The memory size can be larger than the region’s actual size in the object file.
Region flags. Currently these flags are used only to specify the type of address for the region, but others may be added in the future. Possible values are as follows:

- **REG_ABS_ADDR_F**
  The region has an absolute address, as specified by `regc_addr.vm_addr`.

- **REG_REL_ADDR_F**
  The region has a relative address, as specified by `regc_addr.rel_addr`.

- **NULL**
  The region's address will be assigned by the loader.

The index in the load command map of the load command for the associated relocation section, if there is one. If there is no associated relocation section, this field is set to `MO_INVALID_LCID`. See "Relocation Information," later in this reference page.

The alignment in bytes that is required for the data contained in the region to be referenced properly by the hardware. For example, a region that contains double precision floating point numbers may have to be aligned at least on a four-byte boundary. This field is used by the linker when combining regions from several object files into a single region for an executable file.

The region's usage type (that is, text, data, bss, and so on). The values for this field are machine dependent and are defined in the `mach_o_types.h` file.

The protection for the region. The loader must translate these generic protection flags (from the `mach_o_format.h` file) into the actual protection values used by the system:

- **MO_PROT_NONE**
- **MO_PROT_READ**
- **MO_PROT_WRITE**
- **MO_PROT_EXECUTE**

A region's address can be specified as absolute, relative to another region, or not specified at all. If a region has an absolute address, it is loaded at that address. It cannot be relocated to a different virtual address. However, it may require that
another region be relocated to resolve external references to addresses in that
region. If a region has a relative address, it must indicate the region that it is
relative to. The load command for the (nonrelative) region being referenced must
have a lower load command map index than the (relative) region load commands
that reference it. If a region has no virtual address specified, the loader assigns one
for it.

The memory size specified for a region can be larger than the region's actual size
in the file. Memory in excess of the region's actual size is allocated zero fill on
demand. For a bss (uninitialized data) region, the region's load command might
not have an associated section in the file.

For executable (that is, nonlinkable) files, a region's offset in the file must be page
aligned. The page boundary defines the finest granularity for protection when
mapping into virtual memory. In other words, two parts of the same page cannot
be mapped with different protections.

The `regc_region_name`, `regc_addralign`, and `regc_usage_type` fields are used only
by the linker.

The `regc_usage_type` field serves to distinguish text, data, read-only data, and so
forth. It allows the linker to combine similar region types from multiple object
files. It is the linker's responsibility to combine \( n \) types of input regions into \( m \)
types of output regions. Region usage, though, is also affected by the protection
attribute. At a minimum, there must be a different region for each kind of
protection attribute. One possible strategy is to combine regions first by usage type
and then combine the resulting regions by protection attribute.

The `regc_addralign` field allows the linker to ensure that each region in a page-
aligned composite region is aligned properly for the data that the region contains.

Packages

The OSF/I loader implements a two-dimensional name space for managing symbol
resolution. The first dimension consists of named abstractions called **packages**.
The second dimension consists of named symbols.

If an object module references a symbol (which can be a routine or a data item) in
another module, the referenced symbol is said to be **imported**. If an object module
makes a symbol available for referencing by other modules, the symbol is said to
be **exported**. **Symbol resolution** is the process of binding each imported symbol
to the address of a corresponding exported symbol.

Packages are containers for symbols. Package names must be unique across a
system. Symbol names must be unique within a package. Several exported
symbols, then, can have the same name as long as they are associated with different
packages.
The **OSF/ROSE** object format supports the use of the 
*(package_name, symbol_name)* pair for symbol references. At load time, the 
OSF/1 loader resolves the package names in the object file to the file pathnames of 
the appropriate libraries (modules) by searching a hierarchy of package tables. A 
system-wide package table contains the default package to library mappings. A set 
of per-process tables contain mappings specific to a given process. The loader 
searches the per-process tables first and then the system-wide table.

The **OSF/ROSE** format implements separate package lists for imported and 
exported symbols. Packages are implemented separately from the symbol 
references because they consist of more than just package names. A reference (via 
the symbol load command) specifies its associated package by using the index of 
the package in the appropriate package list.

In OSF/1, the package lists are generated by the linker and not by the compiler or 
assembler. Thus exported and imported symbols in unlinked object files need not 
have package names. (This is likely to change in future releases.) The OSF/1 
Release 1.0 linker does not use package names in its own symbol resolution policy.

Package Entries

The package load command does not have an associated section. It ends in a 
variable-length array. The array entries are described by the following structure 
declaration from the **mach_o_format.h** file:

```c
typedef struct pkg_entry_t {
    mo_offset_t pe_pkg_name;
    mo_addr_t pe_version_addr;
} pkg_entry_t;
```

The package entry fields are defined as follows:

* **pe_pkg_name** The package name, specified as an offset into the strings section 
whose map index is *pkgc_strings_id*.

* **pe_version_addr** The address of information describing the version of the package 
referenced during linking. (Not used in OSF/1.)
Package Load Command

There must be a separate package load command for the import package list and the export package list. In each case the structure is the same, with the value of the pkgc_flags field determining the type of list. The package load command is described by the following structure declaration from the mach_o_format.h file:

```c
typedef struct package_command_t {
    ldc_header_t ldc_header;
    mo_short_t pkgc_flags;
    mo_short_t pkgc_nentries;
    mo_lcid_t pkgc_strings_id;
    pkg_entry_t pkgc_pkg_list[l];
} package_command_t;
```

The package load command fields are defined as follows:

- **ldc_header**: The load command header for the package load command. In this structure, the ldci_cmd_type field must be set to LDC_PACKAGE. The section offset and length fields must be set to 0 (zero).

- **pkgc_flags**: The type of package list. Possible values are as follows:
  - PKG_EXPORT_F
  - PKG_IMPORT_F

- **pkgc_nentries**: The number of entries in the package list.

- **pkgc_strings_id**: The index of a strings load command in the load command map. The index is used to establish a link to the strings section that contains the package name strings.

- **pkgc_pkg_list**: The variable-length array that contains the packages (type pkg_entry_t) in the list.

Symbols

The OSF/ROSE format provides a single load command for all symbol types. The symbol load command requires an associated section. An object file can have several different kinds of symbol sections, as determined by the symc_kind field. Individual symbols are defined in their respective sections using the symbol_info_t structure supplied with the OSF/ROSE format.
The OSF/ROSE format supports the following types of symbols:

- **Defined Symbols** — Identify locations or actual values within the object file. Some of these symbols may be exported (that is, available to other object files). They have the export flag set. Exported symbols in executable files also have package names. (A package name is specified as an index into the export package list.) Defined symbols that are not exported have the value MO_INVALID_PKG_INDEX in their package_index fields.

- **Imported Symbols** — Reference symbols in other modules. These symbols represent a module's unresolved references. Like exported symbols, imported symbols in executable files have associated package names (specified as indexes into the import package list.) The OSF/1 assembler identifies each imported symbol as being either code or data. This distinction can be used by the loader to allow lazy evaluation of unresolved code references (although the OSF/1 loader does not do this).

- **Stab Symbols** — Provide information for use by symbolic debuggers. Most of the information is stored as part of the name string, but several type fields (used only for stabs) are used as well. The OSF/1 symbolic debugger uses this format. Other vendors may wish to use their own format for debugging information.

### Symbol Entries

Each entry in a symbol section is described by the following structure declaration from the mach_o_format.h file:

```c
typedef struct symbol_info_t {
    union {mo_offset_t symbol_name;
        mo_ptr_t symbol_nameP;
    } si_name;
    mo_short_t si_package_index;
    mo_short_t si_type;
    mo_short_t si_flags;
    mo_byte_t si_reserved_byte;
    mo_byte_t si_sc_type;
    union { mo_addr_t def_val;
        mo_long_t imp_val;
        mo_long_t lit_val;
        mo_vm_addr_t abs_val;
    } si_value;
} symbol_info_t;
```
The symbol entry fields are defined as follows:

\texttt{si\_name}

The name of the symbol. The symbol name is defined by a union and can be specified either as an offset into the associated strings section (\texttt{si\_name.symbol\_name}) or as a pointer (\texttt{si\_name.symbol\_nameP}). The pointer form should be used only when \texttt{symbol\_info\_t} is used as part of a runtime data structure such as in the linker -- and never in an object file.

\texttt{si\_package\_index}

The index of the associated package within the appropriate package list. This field is used only for exported and imported symbols. Otherwise, its value is \texttt{MO\_INVALID\_PKG\_INDEX}, defined in the \texttt{mach\_o\_types.h} file.

\texttt{si\_type}

Encoded type information for debug symbols. (The encoding used in OSF/1 Release 1.0 was "inherited" for easier porting and is not defined here.)

\texttt{si\_flags}

Flags describing the symbol or how it is used. Possible values are as follows:

\texttt{SI\_EXPORT\_F}

The defined symbol is exported (that is, made visible to other object files).

\texttt{SI\_IMPORT\_F}

The symbol is imported (that is, its value is defined in another object file). The \texttt{si\_value.imp\_val} field contains the index of this \texttt{symbol\_info\_t} structure in the associated import list.

\texttt{SI\_LOCAL\_F}

The defined symbol is local (that is, it is not visible outside the object file).

\texttt{SI\_CODE\_F}

The imported symbol is referenced via calls.

\texttt{SI\_DATA\_F}

The imported symbol is referenced via data references.

\texttt{SI\_LITERAL\_F}

The \texttt{si\_value} field contains the actual value of the symbol, and not just the address.

\texttt{SI\_FORWARD\_F}

The value of this symbol is the address of another symbol, which contains the "real" value. (The OSF/1 compiler tools do not currently use this flag.)
SI_COMMON_F
The data represented by the symbol will live in the common area and so the value field is not used to find the address. Instead, if the symbol is defined, si_value.lit_val contains the symbol's size in bytes.

SI_LOCAL_LABEL_F
The symbol represents a local (internal) label. Use of this flag makes it unnecessary to use special naming conventions to identify such labels.

SI_ABSOLUTE_VALUE_F
The value of this defined symbol is an absolute virtual address, referenced as si_value.abs_val.

si_reserved_byte
Reserved for future use (possibly more flags) and should not be used for such things as more debugging information.

si_sc_type
Storage class type information. As with si_type, the values used by OSF/1 were inherited and are not described in this document.

si_value
The symbol's value. This field is used in conjunction with the si_flags fields. The symbol value is defined by a union.

If SI_IMPORTED_F is set, si_value.imp_val is used and contains this symbol's index in the import section.

If SI_LITERAL_F is set, si_value.lit_val is used and contains either the value itself (as opposed to an address), or the size of the symbol.

If SI_ABSOLUTE_VALUE_F is set, si_value.abs_val is used and contains an absolute virtual memory address.

If none of the above flags is set, si_value.def_val is used and contains an address as an offset within an object file section.
Symbol Load Command

The symbol load command is described by the following structure declaration from the `mach_o_format.h` file:

```c
typedef struct symbols_command_t {
    ldc_header_t ldc_header;
    mo_short_t symc_kind;
    mo_short_t symc_short_reserved;
    mo_long_t symc_nentries;
    mo_lcid_t symc_pkg_list;
    mo_lcid_t symc_strings_section;
    mo_lcid_t symc_reloc_addr;
    union { mo_short_t n_exported_symb;
        mo_long_t long_reserved;
    } symc_other;
} symbols_command_t;
```

The symbol load command fields are defined as follows:

- **ldc_header**: The load command header for the symbol load command. In this structure, the `ldci_cmd_type` field must be set to LDC_SYMBOLS. The section offset and length fields must be filled in.

- **symc_kind**: The kind of symbol section associated with the load command. Possible values are as follows:
  - SYMC_IMPORTS: The section contains imported symbols.
  - SYMC_DEFINED_SYMBOLS: The section contains defined, and possibly exported, symbols.
  - SYMC_STABS: The section contains stab symbols (for use by symbolic debuggers).

- **symc_short_reserved**: Reserved for future use.

- **symc_nentries**: The number of `symbol_info_t` entries in the associated symbols section.

- **symc_pkg_list**: The index of a package load command in the load command map. The package load command contains the associated package list.
**symc_strings_section**

The index of a strings load command in the load command map. The strings load command identifies the strings section that contains the symbol names.

**symc_reloc_addr**

The index in the load command map of the load command for the associated relocation section, if there is one. If there is no associated relocation section, this field is set to MO_INVALID_LCID. See "Relocation Information," later in this reference page.

**symc_other**

Used for additional, kind-related information. For defined symbol sections, this field contains the number of exported symbols. This information enables programs that are only interested in exported symbols to avoid having to look at every symbol entry. (Although it is preferable to put all the exported symbols first in the section, it is not required.)

**Relocation Information**

Relocation is the process of modifying references so that they reflect the actual addresses of the entities they reference.

An object file can contain references with incomplete or missing addresses. The linker must modify such references as regions are moved and external functions become internal. The loader must also modify such references when virtual addresses are assigned. The OSF/ROSE format specifies a separate relocation section for each region or symbol section containing references that need to be adjusted. The relocation load command serves as the header for a relocation section. A region's incomplete references are defined (via the reloc_info_t structure) as entries in the relocation section associated with the region.

The OSF/ROSE format provides two methods for relocation entries to specify the referenced locations:

- **Symbol Relative** — With this method, the referenced location has a name and the relocation entry "points to" a symbol information structure. The symbol information structure connects the symbol name with the location.

- **Location Relative** — With this method, the relocation entry specifies the referenced location explicitly. Location-relative relocation can be used only for references within the same object file and is provided primarily as an optimization.

Because different kinds of references may need to be relocated, the relocation entries also specify how the referencing fields are to be updated. The update information is contained in the type field for each relocation entry. The values of the relocation types are machine dependent.
Each relocation entry contains a set of flags whose general purpose is to specify how the target address is to be interpreted. Some of the flags indicate the type of target address specification to use. Other flags indicate extra processing that must be done to a target address before updating the referencing field.

**Note:** The following algorithm is not implemented in OSF/1 Release 1.0. Instead, it describes now the indirect flag is intended to be used.

The indirect flag, in particular, is intended to be used for external or long-distance references in position-independent-code (PIC) programs. It indicates that the linker must generate an address constant in the data table (also known as the program table or table of contents). The linker is responsible for building the data table from those relocation entries that have the indirect flag set. When the linker finds a relocation entry that has the indirect flag set, it performs the following steps:

1. It generates an address constant for the referenced location and inserts this constant as an entry in the data table. If the address constant entry already exists in the data table, the linker does not duplicate it.

2. It relocates the referencing field using the address or offset of the address constant entry in the data table.

3. It moves the relocation entry from the referencing field to the corresponding address constant entry in the data table. The linker turns off the indirect flag in the moved relocation entry and assigns a different relocation type.

The linker follows this procedure even if the assembler generates data tables. In this case, the linker fabricates a new data table as above and ignores or discards the data tables produced by the assembler.

**Relocation Entries**

Each entry in a relocation section is described by the following structure declaration from the `mach_o_format.h` file:

```c
typedef struct reloc_info_t {
    mo_offset_t     ri_roffset;
    mo_short_t      ri_flags;
    mo_short_t      ri_size_type;
    union { mo_index_t symbol_index;
            mo_addr_t  loc_addr; }
        ri_value;
} reloc_info_t;
```

The relocation entry fields are defined as follows:

- **ri_roffset** The offset from the beginning of the section of the first byte to be relocated.
ri_flags  Indicators specifying extra processing or how ri_value is to be interpreted. Possible values are as follows:

RI_PC_REL_F  Derive the relocated value as the difference between the address of the location being relocated and the address of the location being referenced.

RI_INDIRECT_F  Interpret ri_offset as a reference to an address constant entry in the linker-generated data table.

RI_SYMBOL_F  Interpret ri_value as a symbol table index.

RI_LOC_F  Interpret ri_value as an internal address within the object file.

ri_size_type  An indicator specifying how the referencing field is to be updated. Values are machine dependent.

ri_value  The referenced location. The location value is defined by a union. For a symbol-relative reference, ri_value.symbol_index specifies the symbol being referenced. For a location-relative reference, ri_value.loc_addr specifies the actual location within the same object file.

Relocation Load Command

The relocation load command is described by the following structure declaration from the mach_o_format.h file:

typedef struct reloc_command_t {
  ldc_header_t ldc_header;
  mo_long_t relc_nentries;
  mo_lcid_t relc_owner_section;
  mo_long_t relc_reserved;
} reloc_command_t;

The relocation load command fields are defined as follows:

ldc_header  The load command header for the relocation load command. In this structure, the ldci_cmd_type field must be set to LDC_RELOC. The section offset and length fields must be filled in.

relc_nentries  The number of relocation entries in the associated section.
relc_owner_section
The index in the load command map of the load command for the
region or section being relocated.

relc_reserved  Reserved for future use.

Strings
The strings sections contain the strings referenced by the load commands and the
object file’s meta data. The OSF/ROSE format supports multiple strings sections.
This allows strippable strings, such as those used by debug symbols, to be clearly
separated from non-strippable strings, such as those used by the load commands. It
also allows load commands and sections that have their own strings sections to be
added or replaced more easily. (The OSF/l compiler tools do not fully support
multiple strings sections.)

String references are of type mo_addr_t. In other words, they specify the index of
a strings load command in the load command map and an offset into the associated
strings section.

The entries in a strings section are null-terminated strings.

The strings load command is described by the following structure declaration from
the mach_o_format.h file:

typedef struct strings_command_t {
    ldc_header_t ldc_header;
    mo_long_t strc_ftags;
    } strings_command_t;

The strings load command fields are defined as follows:

ldc_header  The load command header for the strings load command. In this
structure, the ldc_cmd_type field must be set to LDC_STRINGS. The
section offset and length fields must be filled in.

strc_ftags  None are defined currently. These flags could indicate such things
as use of multiple-byte encoding, compression, strings that are
preceded by a count, and so on.

Program Main Entry
The OSF/ROSE format provides the entry load command for specifying the main
entry point in a program. An object file can contain only one entry load command.

If the module is to be loaded by the kern_exec() function, the absolute address
field (entc_absaddr) is required. Otherwise, it is optional.
The entry load command is described by the following structure declaration from the `mach_o_format.h` file:

```c
typedef struct entry_command_t {
    ldc_header_t ldc_header;
    mo_short_t entc_flags;
    mo_short_t entc_short_reserved;
    mo_vm_addr_t entc_absaddr;
    mo_addr_t entc_entry_pt;
} entry_command_t;
```

The entry load command fields are defined as follows:

- **ldc_header**
  The load command header for the entry load command. In this structure, the `ldci_cmd_type` field must be set to `LDC_MAIN_ENTRY`. The section offset and length fields must be set to 0 (zero).

- **entc_flags**
  Entry flags. The only currently defined value is as follows:
  
  ENT_VALID_ABSADDR_F
  The `entc_absaddr` field has a valid address value.

- **entc_short_reserved**
  Reserved for future use.

- **entc_absaddr**
  The absolute address of the entry point in virtual memory. This field is required for loading by the `kern_exec()` function.

- **entc_entry_pt**
  The address of the entry point within the object file, expressed as the load command map index of a region load command along with an offset.

**Generation Information**

The OSF/ROSE format provides a generation information load command. It can be used to embed the following pieces of information in the object file:

- The creation date and time for the object file.
- The name of the assembler or linker used to create the object file.
- The version of the assembler or linker used to create the object file.
- The build time stamp of the assembler or linker used to create the object file.
- The options specified to the assembler or linker.

The generation information load command does not have an associated section. Parts of the information, though, are stored as strings in the strings section used by the load commands. As such, these strings are not strippable.
Currently, the OSF/1 linker retains no generation information from the component object files. It puts only information about itself in this load command.

The generation information load command is described by the following structure declaration from the `mach_o_format.h` file:

```c
typedef struct gen_info_command_t {
    ldc_header_t ldc_header;
    time_t genc_obj_create_time;
    mo_addr_t genc_creator_name;
    mo_addr_t genc_creator_version;
    time_t genc_creator_time;
    mo_addr_t genc_options_to_creator;
} gen_info_command_t;
```

The generation information load command fields are defined as follows:

- **ldc_header**: The load command header for the generation information load command. In this structure, the `ldci_cmd_type` field must be set to `LDC_GEN_INFO`. The section offset and length fields must be set to 0 (zero).

- **genc_obj_create_time**: The date and time at which the object file was created.

- **genc_creator_name**: The name of the assembler or linker that created the object file.

- **genc_creator_version**: The version of the assembler or linker that created the object file.

- **genc_creator_time**: The build time stamp of the assembler or linker that created the object file.

- **genc_options_to_creator**: The options specified to the assembler or linker.

**Function Tables**

The OSF/ROSE format provides a separate function table load command. This load command establishes an array of function entry points that are to be called outside of the main flow of the program, usually by the loader.

This load command is supported by the OSF/1 loader, but is not generated by the OSF/1 compiler tools.

Sets of related functions are grouped by means of the type field. The type field indicates when the functions in the associated table are to be called. Currently, two function types are supported: initialization and termination. Initialization functions
are called when the object file is loaded. Termination functions are called when the file is unloaded. As a rule, an object file should contain no more than one table of each type.

Since the function entry points are stored in an array, each component of a linked module can have its own set of functions. The array format also allows the function calls to be ordered.

The function table load command does not have an associated section. The functions are treated as part of the program and are located in regions, either separately or in combination with other code. The load command specifies only the region and offset of each function’s entry point.

The function table load command is described by the following structure declaration from the `mach_o_format.h` file:

```
typedef struct func_table_command_t {
    ldc_header_t ldc_header;
    mo_short_t fntc_type;
    mo_short_t fntc_nentries;
    mo_addr_t fntc_table_name;
    ino_long_t fntc_reserved;
    mo_addr_t fntc_entry_loc [1];
} func_table_command_t;
```

The function table load command fields are defined as follows:

- **ldc_header** The load command header for the function table load command. In this structure, the `ldci_cmd_type` field must be set to LDC_FUNC_TABLE. The section offset and length fields must be set to 0 (zero).

- **fntc_type** The type of functions in the associated table. Currently defined values are as follows:

  FNTC_INITIALIZATION  
  For functions to be called when the module is loaded

  FNTC_TERMINATION  
  For functions to be called when the module is unloaded

- **fntc_nentries** The number of functions in the associated table.

- **fntc_table_name** The name of the function table, specified as a string address.
Reserved for future use.

An array of the function addresses. Functions are implemented in regions. Each function is identified by the ID of its region’s load command and its offset into that region’s section.

Program Interpreter

If an OSF/ROSE object file uses a program interpreter, the interpreter load command must be the first entry in the load command map. The interpreter load command is designed to have no dependencies on other load commands. It has no associated section. This load command is not supported by the OSF/1 compiler tools or loader.

The interpreter load command is intended to be used when the object file is loaded. The loader would retrieve the pathname for the interpreter and create the initial process image using the interpreter file’s regions, rather than the object file’s regions. The interpreter would be responsible for receiving control from the system and establishing an environment for the program.

The interpreter load command is described by the following structure declaration from the mach_o_format.h file:

```c
typedef struct interpreter_command_t {
    ldc_header_t ldc_header;
    char intc_interpreter_path [1];
} interpreter_command_t;
```

The interpreter load command fields are defined as follows:

- `ldc_header` The load command header for the interpreter load command. In this structure, the `ldci_cmd_type` field must be set to LDC_INTERPRETER. The section offset and length fields must be set to 0 (zero).

- `intc_interpreter_path` A null-terminated string that specifies the pathname for the interpreter.

Related Information

Functions: `decode_mach_o_hdr(3)`, `encode_mach_o_hdr(3)`
passwd

Purpose  Password files

Description

A passwd file is a file consisting of records separated by newline characters, one record per user, containing ten colon (:) separated fields. These fields are as follows:

- **name**: User’s login name
- **password**: User’s encrypted password
- **uid**: User’s ID
- **id**: User’s login group ID
- **class**: User’s general classification (unused)
- **change**: Password change time
- **expire**: Account expiration time
- **gecos**: General information about the user
- **home_dir**: User’s home directory
- **shell**: User’s login shell

The *name* field is the login used to access the computer account, and the *uid* field is the number associated with it. They should both be unique across the system (and often across a group of systems) since they control file access.

While it is possible to have multiple entries with identical login names and/or identical user id’s, it is usually a mistake to do so. Routines that manipulate these files will often return only one of the multiple entries, and that one by random selection.

The login name must never begin with a hyphen (-); also, it is strongly suggested that neither uppercase characters or dots (.) be part of the name, as this tends to confuse mailers. No field may contain a colon (:) as this has been used historically to separate the fields in the user database.

The password field is the encrypted form of the password. If the *password* field is empty, no password is required to gain access to the machine. Because these files contain the encrypted user passwords, they should not be readable by anyone without appropriate privileges.
The *gid* field is the group that the user will be placed in upon login. Since OSF/1 supports multiple groups (see the *groups* command) this field currently has little special meaning.

The *class* field is currently unused. In the near future it will be a key to a *termcap* style database of user attributes.

The *change* field is the number in seconds, Coordinated Universal Time (CUT), from the epoch until the password for the account must be changed. This field may be left empty to turn off the password aging feature.

The *expire* field is the number in seconds, Coordinated Universal Time, from the epoch until the account expires. This field may be left empty to turn off the account aging feature.

The *gecos* field normally contains comma (,) separated subfields as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>User’s full name</td>
</tr>
<tr>
<td>office</td>
<td>User’s office number</td>
</tr>
<tr>
<td>wphone</td>
<td>User’s work phone number</td>
</tr>
<tr>
<td>hphone</td>
<td>User’s home phone number</td>
</tr>
</tbody>
</table>

This information is used by the *finger* command.

The user’s home directory is the full UNIX pathname where the user will be placed on login.

The *shell* field is the command interpreter the user prefers. If the *shell* field is empty, the Bourne shell (/bin/sh) is assumed.

### Related Information

- Functions: *getpwent*(3)
- Commands: *login*(1), *passwd*(1)
protocols

Purpose
Protocol name database

Description
The protocols file contains information regarding the known protocols used in the DARPA Internet. For each protocol, the file should contain a single line with the following information:

- Official protocol name
- Protocol number
- Aliases

Items are separated by any number of blanks, tab characters, or both. A # (number sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Protocol names may contain any printable character other than a field delimiter, newline, or comment character.

Files
/etc/protocols

Related Information
Functions: getprotoent(3)
pty

Purpose  Pseudo terminal driver

Synopsis  pseudo-device pty [ count ]

Description
The pty driver provides support for a device-pair termed a pseudo terminal. A pseudo terminal is a pair of character devices, a master device and a slave device. The slave device provides an interface identical to that described in the tty reference page. However, whereas all other devices which provide the interface described in the tty reference page have a hardware device behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, anything written on the master device is given to the slave device as input and anything written on the slave device is presented as input on the master device.

In configuring, if an optional "count" is given in the specification, that number of pseudo terminal pairs are configured; the default count is 32.

The following ioctl calls apply only to pseudo terminals:

TIOCSTOP  Stops output to a terminal (for example, like entering <ctrl-S>).
Takes no parameter.

TIOCSTART  Restarts output (stopped by TIOCSTOP or by typing <ctrl-S>).
Takes no parameter.

TIOCPKT  Enable or disable packet mode. Packet mode is enabled by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. When applied to the master side of a pseudo terminal, each subsequent read( ) from the terminal will return data written on the slave part of the pseudo terminal preceded by a zero byte (symbolically defined as TIOCPKT_DATA), or a single byte reflecting control status information. In the latter case, the byte is an inclusive-OR of zero or more of the bits:

TIOCPKT_FLUSHREAD
Whenever the read queue for the terminal is flushed.

TIOCPKT_FLUSHWRITE
Whenever the write queue for the terminal is flushed.
TIOCPKT_STOP
Whenever output to the terminal is stopped by <ctrl-S>.

TIOCPKT_START
Whenever output to the terminal is restarted.

TIOCPKT_DOSTOP
Whenever t_stopc is <ctrl-S> and t_startc is <ctrl-Q>.

TIOCPKT_NOSTOP
Whenever the start and stop characters are not <ctrl-S> and <ctrl-Q>.

While this mode is in use, the presence of control status information to be read from the master side may be detected by a select() for exceptional conditions.

This mode is used by the rlogin and rlogind commands to implement a remote-echoed, locally <ctrl-S>/<ctrl-Q> flow-controlled remote login with proper back-flushing of output; it can be used by other similar programs.

TIOCUCNTL Enable or disable a mode that allows a small number of simple user ioctl commands to be passed through the pseudo-terminal, using a protocol similar to that of TIOCPKT. The TIOCUCNTL and TIOCPKT modes are mutually exclusive. This mode is enabled from the master side of a pseudo terminal by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. Each subsequent read() from the master side will return data written on the slave part of the pseudo terminal preceded by a zero byte, or a single byte reflecting a user control operation on the slave side. A user control command consists of a special ioctl operation with no data; the command is given as UIOCCMD(n), where n is a number in the range 1-255. The operation value n will be received as a single byte on the next read() from the master side. The ioctl UIOCCMD(0) is a no-op that may be used to probe for the existence of this facility. As with TIOCPKT mode, command operations may be detected with a select() for exceptional conditions.
TIOCREMOTE

A mode for the master half of a pseudo terminal, independent of TIOCPKT. This mode causes input to the pseudo terminal to be flow controlled and not input edited (regardless of the terminal mode). Each write to the control terminal produces a record boundary for the process reading the terminal. In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 (zero) bytes is like typing an End-of-File character. The TIOCREMOTE mode can be used when doing remote line editing in a window manager, or whenever flow controlled input is required.

Files

/dev/tty[p-r][0-9a-f]
   Master pseudo terminals

/dev/tty[p-r][0-9a-f]
   Slave pseudo terminals
resolver

Purpose  Resolver configuration file

Description

The resolver is a set of routines in the C library that provide access to the Internet Domain Name System. The resolver configuration file contains information that is read by the resolver routines the first time they are invoked by a process. The file is designed to be read by humans and contains a list of keywords with values that provide various types of resolver information.

On a normally configured system this file should not be necessary. The only name server to be queried will be on the local machine, the domain name is determined from the hostname, and the domain search path is constructed from the domain name.

The different configuration options are:

**nameserver**  Internet address (in dot notation) of a name server that the resolver should query. Up to MAXNS (currently 3) name servers may be listed, one per keyword. If there are multiple servers, the resolver library queries them in the order listed. If no nameserver entries are present, the default is to use the name server on the local machine. (The algorithm used is to try a name server, and if the query times out, try the next, until out of name servers, then repeat trying all the name servers until a maximum number of retries are made).

**domain**  Local domain name. Most queries for names within this domain can use short names relative to the local domain. If no domain entry is present, the domain is determined from the local hostname returned by `gethostname()`; the domain part is taken to be everything after the first . (dot). Finally, if the hostname does not contain a domain part, the root domain is assumed.

**search**  Search list for hostname lookup. The search list is normally determined from the local domain name; by default, it begins with the local domain name, then successive parent domains that have at least two components in their names. This may be changed by listing the desired domain search path following the search keyword with spaces or tabs separating the names. Most resolver
queries will be attempted using each component of the search path in turn until a match is found. Note that this process may be slow and will generate a lot of network traffic if the servers for the listed domains are not local, and that queries will time out if no server is available for one of the domains.

The search list is currently limited to six domains with a total of 256 characters.

The **domain** and **search** keywords are mutually exclusive. If more than one instance of these keywords is present, the last instance will override.

The keyword and value must appear on a single line, and the keyword (e.g. **nameserver**) must start the line. The value follows the keyword, separated by white space.

**Files**

`/etc/resolv.conf`

**Related Information**

Functions: `gethostbyname(3)`, `res_mkquery(3)`, `res_send(3)`, `res_init(3)`, `dn_comp(3)`, `dn_expand(3)`

Files: `hostname(5)`

Commands: `named(8)`
ROUTE

Purpose
Kernel packet forwarding database

Synopsis
#include <sys/socket.h>
#include <net/if.h>
#include <net/route.h>

int family
s = socket(PF_ROUTE, SOCK_RAW, family);

Description
The UNIX operating system provides packet routing facilities. The kernel maintains a routing information database, which is used in selecting the appropriate network interface when transmitting packets.

A user process (or possibly multiple cooperating processes) maintains this database by sending messages over a special kind of socket. Routing table changes may only be carried out by the superuser.

The operating system may spontaneously emit routing messages in response to external events, such as receipt of a redirect, or failure to locate a suitable route for a request.

Routing database entries are of two types: those for a specific host, and those for all hosts on a generic subnetwork (as specified by a bit mask and value under the mask). The effect of a wildcard or default route may be achieved by using a mask of all zeros. There may be hierarchical routes.

When the system is booted and addresses are assigned to the network interfaces, each protocol family installs a routing table entry for each interface when it is ready for traffic. Normally the protocol specifies the route through each interface as a "direct" connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests that the packet be sent to the host specified in the packet. Otherwise, the interface is requested to address the packet to the gateway listed in the routing entry (that is, the packet is forwarded).

When routing a packet, the kernel first attempts to find a route to the destination host. Failing that, a search is made for a route to the network of the destination. Finally, any route to a default (wildcard) gateway is chosen. If no entry is found, the destination is declared to be unreachable, and an error message is generated if there are any listeners on the routing control socket described later in this section.

A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the
destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

To open the channel for passing routing control messages, use the socket call shown in the SYNOPSIS section.

The family parameter may be AF_UNSPEC which will provide routing information for all address families, or can be restricted to a specific address family by specifying which one is desired. There can be more than one routing socket open per system.

Messages are formed by a header followed by a small number of sockadders (now variable length), interpreted by position, and delimited by the new length entry in the sockaddr. An example of a message with four addresses might be an ISO redirect: destination, netmask, gateway, and author of the redirect. The interpretation of which addresses are present is given by a bit mask within the header, and the sequence is least significant to most significant bit within the vector.

Any messages sent to the kernel are returned, and copies are sent to all interested listeners. The kernel will provide the process ID for the sender, and the sender may use an additional sequence field to distinguish between outstanding messages. However, message replies may be lost when kernel buffers are exhausted.

The kernel may reject certain messages, and will indicate this by filling in the rtm_errno field. In the current implementation, all routing process run locally, and the values for rtm_errno are available through the normal errno mechanism, even if the routing reply message is lost.

A process may avoid the expense of reading replies to its own messages by issuing a setsockopt() call indicating that the SO_USELOOPBACK option at the SOL_SOCKET level is to be turned off. A process may ignore all messages from the routing socket by shutting down further input with the shutdown() function.

If a route is in use when it is deleted, the routing entry will be marked down and removed from the routing table, but the resources associated with it will not be reclaimed until all references to it are released. User processes can obtain information about the routing entry to a specific destination by using a RTM_GET message, or by reading the /dev/kmem device.

Errors

If messages are rejected, rtm_errno may be set to one of the following values:

[EEXIST] The entry to be created already exists.
[ESRCH] The entry to be deleted does not exist.
[ENOBUFS] Insufficient resources were available to install a new route.
semid_ds

Purpose
Defines a semaphore set

Synopsis
#include <sys/sem.h>

struct semid_ds {
    struct ipc_perm sem_perm;
    struct sem *sem_base;
    u_short sem_nsems;
    time_t sem_otime;
    time_t sem_ctime;
};

Description
The semid_ds structure defines a semaphore set associated with a semaphore ID. There is one semaphore set per semaphore ID.

A semaphore set is implemented as an array of sem_nsems semaphores, with sem_base pointing to the first semaphore in the set.

The IPC permissions for a semaphore set are implemented in a separate, but associated, ipc_perm structure.

A semaphore set is created indirectly via the semget() call. If semget() is called with a non-existent semaphore ID, the kernel allocates a new semid_ds structure, initializes it, and returns the semaphore ID that is to be associated with the semaphore set.

Fields

- **sem_perm**: The ipc_perm structure that defines permissions for semaphore operations. See NOTES.
- **sem_base**: A pointer to the first semaphore in the set. Individual semaphores are defined using the sem structure. See NOTES.
- **sem_nsems**: The number of semaphores in the set. Each semaphore in the set is referenced by a unique integer. A semaphore number is sometimes referred to as sem_num, but this is not a field carried in any of the relevant data structures. Semaphore numbers run sequentially from zero to sem_nsems-1.
- **sem_otime**: The time of the last semop() operation on the set.
- **sem_ctime**: The time of the last semctl() operation that changed a semaphore in the set.
The `sem_perm` field identifies the associated `ipc_perm` structure that defines the permissions for operations on the semaphore set. The `ipc_perm` structure (from the `sys/ipc.h` header file) is shown here.

```c
struct ipc_perm {
    ushort uid;    /* owner's user id */
    ushort gid;    /* owner's group id */
    ushort cuid;   /* creator's user id */
    ushort cgid;   /* creator's group id */
    ushort mode;   /* access modes */
    ushort seq;    /* slot usage sequence number */
    key_t key;     /* key */
};
```

The `mode` field is a 9-bit field that contains the permissions for semaphore operations. The first three bits identify owner permissions; the second three bits identify group permissions; and the last three bits identify other permissions. In each group, the first bit indicates read permission; the second bit indicates write permission; and the third bit is not used.

Individual semaphores are implemented with the `sem` structure. The `sem` structure (from the `sys/sem.h` header file) is shown here:

```c
struct sem {
    u_short semval;
    short sempid;
    u_short semncnt;
    u_short semzcnt;
};
```

The `sem` fields are defined as follows:

- `semval` A nonnegative integer that is the current value of the semaphore.
- `sempid` The process ID of the last process to perform an operation on the semaphore.
semncnt  The number of processes that are currently suspended while waiting for an operation to increment the current semval value.

semzcnt  The number of processes that are currently suspended while waiting for semval to go to zero.

Related Information

Functions: semctl(2), semget(2), semop(2)
services

Purpose  Service name database

Description  The /etc/services file contains information regarding the known services available in the DARPA Internet. For each service, the file should contain a single line with the following information:

- Official service name
- Port number
- Protocol name
- Aliases

Items are separated by any number of blanks, tab characters, or both. The port number and protocol name are considered a single item; a / (slash) is used to separate the port and protocol (for example, 512/tcp). A # (number sign) indicates the beginning of a comment; subsequent characters up to the end of the line are not interpreted by the routines which search the file.

Service names may contain any printable character other than a field delimiter, newline, or comment character.

Files

/etc/services

Related Information

Functions: getservent(3)
shells

Purpose
Shell database

Description
The shells file contains a list of the shells on the system. For each shell, the file should contain a single line consisting of the shell’s path, relative to root.

A # (number sign) indicates the beginning of a comment; subsequent characters up to the end of the line are not interpreted by the routines which search the file. Blank lines are also ignored.

Files
/etc/shells

Related Information
Functions: getusershell(3)
shmid_ds

Purpose
Defines a shared memory region

Synopsis
#include <sys/shm.h>

struct shmid_ds{
    struct ipc_perm shm_perm;
    int shm_segsz;
    u_short shm_lpid;
    u_short shm_cpid;
    u_short shm_nattch;
    time_t shm_atime;
    time_t shm_dtime;
    time_t shm_ctime;
};

Description

The shmid_ds structure defines a shared memory region associated with a shared memory region ID. There is one shared memory region per ID. Collectively, the shared memory regions are maintained in a shared memory table, with the shared memory region IDs identifying the entries in the table.

The IPC permissions for the shared memory regions are implemented in a separate, but associated, ipc_perm structure.

A shared memory region is created indirectly via the shmget() call. If shmget() is called with a non-existent shared memory region ID, the kernel allocates a new shmid_ds structure, initializes it, and returns the ID that is to be associated with the region.

The kernel allocates actual memory of shm_segsz bytes only when a process attaches a region to its address space. Attached regions are maintained in a separate region table. The entries in the shared memory table point to the associated attached regions in the region table. The same shared memory region can be attached multiple times, by the same or different processes. Each attachment of the region creates a new entry in the region table.

After a process attaches a shared memory region, the region becomes part of the process’s virtual address space. Processes access shared memory regions by using the same machine instructions used to access any virtual address.
**Fields**

- **shm_perm**: The *ipc_perm* structure that defines permissions for shared memory operations. See NOTES.
- **shm_segsz**: The size of the shared memory region, in bytes.
- **shm_cpid**: The process ID of the process that created the shared memory region ID.
- **shm_lpid**: The process ID of the last process that performed a *shmat()* or *shmdt()* operation on the shared memory region.
- **shm_nattach**: The number of processes that currently have this region attached.
- **shm_atime**: The time of the last *shmat()* operation.
- **shm_dtime**: The time of the last *shmdt()* operation.
- **shm_ctime**: The time of the last *shmctl()* operation that changed a member of the *shm_id_ds* structure.

**Notes**

The *shm_perm* field identifies the associated *ipc_perm* structure that defines the permissions for operations on the shared memory region. The *ipc_perm* structure (from the *sys/ipc.h* header file) is shown here.

```
struct ipc_perm {
    ushort uid; /* owner’s user id */
    ushort gid; /* owner’s group id */
    ushort cuid; /* creator’s user id */
    ushort cgid; /* creator’s group id */
    ushort mode; /* access modes */
    ushort seq; /* slot usage sequence number */
    key_t key; /* key */
};
```

The *mode* field is a nine-bit field that contains the permissions for shared memory operations. The first three bits identify owner permissions; the second three bits identify group permissions; and the last three bits identify other permissions. In each group, the first bit indicates read permission; the second bit indicates write permission; and the third bit is not used.

**Related Information**

Functions: *shmat(2), shmdt(2), shmct1(2), shmget(2)*
signal.h

Purpose
Contains definitions and variables used by signal functions

Description
The /usr/include/signal.h file defines the signals described in the following table.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGHUP</td>
<td>1</td>
<td>Hangup.</td>
</tr>
<tr>
<td>SIGINT</td>
<td>2</td>
<td>Interrupt.</td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>3</td>
<td>Quit.¹</td>
</tr>
<tr>
<td>SIGILL</td>
<td>4</td>
<td>Invalid instruction (not reset when caught).¹</td>
</tr>
<tr>
<td>SIGTRAP</td>
<td>5</td>
<td>Trace trap (not reset when caught).¹</td>
</tr>
<tr>
<td>SIGABRT</td>
<td>6</td>
<td>End process (see the abort() function).¹</td>
</tr>
<tr>
<td>SIGEMT</td>
<td>7</td>
<td>EMT instruction.</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>8</td>
<td>Arithmetic exception, integer divide by 0 (zero), or floating-point exception.¹</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>9</td>
<td>Kill (cannot be caught or ignored).</td>
</tr>
<tr>
<td>SIGBUS</td>
<td>10</td>
<td>Specification exception.</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>11</td>
<td>Segmentation violation.</td>
</tr>
<tr>
<td>SIGSYS</td>
<td>12</td>
<td>Invalid parameter to system call.¹</td>
</tr>
<tr>
<td>SIGPIPE</td>
<td>13</td>
<td>Write on a pipe when there is no process to read it.</td>
</tr>
<tr>
<td>SIGALRM</td>
<td>14</td>
<td>Alarm clock.</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>15</td>
<td>Software termination signal.</td>
</tr>
<tr>
<td>SIGURG</td>
<td>16</td>
<td>Urgent condition on I/O channel.²</td>
</tr>
<tr>
<td>SIGSTOP</td>
<td>17</td>
<td>Stop (cannot be caught or ignored).³</td>
</tr>
<tr>
<td>SIGTSTP</td>
<td>18</td>
<td>Interactive stop.³</td>
</tr>
<tr>
<td>SIGCONT</td>
<td>19</td>
<td>Continue if stopped (cannot be caught or ignored).⁴</td>
</tr>
<tr>
<td>SIGCHLD</td>
<td>20</td>
<td>To parent on child stop or exit.²</td>
</tr>
<tr>
<td>SIGTTIN</td>
<td>21</td>
<td>Background read attempted from control terminal.³</td>
</tr>
<tr>
<td>SIGTTOU</td>
<td>22</td>
<td>Background write attempted from control terminal.³</td>
</tr>
</tbody>
</table>
## Signal Number Meaning

<table>
<thead>
<tr>
<th>Signal</th>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGIO</td>
<td>23</td>
<td>Input/Output possible or completed.</td>
</tr>
<tr>
<td>SIGXCPU</td>
<td>24</td>
<td>CPU time limit exceeded (see the <code>setrlimit()</code> function).</td>
</tr>
<tr>
<td>SIGXFSZ</td>
<td>25</td>
<td>File size limit exceeded (see the <code>setrlimit()</code> function).</td>
</tr>
<tr>
<td>SIGVTALRM</td>
<td>26</td>
<td>Virtual time alarm (see the <code>setitimer()</code> function).</td>
</tr>
<tr>
<td>SIGPROF</td>
<td>27</td>
<td>Profiling time alarm (see the <code>setitimer()</code> function).</td>
</tr>
<tr>
<td>SIGWINCH</td>
<td>28</td>
<td>Window size change.</td>
</tr>
<tr>
<td>SIGINFO</td>
<td>29</td>
<td>Information request.</td>
</tr>
<tr>
<td>SIGUSR1</td>
<td>30</td>
<td>User-defined signal 1.</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>31</td>
<td>User-defined signal 2.</td>
</tr>
</tbody>
</table>

### Notes to table:

1. Default action includes creating a core dump file.
2. Default action is to ignore these signals.
3. Default action is to stop the process receiving these signals.
4. Default action is to restart or continue the process receiving these signals.

The three types of actions that can be associated with a signal: SIG_DFL, SIG_IGN, or a pointer to a function are described as follows:

**SIG_DFL**

Default action: signal-specific default action. Except for those signal numbers marked with a 2, 3, or 4, the default action for a signal is to end the receiving process with all of the consequences described in the `exit()` system call. In addition, a memory image file is created in the current directory of the receiving process if the signal parameter is one for which a superscript 1 appears in the preceding list and the following conditions are met:

- The effective user ID and the real user ID of the receiving process are equal.
- An ordinary file named `core` exists in the current directory and is writable, or it can be created. If the file must be created, it will have the following properties:
  - The access permission code 0666 (0x1B6), modified by the file creation mask (see the `umask()` function).
  - A file owner ID that is the same as the effective user ID of the receiving process.
  - A file group ID that is the same as the effective group ID of the receiving process.
For signal numbers marked with a superscript 4, the default action is to restart the receiving process if it is stopped, or to continue execution of the receiving process.

For signal numbers marked with a superscript 3, the default action is to stop the execution of the receiving process temporarily. When a process stops, a SIGCHLD signal is sent to its parent process, unless the parent process has set the SA_NOCLDSTOP bit. While a process is stopped, any additional signals that are sent to the process are not delivered until the process is continued. An exception to this is SIGKILL, which always terminates the receiving process. Another exception is SIGCONT, which always causes the receiving process to restart or continue running. A process whose parent has ended shall be sent a SIGKILL signal if the SITSTP, SIGTTIN, or SIGTTOU signals are generated for that process.

For signal numbers marked with a superscript 2, the default action is to ignore the signal. In this case, delivery of the signal has no effect on the receiving process.

If a signal action is set to SIG_DFL while the signal is pending, the signal remains pending.

SIG_IGN    Ignore signal.

Delivery of the signal has no effect on the receiving process. If a signal action is set to SIG_IGN while the signal is pending, the pending signal is discarded.

Note that the SIGKILL, SIGSTOP, and SIGCONT signals cannot be ignored.

pointer to a function

Catch signal.

Upon delivery of the signal, the receiving process is to run the signal-catching function specified by the pointer to function. The signal-handler subroutine can be declared as follows:

    void handler(signal)
    int signal;

The signal parameter is the signal number.

A new signal mask is calculated and installed for the duration of the signal-catching function (or until sigprocmask() or sigsuspend() system calls are made). This mask is formed by taking the union of the process signal mask, the mask associated with the action for the signal being delivered, and a mask corresponding to the signal being delivered. The mask associated with the signal-catching
function is not allowed to block those signals that cannot be ignored. This is enforced by the kernel without causing an error to be indicated. If and when the signal-catching function returns, the original signal mask is restored (modified by any `sigprocmask()` calls that were made since the signal-catching function was called) and the receiving process resumes execution at the point it was interrupted.

The signal-catching function can cause the process to resume in a different context by calling the `longjmp()` subroutine. When the `longjmp()` subroutine is called, the process leaves the signal stack, if it is currently on it, and restores the process signal mask to the state when the corresponding `setjmp()` call was made.

Once an action is installed for a specific signal, it remains installed until another action is explicitly requested (by another call to the `sigaction()` system call), or until one of the `exec` system calls is called.

If a signal action is set to a pointer to a function while the signal is pending, the signal remains pending.

When signal-catching functions are invoked asynchronously with process execution, the behavior of some of the functions defined by this standard is unspecified if they are called from a signal-catching function. The following set of functions are reentrant with respect to signals (that is, applications can invoke them, without restriction, from signal-catching functions):

- `_exit()`, `access()`, `alarm()`, `chdir()`
- `chmod()`, `chown()`, `close()`, `creat()`, `fcntl()`, `getegid()`, `geteuid()`, `getgid()`, `getgroups()`, `getpgrp()`, `getpid()`, `kill()`, `link()`, `mkfifo()`, `mksfifo()`, `open()`, `read()`, `rename()`, `setgid()`, `setpgid()`, `setuid()`, `sigaddset()`, `sigdelset()`, `sigfillset()`, `sigismember()`, `signal()`, `sigpending()`, `sigprocmask()`, `sleep()`, `statx()`, `tcflush()`, `tcgetattr()`, `tcgetattr()`, `tcsetattr()`, `tcsetpgrp()`, `tcsetattr()`, `tcsetptr()`, `tcsetpgrp()`, `time()`, `times()`, `umask()`, `uname()`, `wait2()`

All other system calls should not be called from signal-catching functions since their behavior is undefined.
Related Information

Functions: sigaction(2), sigblock(2), sigemptyset(3), siginterrupt(3),
           siglongjmp(3), sigpause(3), sigpending(2), sigprocmask(2), sigreturn(2),
           sigset(3), sigsetjmp(3), sigstack(2), sigsuspend(2), sigvec(2), sigwait(3)
spp

Purpose
Xerox sequenced packet protocol (SPP)

Synopsis
#include <sys/socket.h>
#include <netns/ns.h>
s = socket(AF_NS, SOCK_STREAM, 0);
#include <netns/sp.h>
s = socket(AF_NS, SOCK_SEQPACKET, 0);

Description
The SPP provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the SOCK_STREAM abstraction. SPP uses the standard NS address formats.

Sockets utilizing the SPP are either active or passive. Active sockets initiate connections to passive sockets. By default, SPP sockets are created active; to create a passive socket the listen() function must be used after binding the socket with the bind() function. Only passive sockets may use the accept() function to accept incoming connections. Only active sockets may use the connect() function to initiate connections.

Passive sockets may underspecify their location to match incoming connection requests from multiple networks. This technique, termed wildcard addressing, allows a single server to provide service to clients on multiple networks. To create a socket which listens on all networks, the NS address of all zeroes must be bound. The SPP port may still be specified at this time; if the port is not specified the system will assign one. Once a connection has been established the socket’s address is fixed by the peer entity’s location. The address assigned the socket is the address associated with the network interface through which packets are being transmitted and received. Normally this address corresponds to the peer entity’s network.

If the SOCK_SEQPACKET socket type is specified, each packet received includes the actual 12-byte sequenced packet header for the user to inspect. This facilitates the implementation of higher level Xerox protocols which make use of the data stream type field and the end of message bit. Conversely, the user is required to supply a 12-byte header, the only parts of which are inspected are the data stream type and the end of message fields.
For either socket type, packets received with the *attention* bit set are interpreted as out of band data. Data sent with send(..., ..., MSG_OOB) cause the attention bit to be set.

The following socket options are available:

**SO_DEFAULT_HEADERS**
Determines the data stream type and whether the end of message bit is to be set on every ensuing packet.

**SO_MTU**
Specifies the maximum amount of user data in a single packet. The default is 576 bytes - `sizeof(struct spidp)`. This quantity affects windowing; increasing it without increasing the amount of buffering in the socket will lower the number of unread packets accepted. Anything larger than the default will not be forwarded by a bona fide Xerox product internetwork router. The data argument for the `setsockopt()` function must be an *unsigned short*.

**Errors**

If a socket option fails, `errno` may be set to one of the following values:

- [EISCONN] The socket to be connected already has a connection.
- [ENOBUFS] The system ran out of memory for an internal data structure.
- [ETIMEDOUT]
  The connection was dropped due to excessive retransmissions.
- [ECONNRESET]
  The remote peer forced the connection to be closed.
- [ECONNREFUSED]
  The remote peer actively refused establishment of a connection (usually because no process is listening to the port).
- [EADDRINUSE]
  An attempt was made to create a socket with a port which has already been allocated.
- [EADDRNOTAVAIL]
  An attempt was made to create a socket with a network address for which no network interface exists.

**Related Information**

Files: `netintro(7), ns(7)`
stab(4)

Purpose
Symbol table types

Synopsis
#include <stab.h>

Description
The stab.h header file defines some values of the n_type field of the symbol table of a.out files. These are the types for permanent symbols (that is, not local labels, etc.) used by the old debugger sdb. Symbol table entries can be produced by the .stabs assembler directive. This allows one to specify a double-quote delimited name, a symbol type, one char and one short of information about the symbol, and an unsingned long (usually an address). To avoid having to produce an explicit label for the address field, the .stabd directive can be used to implicitly address the current location. If no name is needed, symbol table entries can be generated using the .stabn directive. The loader promises to preserve the order of symbol table entries produced by .stab directives. An element of the symbol table consists of the following structure:

/* SYMBOL INFORMATION ENTRY
* This is used for defined symbols, imports and stabs. The type (kind)
* of the associated symbols load command determines which.
*/
typedef struct symbol_info_t {  
    union {mo_offset_t symbol_name;  
            mo_ptr_t symbol_nameP;  
        } si_name;  
    mo_short_t si_package_index;  
    mo_short_t si_type;  
    mo_short_t si_flags;  
    mo_byte_t si_reserved byte;  
    mo_byte_t si_sc_type;  
    union {mo_addr_t def_val;/* defined section, offset */  
            mo_long_t imp_val;/* index in import list */  
            mo_long_t lit_val;/* literal value */  
            mo_vm_addr_tabs_val;/* absolute value */  
        } si_value;  
} symbol_info_t;
The low bits of the `si_sc_type` field are used to place a symbol into at most one segment, according to the following masks. A symbol can be in none of these segments by having none of these segment bits set.

/*
 * Simple values for `si_sc_type`.
*/
#define N_UNDF 0x0 /* undefined */
#define N_ABS 0x2 /* absolute */
#define N_TEXT 0x4 /* text */
#define N_DATA 0x6 /* data */
#define N_BSS 0x8 /* bss */
#define N_EXT 01 /* external bit, or'ed in */

The `n_value` field of a symbol is relocated by the linker, `ld`, as an address within the appropriate segment. The `n_value` fields of symbols not in any segment are unchanged by the linker. In addition, the linker will discard certain symbols, according to rules of its own, unless the `si_sc_type` field has one of the following bits set:

#define N_STAB 0xe0 /* if any of these bits set, don’t discard */

This allows up to 112 (7 * 16) symbol types, split between the various segments. Some of these have already been claimed. The old symbolic debugger, `sdb`, uses the following `n_type` values:

#define N_GSYM 0x20 /* global symbol: name,,0,type,0 */
#define N_FNAME 0x22 /* procedure name (f77 kludge): name,,0 */
#define N_FUN 0x24 /* procedure: name,,0,linenumber,address */
#define N_STSYM 0x26 /* static symbol: name,,0,type,address */
#define N_LCSYM 0x28 /* .lcomm symbol: name,,0,type,address */
#define N_RSYM 0x40 /* register sym: name,,0,type,register */
#define N_SLINE 0x44 /* src line: 0,,0,linenumber,address */
#define N_SSYM 0x60 /* structure elt: name,,0,type,struct_offset */
#define N_SO 0x64 /* source file name: name,,0,0,address */
#define N_LSYM 0x80 /* local sym: name,,0,type,offset */
#define N_SOL 0x84 /* #included file name: name,,0,0,address */
#define N_PSYM 0xa0 /* parameter: name,,0,type,offset */
#define N_ENTRY 0xa4 /* alternate entry: name,,linenumber,address */
#define N_LBRAC 0xc0 /* left bracket: 0,,0,nesting level,address */
#define N_RBRAC 0xe0 /* right bracket: 0,0,nesting level,address */
#define N_BCOMM 0xe2 /* begin common: name,, */
#define N_ECOMM 0xe4 /* end common: name,, */
#define N_ECOML 0xe8 /* end common (local name): ,,address */
#define N_LENG 0xfe /* second stab entry with length information */

The comments give sdb conventional use for .stabs and the n_name, n_other, n_desc, and n_value fields of the given n_type. The sdb debugger uses the n_desc field to hold a type specifier in the form used by the Portable C Compiler, cc.

The Berkeley Pascal compiler, pc, uses the following si_sc_type value:
#define N_PC 0x30 /* global pascal symbol: name,,0,subtype,line */

and uses the following subtypes to do type checking across separately compiled files:
1 Source filename
2 Included filename
3 Global label
4 Global constant
5 Global type
6 Global variable
7 Global function
8 Global procedure
9 External function
10 External procedure
11 Library variable
12 Library routine

Related Information

Commands: as(1), ld(1)
tar

Purpose  Tape archive file format

Description

The `tar` command dumps several files into one, in a medium suitable for transportation.

A tar tape or tar file is a series of blocks, with each block of size TBLOCK. A file on the tape is represented by a header block which describes the file, followed by zero or more blocks which give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an end-of-file indicator.

The blocks are grouped for physical I/O operations. Each group of \( n \) blocks (where \( n \) is set by the `b` keyletter on the `tar` command line, with a default of 20 blocks) is written with a single system call. On nine-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads.

The header block looks like:

```c
#define TBLOCK 512
#define NAMSIZ 100

union hblock {
    char dummy[TBLOCK];
    struct header {
        char name[NAMSIZ];
        char mode[8];
        char uid[8];
        char gid[8];
        char size[12];
        char mtime[12];
        char chksum[8];
        char linkflag;
        char linkname[NAMSIZ];
    } dbuf;
};
```
The name field is a null-terminated string. The other fields are zero-filled octal numbers in ASCII format. If the width of each field is given as $w$, each field contains $w-2$ digits, a space, and a null, with the exception of the size and mtime fields, which do not contain the trailing null, and the chksum field, which has a null followed by a space.

The name field is the name of the file, as specified on the tar command line. Files dumped because they were in a directory that was named in the command line have the directory name as prefix and /filename as suffix.

The mode field is the file mode, with the top bit masked off. The uid and gid fields are the user and group numbers that own the file. The size field is the size of the file in bytes. Links and symbolic links are dumped with this field specified as zero.

The mtime field is the modification time of the file at the time it was dumped.

The chksum field is an octal ASCII value which represents the sum of all the bytes in the header block. When calculating the checksum, the chksum field is treated as if it were all blanks.

The linkflag field is null if the file is a regular or special file, ASCII 1 if it is an hard link, and ASCII 2 if it is a symbolic link. The name that the file is linked to, if any, is in the linkname field, with a trailing null. Unused fields of the header are binary zeros (and are included in the checksum).

The first time a given i-node number is dumped, it is dumped as a regular file. Subsequently, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved but the file it was linked to is not, an error message is printed and the tape must be manually rescanned to retrieve the file that it is linked to.

The encoding of the header is designed to be portable across machines.

Related Information

Commands: tar(1)
tcp

**Purpose**  
Internet transmission control protocol

**Synopsis**  
```
#include <sys/socket.h>
#include <netinet/in.h>
s = socket(AF_INET, SOCK_STREAM, 0);
```

**Description**  
The TCP provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the SOCK_STREAM abstraction. TCP uses the standard Internet address format and, in addition, provides a per-host collection of port addresses. Thus, each address is composed of an Internet address specifying the host and network, with a specific TCP port on the host identifying the peer entity.

Sockets utilizing the TCP are either active or passive. Active sockets initiate connections to passive sockets. By default, TCP sockets are created active; to create a passive socket the `listen()` function must be used after binding the socket with the `bind()` function. Only passive sockets may use the `accept()` function to accept incoming connections. Only active sockets may use the `connect()` function to initiate connections.

Passive sockets may underspecify their location to match incoming connection requests from multiple networks. This technique, termed *wildcard addressing*, allows a single server to provide service to clients on multiple networks. To create a socket which listens on all networks, the Internet address INADDR_ANY must be bound. The TCP port may still be specified at this time; if the port is not specified the system will assign one. Once a connection has been established the socket’s address is fixed by the peer entity’s location. The address assigned the socket is the address associated with the network interface through which packets are being transmitted and received. Normally this address corresponds to the peer entity’s network.

TCP supports one socket option which is set with the `setsockopt()` function and tested with the `getsockopt()` function. Under most circumstances, TCP sends data when it is presented; when outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgement is received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this gathering
of output may cause significant delays. Therefore, TCP provides a Boolean option, TCP_NODELAY (from the netinet/tcp.h header file), to defeat this algorithm. The option level for the setsockopt() function is the protocol number for TCP, available from the getprotobyname() function.

Options at the IP transport level may be used with TCP; see ip(4). Incoming connection requests that are source-routed are noted, and the reverse source route is used in responding.

**Errors**

If a socket operation fails, errno may be set to one of the following values:

- [EISCONN] The socket to be connected already has a connection.
- [ENOBUFFS] The system ran out of memory for an internal data structure.
- [ETIMEDOUT] A connection was dropped due to excessive retransmissions.
- [ECONNRESET] The remote peer forced the connection to be closed.
- [ECONNREFUSED] The remote peer actively refuses connection establishment (usually because no process is listening to the port).
- [EADDRINUSE] An attempt is made to create a socket with a port which has already been allocated.
- [EADDRNOTAVAIL] An attempt is made to create a socket with a network address for which no network interface exists.

**Related Information**

Functions: getsockopt(2), socket(2)
Files: netintro(7), inet(7), ip(7)
terminfo

Purpose  Describes terminals by capability

Description

A **terminfo** file is a database that describes the capabilities and method of operation of various terminals. The database includes definitions of initialization sequences, padding requirements, cursor positioning, and other command sequences that control specific terminals.

Before a **terminfo** source file can be used, it must be compiled using the **tic** command. The compiled **terminfo** entries are placed into subdirectories of the `/usr/lib/terminfo` directory. This directory may be redefined with the **TERMINFO** environment variable. See the **EXAMPLE** section for more information on using the **TERMINFO** environment variable.

Each **terminfo** file entry consists of a number of fields separated by commas. Any white space between commas is ignored. The first field for each terminal supplies the names the terminal is known by, separated by vertical bars (|). The first name given is the most common abbreviation for the terminal, the last name given is a long name fully identifying the terminal, and all others are synonyms for the terminal name. All names except the last are in lowercase and do not contain any white space.

The fields following the terminal name supply the capabilities of the terminal. Although capability names have no absolute length limit, an informal limit of 5 characters is adopted to keep them short and to allow the tabs in the source file **caps** to be aligned. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64 standard of 1979.

Terminal names (except the last) are chosen using the following conventions. A root name is chosen to represent the particular hardware class of the terminal. This
name does not contain hyphens, except to avoid synonyms that conflict with other names. Possible modes for the hardware or user preferences are indicated by appending a - (hyphen) and one of the suffixes listed below:

- **am** With automatic margins (usually default)
- **c** Color mode
- **w** Wide mode (more than 80 columns)
- **nam** Without automatic margins
- **n** Number of lines on the screen
- **na** No arrow keys (leave them in local)
- **np** Number of pages of memory
- **rv** Reverse video

Thus, a vt100 terminal in 132-column mode would be vt100-w.

Capabilities in the terminfo file are of three types:

- Boolean capabilities indicate that the terminal has some particular feature. Boolean capabilities are evaluated as true if the corresponding name is in the terminal description.
- Numeric capabilities give the size of the terminal or the size of particular delays.
- String capabilities give a sequence that can be used to perform particular terminal operations.

To continue an entry onto multiple lines, place white space at the beginning of each subsequent line. Include a comment on a line beginning with the # (number sign) character. To comment out an individual capability, precede it with a . (dot).

**List of Capabilities**

The following table shows the C variable (which the programmer uses to access the terminfo capabilities), the capability name (the short name used in the text of the database), the 2-letter internal code used in the compiled database (always corresponding to a termcap capability name), and a short description of each capability.
<table>
<thead>
<tr>
<th>Boolean</th>
<th>Name</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_left_margin</td>
<td>bw</td>
<td>bw</td>
<td>Indicates <strong>cub1</strong> wraps from column 0 (zero) to last column.</td>
</tr>
<tr>
<td>auto_right_margin</td>
<td>am</td>
<td>am</td>
<td>Indicates terminal has automatic margins.</td>
</tr>
<tr>
<td>beehive_glitch</td>
<td>xsb</td>
<td>xs</td>
<td>Indicates a terminal with F1=(&lt;\text{esc})&gt; and F2=(&lt;\text{Ctrl-C})&gt;</td>
</tr>
<tr>
<td>ceol_standout_glitch</td>
<td>xhp</td>
<td>xs</td>
<td>Indicates standout not erased by overwriting.</td>
</tr>
<tr>
<td>eat_newline_glitch</td>
<td>xenl</td>
<td>xn</td>
<td>Ignores newline character after 80 columns.</td>
</tr>
<tr>
<td>erase_overstrike</td>
<td>eo</td>
<td>eo</td>
<td>Erases overstrikes with a blank.</td>
</tr>
<tr>
<td>generic_type</td>
<td>gn</td>
<td>gn</td>
<td>Indicates generic line type (such as dialup, switch)</td>
</tr>
<tr>
<td>hard_copy</td>
<td>hc</td>
<td>hc</td>
<td>Indicates hardcopy terminal.</td>
</tr>
<tr>
<td>has_meta_key</td>
<td>km</td>
<td>km</td>
<td>Indicates terminal has a meta key (shift, sets parity bit).</td>
</tr>
<tr>
<td>has_status_line</td>
<td>hs</td>
<td>hs</td>
<td>Indicates terminal has extra status line.</td>
</tr>
<tr>
<td>insert_null_glitch</td>
<td>in</td>
<td>in</td>
<td>Indicates insert mode distinguishes nulls.</td>
</tr>
<tr>
<td>memory_above</td>
<td>da</td>
<td>da</td>
<td>Retains information above display in memory.</td>
</tr>
<tr>
<td>memory_below</td>
<td>db</td>
<td>db</td>
<td>Retains information below display in memory.</td>
</tr>
<tr>
<td>move_insert_mode</td>
<td>mir</td>
<td>mi</td>
<td>Indicates safe to move while in insert mode.</td>
</tr>
<tr>
<td>move_standout_mode</td>
<td>msgr</td>
<td>ms</td>
<td>Indicates safe to move in standout modes.</td>
</tr>
<tr>
<td>over_strike</td>
<td>os</td>
<td>os</td>
<td>Indicates terminal overstrikes.</td>
</tr>
<tr>
<td>status_line_esc_ok</td>
<td>eslok</td>
<td>es</td>
<td>Indicates escape can be used on the status line.</td>
</tr>
<tr>
<td>teleray_glitch</td>
<td>xt</td>
<td>xt</td>
<td>Indicates destructive tabs and blanks inserted while entering standout mode</td>
</tr>
<tr>
<td>tilde_glitch</td>
<td>hz</td>
<td>hz</td>
<td>Indicates terminal cannot print <code>~</code> (tilde) characters.</td>
</tr>
<tr>
<td>transparent_underline</td>
<td>ul</td>
<td>ul</td>
<td>Overstrikes with underline character.</td>
</tr>
<tr>
<td>xon_xoff</td>
<td>xon</td>
<td>xo</td>
<td>Indicates terminal uses xon/xoff handshaking.</td>
</tr>
<tr>
<td>Number</td>
<td>Name</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------</td>
<td>------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>columns</td>
<td>cols</td>
<td>co</td>
<td>Specifies the number of columns in a line. Provides tabs initially every # spaces.</td>
</tr>
<tr>
<td>init_tabs</td>
<td>it</td>
<td>it</td>
<td>Specifies the number of lines on screen or page.</td>
</tr>
<tr>
<td>lines</td>
<td>lines</td>
<td>li</td>
<td>Specifies the number of lines of memory if greater than the number of lines on the screen. A value of 0 (zero) indicates that the number of lines is variable.</td>
</tr>
<tr>
<td>lines_of_memory</td>
<td>lines_of_memory</td>
<td>lm</td>
<td></td>
</tr>
<tr>
<td>magic_cookie_glitch</td>
<td>magic_cookie_glitch</td>
<td>xmc</td>
<td>Indicates number of blank characters left by \texttt{sms} or \texttt{rms}.</td>
</tr>
<tr>
<td>padding_baud_rate</td>
<td>padding_baud_rate</td>
<td>pb</td>
<td>Indicates lowest baud where carriage return and line return padding is needed.</td>
</tr>
<tr>
<td>virtual_terminal</td>
<td>virtual_terminal</td>
<td>vt</td>
<td>Indicates virtual terminal number.</td>
</tr>
<tr>
<td>width_status_lines</td>
<td>width_status_lines</td>
<td>wsl</td>
<td>Specifies the number of columns in status line.</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------</td>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>appl_defined_str</td>
<td>apstr</td>
<td>za</td>
<td>Application-defined terminal string.</td>
</tr>
<tr>
<td>back_tab</td>
<td>cbt</td>
<td>bt</td>
<td>Back tab. (P)</td>
</tr>
<tr>
<td>bell</td>
<td>bel</td>
<td>bi</td>
<td>Produces an audible signal (bell). (P)</td>
</tr>
<tr>
<td>box_chars_1</td>
<td>box1</td>
<td>bx</td>
<td>Box characters primary set.</td>
</tr>
<tr>
<td>box_chars_2</td>
<td>box2</td>
<td>by</td>
<td>Box characters alternate set.</td>
</tr>
<tr>
<td>box_attr_1</td>
<td>batt1</td>
<td>Bx</td>
<td>Attributes for box_chars_1.</td>
</tr>
<tr>
<td>box_attr_2</td>
<td>batt2</td>
<td>By</td>
<td>Attributes for box_chars_2.</td>
</tr>
<tr>
<td>carriage_return</td>
<td>cr</td>
<td>cr</td>
<td>Indicates carriage return. (P*)</td>
</tr>
<tr>
<td>change_scroll_region</td>
<td>csr</td>
<td>cs</td>
<td>Changes scroll region to lines 1 through 2. (PG)</td>
</tr>
<tr>
<td>clear_all_tabs</td>
<td>tbc</td>
<td>ct</td>
<td>Clears all tab stops. (P)</td>
</tr>
<tr>
<td>clear_screen</td>
<td>clear</td>
<td>cl</td>
<td>Clears screen and puts cursor in home position. (P*)</td>
</tr>
<tr>
<td>clr_eol</td>
<td>el</td>
<td>ce</td>
<td>Clears to end of line. (P)</td>
</tr>
<tr>
<td>clr_eos</td>
<td>ed</td>
<td>cd</td>
<td>Clears to end of the display. (P*)</td>
</tr>
<tr>
<td>color_bg_0</td>
<td>colb0</td>
<td>d0</td>
<td>Background color 0 black.</td>
</tr>
<tr>
<td>color_bg_1</td>
<td>colb1</td>
<td>d1</td>
<td>Background color 1 red.</td>
</tr>
<tr>
<td>color_bg_2</td>
<td>colb2</td>
<td>d2</td>
<td>Background color 2 green.</td>
</tr>
<tr>
<td>color_bg_3</td>
<td>colb3</td>
<td>d3</td>
<td>Background color 3 brown.</td>
</tr>
<tr>
<td>color_bg_4</td>
<td>colb4</td>
<td>d4</td>
<td>Background color 4 blue.</td>
</tr>
<tr>
<td>color_bg_5</td>
<td>colb5</td>
<td>d5</td>
<td>Background color 5 magenta.</td>
</tr>
<tr>
<td>color_bg_6</td>
<td>colb6</td>
<td>d6</td>
<td>Background color 6 cyan.</td>
</tr>
<tr>
<td>color_bg_7</td>
<td>colb7</td>
<td>d7</td>
<td>Background color 7 white.</td>
</tr>
<tr>
<td>color_fg_0</td>
<td>colf0</td>
<td>c0</td>
<td>Foreground color 0 white.</td>
</tr>
<tr>
<td>color_fg_1</td>
<td>colf1</td>
<td>c1</td>
<td>Foreground color 1 red.</td>
</tr>
<tr>
<td>color_fg_2</td>
<td>colf2</td>
<td>c2</td>
<td>Foreground color 2 green.</td>
</tr>
<tr>
<td>color_fg_3</td>
<td>colf3</td>
<td>c3</td>
<td>Foreground color 3 brown.</td>
</tr>
<tr>
<td>color_fg_4</td>
<td>colf4</td>
<td>c4</td>
<td>Foreground color 4 blue.</td>
</tr>
<tr>
<td>color_fg_5</td>
<td>colf5</td>
<td>c5</td>
<td>Foreground color 5 magenta.</td>
</tr>
<tr>
<td>color_fg_6</td>
<td>colf6</td>
<td>c6</td>
<td>Foreground color 6 cyan.</td>
</tr>
<tr>
<td>color_fg_7</td>
<td>colf7</td>
<td>c7</td>
<td>Foreground color 7 black.</td>
</tr>
<tr>
<td>column_address</td>
<td>hpa</td>
<td>ch</td>
<td>Sets cursor column. (PG)</td>
</tr>
<tr>
<td>command_character</td>
<td>cmdch</td>
<td>CC</td>
<td>Indicates terminal command prototype character can be set.</td>
</tr>
<tr>
<td>cursor_address</td>
<td>cup</td>
<td>cm</td>
<td>Indicates screen relative cursor motion row #1 col #2. (PG)</td>
</tr>
<tr>
<td>cursor_down</td>
<td>cud1</td>
<td>do</td>
<td>Moves cursor down one line.</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>cursor_home</td>
<td>home</td>
<td>ho</td>
<td>Moves cursor to home position (if no <code>cup</code>)</td>
</tr>
<tr>
<td>cursor_invisible</td>
<td>civis</td>
<td>vi</td>
<td>Makes cursor invisible</td>
</tr>
<tr>
<td>cursor_left</td>
<td>cub1</td>
<td>le</td>
<td>Moves cursor left one space</td>
</tr>
<tr>
<td>cursor_mem_address</td>
<td>mrcup</td>
<td>CM</td>
<td>Indicates memory relative cursor addressing.</td>
</tr>
<tr>
<td>cursor_normal</td>
<td>cnorm</td>
<td>ve</td>
<td>Makes cursor appear normal (undo vs or vi).</td>
</tr>
<tr>
<td>cursor_right</td>
<td>cuf1</td>
<td>nd</td>
<td>Indicates nondestructive space (cursor right).</td>
</tr>
<tr>
<td>cursor_to_ll</td>
<td>ll</td>
<td>ll</td>
<td>Moves cursor to first column of last line (if no <code>cup</code>).</td>
</tr>
<tr>
<td>cursor_up</td>
<td>cuu1</td>
<td>up</td>
<td>Moves cursor up one line (cursor up).</td>
</tr>
<tr>
<td>cursor_visible</td>
<td>cvvis</td>
<td>vs</td>
<td>Makes cursor very visible</td>
</tr>
<tr>
<td>delete_character</td>
<td>dch1</td>
<td>dc</td>
<td>Deletes character. (P*)</td>
</tr>
<tr>
<td>delete_line</td>
<td>dl1</td>
<td>dl</td>
<td>Deletes line. (P*)</td>
</tr>
<tr>
<td>dis_status_line</td>
<td>dsl</td>
<td>ds</td>
<td>Disables status line</td>
</tr>
<tr>
<td>down_half_line</td>
<td>hd</td>
<td>hd</td>
<td>Indicates subscript (forward 1/2 linefeed).</td>
</tr>
<tr>
<td>enter_alt_charset_mode</td>
<td>smacs</td>
<td>as</td>
<td>Starts alternate character set. (P)</td>
</tr>
<tr>
<td>enter_blink_mode</td>
<td>blink</td>
<td>mb</td>
<td>Enables blinking.</td>
</tr>
<tr>
<td>enter_bold_mode</td>
<td>bold</td>
<td>md</td>
<td>Enables bold (extra bright) mode.</td>
</tr>
<tr>
<td>enter_ca_mode</td>
<td>smcup</td>
<td>ti</td>
<td>Begins programs that use <code>cup</code>.</td>
</tr>
<tr>
<td>enter_delete_mode</td>
<td>smdc</td>
<td>dm</td>
<td>Starts delete mode.</td>
</tr>
<tr>
<td>enter_dim_mode</td>
<td>dim</td>
<td>mh</td>
<td>Enables half-bright mode</td>
</tr>
<tr>
<td>enter_insert_mode</td>
<td>smir</td>
<td>m</td>
<td>Starts insert mode.</td>
</tr>
<tr>
<td>enter_protected_mode</td>
<td>prot</td>
<td>mp</td>
<td>Enables protected mode</td>
</tr>
<tr>
<td>enter_reverse_mode</td>
<td>rev</td>
<td>mr</td>
<td>Enables reverse video mode</td>
</tr>
<tr>
<td>enter_secure_mode</td>
<td>invis</td>
<td>mk</td>
<td>Enables blank mode (characters invisible).</td>
</tr>
<tr>
<td>enter_standout_mode</td>
<td>smso</td>
<td>so</td>
<td>Begins standout mode</td>
</tr>
<tr>
<td>enter_underline_mode</td>
<td>smul</td>
<td>us</td>
<td>Starts underscore mode</td>
</tr>
<tr>
<td>erase_chars</td>
<td>ech</td>
<td>ec</td>
<td>Erases #1 characters. (PG)</td>
</tr>
<tr>
<td>exit_alt_charset_mode</td>
<td>rmacs</td>
<td>ae</td>
<td>Ends alternate character set. (P)</td>
</tr>
<tr>
<td>exit_attribute_mode</td>
<td>sgr0</td>
<td>me</td>
<td>Disables all attributes</td>
</tr>
<tr>
<td>exit_ca_mode</td>
<td>mcup</td>
<td>te</td>
<td>Ends programs that use <code>cup</code>.</td>
</tr>
<tr>
<td>exit_delete_mode</td>
<td>rmdc</td>
<td>ed</td>
<td>Ends delete mode.</td>
</tr>
<tr>
<td>exit_insert_mode</td>
<td>rmir</td>
<td>el</td>
<td>Ends insert mode.</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>exit_standout_mode</td>
<td>rmso</td>
<td>se</td>
<td>Ends stand out mode.</td>
</tr>
<tr>
<td>exit_underline_mode</td>
<td>rmul</td>
<td>ue</td>
<td>Ends underscore mode.</td>
</tr>
<tr>
<td>flash_screen</td>
<td>lash</td>
<td>vb</td>
<td>Indicates visual bell (may not move cursor).</td>
</tr>
<tr>
<td>font_0</td>
<td>font0</td>
<td>f0</td>
<td>Select font 0.</td>
</tr>
<tr>
<td>font_1</td>
<td>font1</td>
<td>f1</td>
<td>Select font 1.</td>
</tr>
<tr>
<td>font_2</td>
<td>font2</td>
<td>f2</td>
<td>Select font 2.</td>
</tr>
<tr>
<td>font_3</td>
<td>font3</td>
<td>f3</td>
<td>Select font 3.</td>
</tr>
<tr>
<td>font_4</td>
<td>font4</td>
<td>f4</td>
<td>Select font 4.</td>
</tr>
<tr>
<td>font_5</td>
<td>font5</td>
<td>f5</td>
<td>Select font 5.</td>
</tr>
<tr>
<td>font_6</td>
<td>font6</td>
<td>f6</td>
<td>Select font 6.</td>
</tr>
<tr>
<td>font_7</td>
<td>font7</td>
<td>f7</td>
<td>Select font 7.</td>
</tr>
<tr>
<td>form_feed</td>
<td>ff</td>
<td>ff</td>
<td>Ejects page (hard-copy terminal). (P*)</td>
</tr>
<tr>
<td>from_status_line</td>
<td>fsl</td>
<td>fs</td>
<td>Returns from status line.</td>
</tr>
<tr>
<td>init_1string</td>
<td>is1</td>
<td>i1</td>
<td>Initializes terminal.</td>
</tr>
<tr>
<td>init_2string</td>
<td>is2</td>
<td>i2</td>
<td>Initializes terminal.</td>
</tr>
<tr>
<td>init_3string</td>
<td>is3</td>
<td>i3</td>
<td>Initializes terminal.</td>
</tr>
<tr>
<td>init_file</td>
<td>if</td>
<td>if</td>
<td>Identifies file containing &quot;is&quot;</td>
</tr>
<tr>
<td>insert_character</td>
<td>ich1</td>
<td>ic</td>
<td>Inserts character. (P)</td>
</tr>
<tr>
<td>insert_line</td>
<td>il1</td>
<td>al</td>
<td>Adds new blank line.</td>
</tr>
<tr>
<td>insert_padding</td>
<td>ip</td>
<td>ip</td>
<td>Inserts pad after character inserted. (P*)</td>
</tr>
<tr>
<td>key_backspace</td>
<td>kbs</td>
<td>kb</td>
<td>Sent by backspace key.</td>
</tr>
<tr>
<td>key_back_tab</td>
<td>kbtab</td>
<td>k0</td>
<td>Sent by backtab key.</td>
</tr>
<tr>
<td>key_catab</td>
<td>ktbc</td>
<td>ka</td>
<td>Sent by clear-all-tabs key.</td>
</tr>
<tr>
<td>key_clear</td>
<td>kclr</td>
<td>kC</td>
<td>Sent by clear-screen or erase key.</td>
</tr>
<tr>
<td>key_ctab</td>
<td>kctab</td>
<td>kt</td>
<td>Sent by clear-tab key.</td>
</tr>
<tr>
<td>key_command</td>
<td>kcmd</td>
<td>kc</td>
<td>Command request key.</td>
</tr>
<tr>
<td>key_command-pane</td>
<td>kcnp</td>
<td>kW</td>
<td>Command pane key.</td>
</tr>
<tr>
<td>key_dc</td>
<td>kdch1</td>
<td>kD</td>
<td>Sent by delete-character key.</td>
</tr>
<tr>
<td>key_dl</td>
<td>kd1</td>
<td>kL</td>
<td>Sent by delete-line key.</td>
</tr>
<tr>
<td>key_do</td>
<td>kdo</td>
<td>ki</td>
<td>Do request key.</td>
</tr>
<tr>
<td>key_down</td>
<td>kcud1</td>
<td>kd</td>
<td>Sent by terminal cursor down key.</td>
</tr>
<tr>
<td>key_eic</td>
<td>krmir</td>
<td>kM</td>
<td>Sent by rmir or smir in insert mode.</td>
</tr>
<tr>
<td>key_end</td>
<td>kend</td>
<td>kw</td>
<td>End key.</td>
</tr>
<tr>
<td>key_eol</td>
<td>ke1</td>
<td>kE</td>
<td>Sent by clear-to-end-of-line key.</td>
</tr>
<tr>
<td>key_eos</td>
<td>ked</td>
<td>kS</td>
<td>Sent by clear-to-end-of-screen key.</td>
</tr>
<tr>
<td>key_f0</td>
<td>kf0</td>
<td>k0</td>
<td>Sent by function key F0.</td>
</tr>
<tr>
<td>key_f1</td>
<td>kf1</td>
<td>k1</td>
<td>Sent by function key F1.</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>key_f2</td>
<td>kf2</td>
<td>k2</td>
<td>Sent by function key F2.</td>
</tr>
<tr>
<td>key_f3</td>
<td>kf3</td>
<td>k3</td>
<td>Sent by function key F3.</td>
</tr>
<tr>
<td>key_f4</td>
<td>kf4</td>
<td>k4</td>
<td>Sent by function key F4.</td>
</tr>
<tr>
<td>key_f5</td>
<td>kf5</td>
<td>k5</td>
<td>Sent by function key F5.</td>
</tr>
<tr>
<td>key_f6</td>
<td>kf6</td>
<td>k6</td>
<td>Sent by function key F6.</td>
</tr>
<tr>
<td>key_f7</td>
<td>kf7</td>
<td>k7</td>
<td>Sent by function key F7.</td>
</tr>
<tr>
<td>key_f8</td>
<td>kf8</td>
<td>k8</td>
<td>Sent by function key F8.</td>
</tr>
<tr>
<td>key_f9</td>
<td>kf9</td>
<td>k9</td>
<td>Sent by function key F9.</td>
</tr>
<tr>
<td>key_f10</td>
<td>kf10</td>
<td>ka</td>
<td>Sent by function key F10.</td>
</tr>
<tr>
<td>key_f11</td>
<td>kf11</td>
<td>k&lt;</td>
<td>Sent by function key F11.</td>
</tr>
<tr>
<td>key_f12</td>
<td>kf12</td>
<td>k&gt;</td>
<td>Sent by function key F12.</td>
</tr>
<tr>
<td>key_help</td>
<td>khlp</td>
<td>kq</td>
<td>Help key.</td>
</tr>
<tr>
<td>key_home</td>
<td>khome</td>
<td>kh</td>
<td>Sent by home key.</td>
</tr>
<tr>
<td>key_ic</td>
<td>kich1</td>
<td>kl</td>
<td>Sent by insert character/enter insert mode key.</td>
</tr>
<tr>
<td>key_il</td>
<td>kil1</td>
<td>kA</td>
<td>Sent by insert line key.</td>
</tr>
<tr>
<td>key_left</td>
<td>kcub1</td>
<td>kl</td>
<td>Sent by terminal cursor left key.</td>
</tr>
<tr>
<td>key_ll</td>
<td>kll</td>
<td>kH</td>
<td>Sent by home-down key.</td>
</tr>
<tr>
<td>key_newline</td>
<td>knl</td>
<td>kn</td>
<td>New-line key.</td>
</tr>
<tr>
<td>key_nextPane</td>
<td>knpn</td>
<td>kv</td>
<td>Next-pane key.</td>
</tr>
<tr>
<td>key_npage</td>
<td>knp</td>
<td>kN</td>
<td>Sent by next-page key.</td>
</tr>
<tr>
<td>key_ppage</td>
<td>kpp</td>
<td>kP</td>
<td>Sent by previous-page key.</td>
</tr>
<tr>
<td>key_prevCmd</td>
<td>kpcmd</td>
<td>kp</td>
<td>Sent by previous-command key.</td>
</tr>
<tr>
<td>key_quit</td>
<td>kquit</td>
<td>kQ</td>
<td>Quit key.</td>
</tr>
<tr>
<td>key_right</td>
<td>kcuf1</td>
<td>kr</td>
<td>Sent by terminal cursor right key.</td>
</tr>
<tr>
<td>key_scroll_left</td>
<td>ksc1</td>
<td>kz</td>
<td>Scroll left.</td>
</tr>
<tr>
<td>key_scroll_right</td>
<td>kscr</td>
<td>kZ</td>
<td>Scroll right.</td>
</tr>
<tr>
<td>key_select</td>
<td>ksel</td>
<td>kU</td>
<td>Select key</td>
</tr>
<tr>
<td>key_sf</td>
<td>kind</td>
<td>kF</td>
<td>Sent by scroll-forward/down key.</td>
</tr>
<tr>
<td>key_smap_in1</td>
<td>kmpf1</td>
<td>Kv</td>
<td>Input for special mapped key 1.</td>
</tr>
<tr>
<td>key_smap_out1</td>
<td>kmp1</td>
<td>KV</td>
<td>Output for mapped key 1.</td>
</tr>
<tr>
<td>key_smap_in2</td>
<td>kmpf2</td>
<td>Kw</td>
<td>Input for special mapped key 2.</td>
</tr>
<tr>
<td>key_smap_out2</td>
<td>kmp2</td>
<td>KW</td>
<td>Output for mapped key 2.</td>
</tr>
<tr>
<td>key_smap_in3</td>
<td>kmpf3</td>
<td>Kx</td>
<td>Input for special mapped key 3.</td>
</tr>
<tr>
<td>key_smap_out3</td>
<td>kmp3</td>
<td>KX</td>
<td>Output for mapped key 3.</td>
</tr>
<tr>
<td>key_smap_in4</td>
<td>kmpf4</td>
<td>Ky</td>
<td>Input for special mapped key 4.</td>
</tr>
<tr>
<td>key_smap_out4</td>
<td>kmp4</td>
<td>KY</td>
<td>Output for mapped key 4.</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>key_smap_in5</td>
<td>kmpf5</td>
<td>Kz</td>
<td>Input for special mapped key 5.</td>
</tr>
<tr>
<td>key_smap_out5</td>
<td>kmpt5</td>
<td>KZ</td>
<td>Output for mapped key 5.</td>
</tr>
<tr>
<td>key_sr</td>
<td>kri</td>
<td>kR</td>
<td>Sent by scroll-backward/up key.</td>
</tr>
<tr>
<td>key_stab</td>
<td>khts</td>
<td>k</td>
<td>Sent by set-tab key.</td>
</tr>
<tr>
<td>key_tab</td>
<td>ktab</td>
<td>ko</td>
<td>Tab key.</td>
</tr>
<tr>
<td>key_up</td>
<td>kcuu1</td>
<td>ku</td>
<td>Sent by terminal cursor up key.</td>
</tr>
<tr>
<td>keypad_local</td>
<td>rmkx</td>
<td>ke</td>
<td>Ends keypad transmit mode.</td>
</tr>
<tr>
<td>keypad_xmit</td>
<td>smkx</td>
<td>ks</td>
<td>Puts terminal in keypad transmit mode.</td>
</tr>
<tr>
<td>lab_f0</td>
<td>f0</td>
<td>l0</td>
<td>Labels function key F0 if not F0.</td>
</tr>
<tr>
<td>lab_f1</td>
<td>lf1</td>
<td>l1</td>
<td>Labels function key F1 if not F1.</td>
</tr>
<tr>
<td>lab_f2</td>
<td>lf2</td>
<td>l2</td>
<td>Labels function key F2 if not F2.</td>
</tr>
<tr>
<td>lab_f3</td>
<td>lf3</td>
<td>l3</td>
<td>Labels function key F3 if not F3.</td>
</tr>
<tr>
<td>lab_f4</td>
<td>lf4</td>
<td>l4</td>
<td>Labels function key F4 if not F4.</td>
</tr>
<tr>
<td>lab_f5</td>
<td>lf5</td>
<td>l5</td>
<td>Labels function key F5 if not F5.</td>
</tr>
<tr>
<td>lab_f6</td>
<td>lf6</td>
<td>l6</td>
<td>Labels function key F6 if not F6.</td>
</tr>
<tr>
<td>lab_f7</td>
<td>lf7</td>
<td>l7</td>
<td>Labels function key F7 if not F7.</td>
</tr>
<tr>
<td>lab_f8</td>
<td>lf8</td>
<td>l8</td>
<td>Labels function key F8 if not F8.</td>
</tr>
<tr>
<td>lab_f9</td>
<td>lf9</td>
<td>l9</td>
<td>Labels function key F9 if not F9.</td>
</tr>
<tr>
<td>lab_f10</td>
<td>lf10</td>
<td>la</td>
<td>Labels function key F10 if not F10.</td>
</tr>
<tr>
<td>meta_on</td>
<td>smm</td>
<td>mm</td>
<td>Enables meta mode (8th bit).</td>
</tr>
<tr>
<td>meta_off</td>
<td>rmm</td>
<td>mo</td>
<td>Disables meta mode.</td>
</tr>
<tr>
<td>newline</td>
<td>nel</td>
<td>nw</td>
<td>Performs newline function (behaves like a carriage return followed by a linefeed).</td>
</tr>
<tr>
<td>pad_char</td>
<td>pad</td>
<td>pc</td>
<td>Pad character (instead of null).</td>
</tr>
<tr>
<td>parm_dch</td>
<td>dch</td>
<td>DC</td>
<td>Deletes #1 characters. (PG*)</td>
</tr>
<tr>
<td>parm_delete_line</td>
<td>dl</td>
<td>DL</td>
<td>Deletes #1 lines. (PG*)</td>
</tr>
<tr>
<td>parm_down_cursor</td>
<td>cud</td>
<td>DO</td>
<td>Moves cursor down #1 lines. (PG*)</td>
</tr>
<tr>
<td>parm_ich</td>
<td>ich</td>
<td>IC</td>
<td>Inserts #1 blank characters. (PG*)</td>
</tr>
<tr>
<td>parm_index</td>
<td>indn</td>
<td>SF</td>
<td>Scrolls forward #1 lines. (PG)</td>
</tr>
<tr>
<td>parm_insert_line</td>
<td>il</td>
<td>AL</td>
<td>Adds #1 new blank lines. (PG*)</td>
</tr>
<tr>
<td>parm_left_cursor</td>
<td>cub</td>
<td>LE</td>
<td>Moves cursor left #1 spaces. (PG)</td>
</tr>
<tr>
<td>parm_right_cursor</td>
<td>cuf</td>
<td>RI</td>
<td>Moves cursor right #1 spaces. (PG*)</td>
</tr>
<tr>
<td>parm_rindex</td>
<td>rin</td>
<td>SR</td>
<td>Scrolls backward #1 lines. (PG)</td>
</tr>
<tr>
<td>parm_up_cursor</td>
<td>cuu</td>
<td>UP</td>
<td>Moves cursor up #1 lines. (PG*)</td>
</tr>
<tr>
<td>pkey_key</td>
<td>pfkey</td>
<td>pk</td>
<td>Programs function key F1 to type string #2.</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pkey_local</td>
<td>pl</td>
<td></td>
<td>Programs function key F1 to execute string #2.</td>
</tr>
<tr>
<td>pkey_xmit</td>
<td>pfx</td>
<td>px</td>
<td>Programs function key F1 to xmit string #2.</td>
</tr>
<tr>
<td>print_screen</td>
<td>mc0</td>
<td>ps</td>
<td>Prints contents of the screen.</td>
</tr>
<tr>
<td>prtr_off</td>
<td>mc4</td>
<td>pf</td>
<td>Disables the printer.</td>
</tr>
<tr>
<td>prtr_on</td>
<td>mc5</td>
<td>po</td>
<td>Enables the printer.</td>
</tr>
<tr>
<td>repeat_char</td>
<td>rep</td>
<td>rp</td>
<td>Repeats character #1 twice. (PG*)</td>
</tr>
<tr>
<td>reset_1string</td>
<td>rs1</td>
<td>r1</td>
<td>Resets terminal to known modes.</td>
</tr>
<tr>
<td>reset_2string</td>
<td>rs2</td>
<td>r2</td>
<td>Resets terminal to known modes.</td>
</tr>
<tr>
<td>reset_3string</td>
<td>rs3</td>
<td>r3</td>
<td>Resets terminal to known modes.</td>
</tr>
<tr>
<td>reset_file</td>
<td>rf</td>
<td>rf</td>
<td>Identifies the file containing reset string.</td>
</tr>
<tr>
<td>restore_cursor</td>
<td>rc</td>
<td>rc</td>
<td>Restores cursor to position of last sc.</td>
</tr>
<tr>
<td>row_address</td>
<td>vpa</td>
<td>cv</td>
<td>Positions cursor to an absolute vertical position (set row). (PG)</td>
</tr>
<tr>
<td>save_cursor</td>
<td>sc</td>
<td>sc</td>
<td>Saves cursor position. (P)</td>
</tr>
<tr>
<td>scroll_forward</td>
<td>ind</td>
<td>sf</td>
<td>Scrolls text up. (P)</td>
</tr>
<tr>
<td>scroll_reverse</td>
<td>ri</td>
<td>sr</td>
<td>Scrolls text down. (P)</td>
</tr>
<tr>
<td>set_attributes</td>
<td>sgr</td>
<td>sa</td>
<td>Defines the video attributes. (PG*)</td>
</tr>
<tr>
<td>set_tab</td>
<td>hts</td>
<td>st</td>
<td>Sets a tab in all rows, current column.</td>
</tr>
<tr>
<td>set_window</td>
<td>wind</td>
<td>wi</td>
<td>Indicates current window is lines #1 to #2 cols #3 to #4.</td>
</tr>
<tr>
<td>tab</td>
<td>ht</td>
<td>a</td>
<td>Tabs to next 8-space hardware tab stop.</td>
</tr>
<tr>
<td>to_status_line</td>
<td>tsl</td>
<td>ts</td>
<td>Moves to status line, column #1.</td>
</tr>
<tr>
<td>underline_char</td>
<td>uc</td>
<td>uc</td>
<td>Underlines one character and moves beyond it.</td>
</tr>
<tr>
<td>up_half_line</td>
<td>hu</td>
<td>hu</td>
<td>Indicates superscript (reverse 1/2 linefeed).</td>
</tr>
<tr>
<td>init_prog</td>
<td>iprog</td>
<td>iP</td>
<td>Locates the program for init.</td>
</tr>
<tr>
<td>key_a1</td>
<td>ka1</td>
<td>K1</td>
<td>Specifies upper left of keypad.</td>
</tr>
<tr>
<td>key_a3</td>
<td>ka3</td>
<td>K3</td>
<td>Specifies upper right of keypad.</td>
</tr>
<tr>
<td>key_b2</td>
<td>kb2</td>
<td>K2</td>
<td>Specifies center of keypad.</td>
</tr>
<tr>
<td>key_c1</td>
<td>kc1</td>
<td>K4</td>
<td>Specifies lower left of keypad.</td>
</tr>
<tr>
<td>key_c3</td>
<td>kc3</td>
<td>K5</td>
<td>Specifies lower right of keypad.</td>
</tr>
<tr>
<td>prtr_non</td>
<td>mc5p</td>
<td>pO</td>
<td>Enables the printer for #1 bytes.</td>
</tr>
</tbody>
</table>
Notes to table:
(P) Indicates that padding can be specified
(G) Indicates that the string is passed through `tparm` with parameters as given (#i)
(*) Indicates that padding can be based on the number of lines affected
(#i) Indicates the \textit{i}th parameter

Example

The following is an uncompiled \texttt{terminfo} entry for the \texttt{xterm} terminal type:

```
xterm\|vs100\|xterm terminal emulator,
   ind=\^J, cols#80, lines#25,  
   clear=E[HE[2J, cubl=\^H, am, cup=E[%i%p1%d;%p2%dH,  
   cufl=E[C, cuu1=E[A, el=E[K, ed=E[J,  
   cud=E[%p1%dB, cuu=E[%p1%dA, cub=E[%p1%dD,  
   cuf=E[%p1%dC, km,  
   smso=E[7m, rmso=E[m, smul@, rmul@,  
   bold=E[1m, rev=E[7m, blink=@, sgr0=E[m,  
   rs1=E[i;1;3;4;5;6lE[7hE[mE[rE[2JE[H, rs2=\@  
   kfl=EOP, kf2=EOQ, kf3=EOR, kf4=EOS, ht=\^I, ri=EM,  
   vt@, xon@, csr=E[%i%p1%d;%p2%dr,  
   il=E[%p1%dL, dl=E[%p1%dM, ill=E[L, dll=E[M,  
   ich=E[%p1%d@, dch=E[%p1%dP, ichl=E[@, dchl=E[P,  
   use=vt100-am,
```

The first line of the \texttt{xterm} entry contains two names for the terminal type (\texttt{xterm} and \texttt{vs100}), and a third name that fully describes the terminal. When the \texttt{terminfo} entry is compiled with the \texttt{tic} command, entries are made in \texttt{/usr/lib/terminfo/x/xterm} and \texttt{/usr/lib/terminfo/v/vs100}, unless the \texttt{TERMINFO} environment variable was used to redefine the default path. The \texttt{TERMINFO} environment variable is useful when testing a new entry, or when you do not have write permission for the \texttt{/usr/lib/terminfo} directory tree. For example, if the \texttt{TERMINFO} environment variable is set to \texttt{/usr/raj/test}, the \texttt{tic} command places the compiled \texttt{terminfo} entries into \texttt{/usr/raj/test/x/xterm} and \texttt{/usr/raj/test/v/vs100}. The \texttt{TERMINFO} environment variable is also referenced by programs that use \texttt{terminfo} (such as \texttt{vi}), so the new entry can be tested right away.

The second line of the \texttt{xterm} entry says that pressing a \texttt{Ctrl-J} causes the screen to scroll up, and that the screen dimensions are 80 columns by 24 lines.
The third line of the entry sets the string that clears the screen (ESCAPE followed by "[H", another ESCAPE, and then the string "[2J"), defines \texttt{<Ctrl-H>} as the backspace key, and declares that the terminal has automatic margins. The string for relative cursor movement is also specified, using \texttt{terminfo} parameter syntax.

The rest of the capabilities are declared likewise. The last line of the entry reads "\texttt{use=vt100-am}" , meaning that the \texttt{vt100-am} terminal entry should be read first as the basis for the \texttt{xterm} terminal entry, with the capabilities explicitly defined overriding their default \texttt{vt100-am} values. Note that the \texttt{smul}, \texttt{rmul}, \texttt{vt}, and \texttt{xon} capabilities are removed by following them with an @ (at sign).

\textbf{Related Information}

- Functions: \texttt{curses(3)}
- Commands: \texttt{tic(1)}

termios.h

Purpose

Defines the structure of the termios file, which provides the terminal interface for POSIX compatibility.

Description

The `/usr/include/termios.h` header file contains information used by system calls that apply to terminal files. The definitions, values, and structure in this file are required for compatibility with the Institute of Electrical and Electronics Engineers (IEEE) P1003.1 Portable Operating System Interface for Computer Environments (POSIX) standard.

The general terminal interface information is contained in the `termio.h` header file. The `termio` structure in the `termio.h` header file defines the basic input, output, control, and line discipline modes. If a calling program is identified as requiring POSIX compatibility, the `termios` structure and additional, POSIX control packet information in the `termios.h` header file are implemented. Window and terminal size operations use the `winsize` structure, which is defined in the `ioctl.h` header file. The `termios` structure in the `termios.h` header file contains the following fields:

**c_iflag**

Describes the basic terminal input control. The initial input control value is all bits clear. The possible input modes are:

- **IGNBRK**
  
  Ignores the break condition. If set, the break condition is not put on the input queue and is therefore not read by any process.

- **BRKINT**
  
  Interrupts signal on the break condition. If set, the break condition generates an interrupt signal and flushes both the input and output queues.

- **IGNPAR**
  
  Ignores characters with parity errors. If set, characters with other framing and parity errors are ignored.

- **PARMRK**
  
  Marks parity errors. If set, a character with a framing or parity error that is not ignored is read as the 3-character sequence: 0377, 0, x, where the x variable is the data of the character received in error. If the ISTRIP mode is not set, then a valid character of 0377 is read as 0377, 0377 to avoid ambiguity. If the PARMRK mode is not set, a framing or parity error that is not ignored is read as the null character.
INPCK
Enables input parity checking. If set, input parity checking is enabled. If not set, input parity checking is disabled. This allows for output parity generation without input parity errors.

ISTRIP
Strips characters. If set, valid input characters are first stripped to 7 bits; otherwise all 8 bits are processed.

INLCR
Maps new-line character (NL) to carriage return (CR) on input. If set, a received NL character is translated into a CR character.

IGNCR
Ignores CR character. If set, a received CR character is ignored (not read).

ICRNL
Maps CR character to NL character on input. If set, a received CR character is translated into a NL character.

IUCLC
Maps uppercase to lowercase on input. If set, a received uppercase, alphabetic character is translated into the corresponding lowercase character.

IXON
Enables start and stop output control. If set, a received STOP character suspends output, and a received START character restarts output. The START and STOP characters perform flow control functions but are not read.

IXANY
Enables any character to restart output. If set, any input character restarts output that was suspended.

IXOFF
Enables start and stop input control. If set, the system transmits a STOP character when the input queue is nearly full and a START character when enough input has been read that the queue is nearly empty again.

IMAXBEL
Echoes the ASCII BEL character if the input stream overflows. Further input is not stored, but any input present in the input stream is not lost. If not set, the BEL character is not echoed, and the input in the input queue is discarded if the input stream overflows.
c_oflag

Specifies how the system treats output. The initial output control value is all bits clear. The possible output modes are:

OPOST
Post-processes output. If set, output characters are processed as indicated by the remaining flags; otherwise, characters are transmitted without change.

OLCUC
Maps lowercase to uppercase on output. If set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with the IUCLC input mode.

ONLCR
Maps NL to CR-NL on output. If set, the NL character is transmitted as the CR-NL character pair.

OCRNL
Maps CR to NL on output. If set, the CR character is transmitted as the NL character.

ONOCR
Indicates no CR output at column 0. If set, no CR character is transmitted at column 0 (first position).

ONLRET
NL performs CR function. If set, the NL character is assumed to do the carriage return function. The column pointer is set to a value of 0 and the delay specified for carriage return is used. Otherwise the NL character is assumed to do the line feed function only; the column pointer remains unchanged. The column pointer is also set to a value of 0 if the CR character is actually transmitted.

The delay bits specify how long a transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. The actual delays depend on line speed and system load.

OFILL
Uses fill characters for delay. If set, fill characters are transmitted for a delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay.
OFDEL
Sets fill characters to the DEL value. If set, the fill character is DEL. If this flag is not set, the fill character is null.

NLDLY
Selects the newline character delays. This is a mask to use before comparing to NL0 and NL1.

NL0 Specifies no delay.
NL1 Specifies one delay of approximately 0.10 seconds. If ONLRET is set, the carriage return delays are used instead of the newline delays. If OFILL is set, two fill characters are transmitted.

CRDLY
Selects the carriage return delays. This is a mask to use before comparing to CR0, CR1, CR2, and CR3.

CR0 Specifies no delay.
CR1 Specifies that the delay is dependent on the current column position. If OFILL is set, this delay transmits two fill characters.
CR2 Specifies one delay of approximately 0.10 seconds. If OFILL is set, this delay transmits four fill characters.
CR3 Specifies one delay of approximately 0.15 seconds.

TABDLY
Selects the horizontal tab delays. This is a mask to use before comparing to TAB0, TAB1, TAB2, and TAB3. If OFILL is set, any of these delays transmit two fill characters.

TAB0 Specifies no delay.
TAB1 Specifies that the delay is dependent on the current column position. If OFILL is set, two fill characters are transmitted.
TAB2 Specifies one delay of approximately 0.10 seconds.
TAB3 Specifies that tabs are to be expanded into spaces.

BSDLY
Selects the backspace delays. This is a mask to use before comparing to BS0 and BS1.
BS0  Specifies no delay.
BS1  Specifies one delay of approximately 0.05 seconds. If OFILL is set, this delay transmits one fill character.

VTDLY
Selects the vertical-tab delays. This is a mask to use before comparing to VT0 and VT1.

VT0  Specifies no delay.
VT1  Specifies one delay of approximately 2 seconds.

FFDLY
Selects the formfeed delays. This is a mask to use before comparing to FFO and FF1.

FF0  Specifies no delay.
FF1  Specifies one delay of approximately 2 seconds.

c_cflag
Describes the hardware control of the terminal. In addition to the basic control modes, this field uses the following control characters:

CBAUD
Specifies baud rate. These bits specify the baud rate for a connection. For any particular hardware, impossible speed changes are ignored.

B0   Hangs up. The zero baud rate is used to hang up the connection. If B0 is specified, the 'data terminal ready' signal is not asserted. Normally, this disconnects the line.

B50  50 baud.
B75  75 baud.
B110 110 baud.
B134 134.5 baud.
B150 150 baud.
B200 200 baud.
B300 300 baud.
B600 600 baud.
B600 600 baud.
B1200 1200 baud.
B1800 1800 baud.
B2400 2400 baud.
B4800 4800 baud.
B9600 9600 baud.
B19200
  19200 baud.
B38400
  38400 baud.
EXTA External A.
EXTB External B.
CSIZE
  Specifies the character size. These bits specify the character
  size in bits for both transmit and receive operations. This
  size does not include the parity bit, if any.
CS5  5 bits.
CS6  6 bits.
CS7  7 bits.
CS8  8 bits.
CSTOPB
  Specifies number of stop bits. If set, 2 stop bits are sent;
  otherwise, only 1 stop bit is sent. Higher baud rates require 2
  stop bits. (At 110 baud, for example, 2 stop bits are
  required.)
CREAD
  Enables receiver. If set, the receiver is enabled. Otherwise,
  characters are not received.
PARENB
  Enables parity. If set, parity generation and detection is
  enabled and a parity bit is added to each character.
PARODD
  Specifies odd parity. If parity is enabled, this specifies odd
  parity. If not set, even parity is used.
HUPCL
  Hangs up on last close. If set, the line is disconnected when
  the last process closes the line or when the process
  terminates (when the 'data terminal ready' signal drops).
CLOCAL
   Specifies a local line. If set, the line is assumed to have a local, direct connection with no modem control. If not set, modem control (dialup) is assumed.

CIBAUD
   Specifies the input baud rate if it is different than the output rate.

PAREXT
   Specifies extended parity for mark and space parity.

The initial hardware control value after an open is B300, CS8, CREAD, and HUPCL

c_lflag
   Controls various terminal functions. The initial value after an open is all bits clear. In addition to the basic modes, this field uses the following mask name symbols:

ISIG   Enables signals. If set, each input character is checked against the INTR and QUIT special control characters. If a character matches one of these control characters, the function associated with that character is performed. If the ISIG function is not set, checking is not done.

ICANON
   Enables canonical input. If set, turns on canonical processing, which enables the erase and kill edit functions as well as the assembly of input characters into lines delimited by NL, EOF, and EOL.

   If the ICANON function is not set, read requests are satisfied directly from the input queue. In this case, a read request is not satisfied until one of the following conditions is met: a) the minimum number of characters specified by MIN are received; or b) the time-out value specified by TIME has expired since the last character was received. This allows bursts of input to be read, while still allowing single character input. The MIN and TIME values are stored in the positions for the EOF and EOL characters, respectively. The time value represents tenths of seconds.

XCASE
   Enables canonical uppercase and lowercase presentation. If set along with the ICANON function, an uppercase letter (or the uppercase letter translated to lowercase by the IUCLC input mode) is accepted on input by preceding it with a \ (backslash) character. The output is then preceded by a backslash character.
ECHO Enables echo. If set, characters are displayed on the terminal screen as they are received.

ECHOE
Echoes erase character as BS-SP-BS. If the ECHO and ECHOE functions are both set and ECHOPRT is not set, the erase character is implemented as a backspace, a space, and then another backspace (ASCII BS-SP-BS). This clears the last character from the screen. If ECHOE is set, but ECHO is not set, the erase character is implemented as ASCII SP-BS.

ECHOK
Echoes NL after kill. If ECHOK is set and ECHOKE is not set, a newline function is performed to clear the line after a KILL character is received. This emphasizes that the line is deleted. Note that an escape character preceding the ERASE or KILL character removes any special function.

ECHONL
Echoes NL. If ECHONL is set, the line is cleared when a newline function is performed whether or not the ECHO function is set. This is useful for terminals that are set to local echo (also referred to as half-duplex). Unless an escape character precedes an EOF, the EOF character is not displayed. Because the ASCII EOT character is the default end-of-file character, this prevents terminals that respond to the EOT character from hanging up.

NOFLSH
Disables queue flushing. If set, the normal flushing of the input and output queues associated with the quit and interrupt characters is not done.

The ICANON, XCASE, ECHO, ECHOE, ECHOK, ECHONL, and NOFLSH special input functions are possible only if the ISIG function is set. These functions can be disabled individually by changing the value of the control character to an unlikely or impossible value (for example, 0377 octal or 0xFF)

ECHOCCTL
Echoes control characters as 'X, where the X variable is the character given by adding 100 octal to the code of the control character. The ASCII DEL character is echoed as '? and the ASCII TAB, NL, and START characters are not
echoed. Unless an escape character precedes an EOF, the EOF character is not displayed. Because the ASCII EOT character is the default End-of-File character, this mask prevents terminals that respond to the EOT character from hanging up.

ECHOPRT
Echoes the first ERASE and WERASE character in a sequence as a \ (backslash), and then erases the characters. Subsequent ERASE and WERASE characters echo the characters being erased (in reverse order).

ECHOKE
Echoes the kill character by erasing from the screen each character on the line.

FLUSHO
Flushes the output. When this bit is set by typing the FLUSH character, data written to the terminal is discarded. A terminal can cancel the effect of typing the FLUSH character by clearing this bit.

PENDIN
Reprints any input that has not yet been read when the next character arrives as input.

IEXTEN
Enables extended (implementation-defined) functions to be recognized from the input data. If this bit is not set, implementation-defined functions are not recognized, and the corresponding input characters are processed as described for ICANON, ISIG, IXON, and IXOFF.

TOSTOP
Sends a SIGTTOU signal when a process in a background process group tries to write to its controlling terminal. The SIGTTOU signal stops the members of the process group. If job control is not supported, this symbol is ignored.

c_cc Specifies an array that defines the special control characters. The relative positions and initial values for each function are:

VINTR
Indexes the INTR control character (Ctrl-Backspace), which sends a SIGINT signal to stop all processes controlled by this terminal.
VQUIT
Indexes the QUIT control character (Ctrl-v or Ctrl-l), which sends a SIGQUIT signal to stop all processes controlled by this terminal and writes a core image file into the current working directory.

VERASE
Indexes the ERASE control character (Backspace), which erases the preceding character. The ERASE character does not erase beyond the beginning of the line (delimited by a NL, EOL, EOF, or EOL2 character).

VKILL
Indexes the KILL control character (Ctrl-u), which deletes the entire line (delimited by a NL, EOL, EOF, or EOL2 character).

VEOF Indexes the EOF control character (Ctrl-d), which can be used at the terminal to generate an end-of-file. When this character is received, all characters waiting to be read are immediately passed to the program without waiting for a new line, and the EOF is discarded. If the EOF is at the beginning of a line (no characters are waiting), zero characters are passed back, which is the standard End-of-File.

VEOL Indexes the EOL control character (Ctrl-@ or ASCII null), which is an additional line delimiter that is not normally used.

VEOL2
Indexes the EOL2 control character (Ctrl-@ or ASCII null), which is an additional line delimiter that is not normally used.

VSTART
Indexes the START control character (Ctrl-q), which resumes output that has been suspended by a STOP character. START characters are ignored if the output is not suspended.
VSUSP
Indexes the SUSP control character (Ctrl-z), which causes a SIGTSTP signal to be sent to all foreground processes controlled by this terminal. This character is recognized during input if the ISIG flag is enabled. If job control is not supported, this character is ignored.

VDSUSP
Indexes the DSUSP control character (Ctrl-y), which causes a SIGTSTP signal to be sent to all foreground processes controlled by this terminal. This character is recognized when the process attempts to read the DSUSP character. If job control is not supported, this character is ignored.

VSTOP
Indexes the STOP control character (Ctrl-s), which can be used to temporarily suspend output. This character is recognized during both input and output if the IXOFF (input control) or IXON (output control) flag is set.

VREPRINT
Indexes the REPRINT control character (Ctrl-r), which reprints all characters that are preceded by a NL character and that have not been read.

VDISCRD
Indexes the DISCARD control character (Ctrl-o), which causes all output to be discarded until another DISCARD character is typed, more input is received, or the condition is cleared by a program.

VWERASE
Indexes the WERASE control character (Ctrl-w), which erases the preceding word. The WERASE character does not erase beyond the beginning of the line (delimited by a NL, EOL, EOF, or EOL2 character).

VLNEXT
Indexes the LNEXT (literal next) control character (Ctrl-v), which causes the special meaning of the next character to be ignored, so that characters can be input without being interpreted by the system.

The character values for INTR, QUIT, SWTCH, ERASE, KILL, EOF, and EOL can be changed. The ERASE, KILL, and EOF characters can also be escaped (preceded with a backslash) so that no special processing is done.
The following values for the `optional_actions` parameter of the `tcsetattr()` function are also defined in the `termios.h` header file:

- **TCSANOW**: Immediately sets the parameters associated with the terminal from the referenced `termios` structure.
- **TCSADRAIN**: Waits until all output written to the object file has been transmitted before setting the terminal parameters from the `termios` structure.
- **TCSAFLUSH**: Waits until all output written to the object file has been transmitted and all input received but not read has been discarded before setting the terminal parameters from the `termios` structure.

The following values for the `queue_selector` parameter of the `tcflush()` function are also defined in this header file:

- **TCIFLUSH**: Flushes data that is received but not read.
- **TCOFLUSH**: Flushes data that is written but not transmitted.
- **TCIOFLUSH**: Flushes both data that is received but not read and data that is written but not transmitted.

The following values for the `action` parameter of the `tcflow()` system call are also defined in the `termios.h` header file:

- **TCOOFF**: Suspends the output of data by the object file named in the `tcflow()` function.
- **TCOON**: Restarts data output that was suspended by the TCOOFF parameter.
- **TCIOFF**: Transmits a stop character to stop data transmission by the terminal device.
- **TCION**: Transmits a start character to start or restart data transmission by the terminal device.

**Files**

```
/usr/include/sys/termios.h
```

The path to the `termios.h` header file.

**Related Information**

- Functions: `ioctl(2), sigvec(2)`
- Commands: `csh(1), getty(1), sh(1), stty(1), tset(1)`
tty

Purpose  General terminal interface

Synopsis  #include <sys/termios.h>

Description  This section describes both a particular special file /dev/tty and the terminal drivers used for conversational computing. Much of the terminal interface performance is governed by the settings of a terminal’s termios structure. This structure provides definitions for terminal input and output processing, control and local modes, and so on. These definitions are found in the termios.h header file.

Line Disciplines

OSF/1 provides different line disciplines for controlling communications lines. In this version of the system there are two disciplines available for use with terminals:

Standard  Standard POSIX-compliant terminal driver, with features for job control, sessions, termios.h support, and so on.

Kanji-support  Standard POSIX-compliant terminal driver with support for the Japanese character set, Kanji. The Kanji terminal driver provides support for multibyte characters.

Line discipline switching is accomplished with the TIOCSETD ioctl:

int ldisc = LDISC;
ioctl(f, TIOCSETD, &ldisc);

Here, LDISC is TTYDISC for the standard POSIX tty driver and KJIDISC for the Kanji terminal driver. By convention, the standard (POSIX) tty driver is discipline 0 (zero) and the Kanji tty driver is discipline 8. Other disciplines exist for special purposes, such as use of communications lines for network connections. The current line discipline can be obtained with the TIOCGETD ioctl. Pending input is discarded when the line discipline is changed.

All of the low-speed asynchronous communications ports can use any of the available line disciplines, no matter what hardware is involved.
The Controlling Terminal

OSF/1 supports the concept of a controlling terminal. Any process in the system can have a controlling terminal associated with it. Certain events, such as the delivery of keyboard generated signals (for example, interrupt, quit, suspend), affect all the processes in the process group associated with the controlling terminal. The controlling terminal also determines the physical device that is accessed when the indirect device /dev/tty is opened.

In earlier versions of UNIX systems, a controlling terminal was implicitly assigned to a process if, at the time an open was done on the terminal, the terminal was not the controlling terminal for any process, and if the process doing the open did not have a controlling terminal. In OSF/1, in accordance with POSIX 1003.1, a process must be a session leader to allocate a controlling terminal. In addition, the allocation is now done explicitly with a call to ioctl(). (This implies that the O_NOCTTY flag to the open() function is ignored.) The following example illustrates the correct sequence for obtaining a controlling tty (no error checking is shown). This code fragment calls the setsid() function to make the current process the group and session leader, and to remove any controlling tty that the process may already have. It then opens the console device and attaches it to the current session as the controlling terminal. Note that the process must not already be a session or process group leader, and the console must not already be the controlling tty of any other session.

```
(void) setsid(); /* become session leader and */
            /* lose controlling tty */
fd = open("/dev/console", O_RDWR);
(void) ioctl(fd, TIOCSCTTY, 0);
```

A process can remove the association it has with its controlling terminal by opening the /dev/tty file and issuing the following call:

```
ioclt(fd, TIOCNOTTY, 0);
```

For example:
```
fd = open("/dev/tty", O_RDWR);
if (fd >= 0) {
    ioctl(fd, TIOCNOTTY, 0);
    close(fd);
}
When a control terminal file is closed, pending input is removed, and pending output is sent to the receiving device.

When a terminal file is opened, the process blocks until a carrier signal is detected. If the open() function is called with the O_NONBLOCK flag set, however, the process does not wait. Instead, the first read() or write() call will wait for carrier to be established. If the CLOCAL mode is set in the termios structure, the driver assumes that modem control is not in effect, and open(), read(), and write() therefore proceed without waiting for a carrier signal to be established.

Process Groups

In OSF/1, each process belongs to a process group with a specific process group ID. Each process belongs to the process group of its creating process. This enables related processes to be signalled. Process group IDs are unique identifiers that cannot be used for other system process groups until the original process group is disbanded. Each process group also has a group leader process. A process group leader has the same process ID as its process group.

Each process group belongs to a session. Each process in the process group also belongs to the process group's session. A process which is not the process group leader can create its own session and process group with a call to the setsid() function. That calling process then becomes the session leader of the new session and of the new process group. The new session has no controlling terminal until the session leader assigns one to it. The calling process's ID is assigned to the new process group. With the setpgid() function, other processes can be added to a process group.

A controlling terminal can have a distinguished process group associated with it known as the foreground process group. The terminal's foreground process group is the one that receives signals generated by the INTR, QUIT, and SUSP special control characters. Certain operations on the terminal are also restricted to processes in the terminal's foreground process group (see "Terminal Access Control"). A terminal's foreground process group may be changed by calling the tcsetpgrp() function. A terminal's current foreground process group may be obtained by calling the tcgetpgrp() function.

Input Processing Modes

The terminal drivers have two major modes, characterized by the kind of processing that takes place on the input characters:

Canonical If a terminal is in canonical mode, input is collected and processed one line at a time. Lines are terminated by a newline (\n), End-of-File (EOF), or End-of-Line (EOL) character. A read request is not returned until the line has been terminated, or a signal has been received. The maximum number of bytes of unread input allowed on an input terminal is 255 bytes. If the maximum number of unread bytes
exceeds 255 bytes, the behavior of the driver depends on the setting of the IMAXBEL input flag (see "Input Editing").

Erase and kill processing is performed on input that has not been terminated by one of the line termination characters. Erase processing removes the last character in the line, kill processing removes the whole line.

**Noncanonical**

This mode eliminates erase and kill processing, making input characters available to the user program as they are typed. Input is not processed into lines. The received bytes are processed according to the MIN and TIME elements of the c_cc array in the *termios* structure.

**MIN** MIN is the minimum number of bytes the terminal can receive in noncanonical mode before a read is considered successful.

**TIME** TIME, measured in 0.1 second granularity, times out sporadic input.

These cases are summarized as follows:

- **MIN>0, TIME>0**
  - In this case, TIME is an interbyte timer that is activated after the first byte of the input line is received, and reset after each byte is received. The read operation is a success if MIN bytes are read before TIME runs out. If TIME runs out before MIN bytes have been received, the characters that were received are returned.

- **MIN>0, TIME=0**
  - In this case, only MIN is used. A queued `read()` waits until MIN bytes are received, or a signal is received.

- **MIN=0, TIME>0**
  - In this case, TIME is used as a read timer that starts when a `read()` call is made. The `read()` call is finished when one byte is read, or when TIME runs out.

- **MIN=0, TIME=0**
  - In this case, either the number of requested bytes or the number of currently available bytes is returned, depending on which is the lesser number. The `read()` function returns a zero if no data was read.

Canonical mode is entered by setting the ICANON flag of the c_iflag field in the terminal’s *termios* structure. Other input processing is performed according to the other flags set in the c_iflag and c_lflag fields.
Input Editing

A terminal ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring. Characters are only lost when:

- The system's character input buffers become completely choked, which is rare.
- The user has accumulated the maximum allowed number of input characters (MAX_INPUT) that have not yet been read by some program. Currently this limit is 255 characters. When this limit is reached, the terminal driver refuses to accept any further input and rings the terminal bell if IMAXBEL is set in the c_iflag field, or throws away all input and output without notice if this flag is not set.

Input characters are normally accepted in either even or odd parity with the parity bit being stripped off before the character is given to the program. The ISTRIP mask of the c_iflag field controls whether the parity bit is stripped (ISTRIP set) or not stripped (ISTRIP not set). By setting the PARENB flag in the c_cflag field, and either setting (not setting) the PARODD flag, it is possible to have input characters with EVEN (ODD) parity discarded or marked (see "Input Modes").

In all of the line disciplines, it is possible to simulate terminal input using the TIOCSTI ioctl, which takes, as its third argument, the address of a character. The system pretends that this character was typed on the argument terminal, which must be the control terminal for the process, unless the process has superuser privileges.

Input characters are normally echoed by putting them in an output queue as they arrive. This may be disabled by clearing the ECHO bit in the c_iflag word using the tcsetattr() call or the TIOCSETA, TIOCSETAW, or TIOCSETAF ioctls.

In canonical mode, terminal input is processed in units of lines. A program attempting to read will normally be suspended until an entire line has been received (but see the description of SIGTTIN in "Terminal Access Control"). No matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information. In read() requests, the O_NONBLOCK flag affects the read() operation behavior.
If O_NONBLOCK is not set, a `read()` request is blocked until data or a signal has been received. If the O_NONBLOCK flag is set, the `read()` request is not blocked, and one of the following situations holds:

- Some data may have been typed, but there may or may not be enough data to satisfy the entire `read` request. In either case, the `read()` function returns the data available, returning the number of bytes of data it read.
- If there is no data for the read operation, the `read()` returns a -1 with an error of EAGAIN.

During input, line editing is normally done with the erase special control character (VERASE) logically erasing the last character typed and the kill special control character (VKILL) logically erasing the entire current input line. These characters never erase beyond the beginning of the current input line or an EOF (End-of-File). These characters, along with the other special control characters, may be entered literally by preceding them with the literal-next character (VLNEXT — default `^
`).

The drivers normally treat either a newline character (`\n`), End-of-File character (EOF), or End-of-Line character (EOL) as terminating an input line, echoing a return and a line feed. If the ICRNL character bit is set in the `c_iflag` word then carriage returns are translated to newline characters on input, and are normally echoed as carriage return-linefeed sequences. If ICRNL is not set, this processing for carriage return is disabled, and it is simply echoed as a return, and does not terminate cooked mode input.

The POSIX terminal driver also provides two other editing characters in normal mode. The word-erase character, normally `<Ctrl-W>`, is a `c_cc` structure special control character VWERASE. This character erases the preceding word, but not any spaces before it. For the purposes of `<Ctrl-W>`, a word is defined as a sequence of nonblank characters, with tabs counted as blanks. However, if the ALTWERASE flag is set in the `c_iflag` word, then a word is considered to be any sequence of alphanumerics or underscores bounded by characters that are not alphanumerics or underscores. Finally, the reprint character, normally `<Ctrl-R>`, is a `c_cc` structure special control character VREPRINT. This character retypes the pending input beginning on a new line. Retyping occurs automatically in canonical mode if characters which would normally be erased from the screen are fouled by program output.
Input Modes

The **termios** structure has an input mode field **c_iflag**, which controls basic terminal input characteristics. These characteristics are masks that can be bitwise inclusive ORed. The masks include:

- **BRKINT**: An interrupt is signalled on a break condition.
- **ICRNL**: All carriage returns are mapped to newline characters when input.
- **IGNBRK**: Break conditions are ignored.
- **IGNCR**: Carriage returns are ignored.
- **IGNPAR**: Characters with parity errors are ignored.
- **INLCR**: Newline characters are mapped to carriage returns when input.
- **INPCK**: Parity checks are enabled on input.
- **ISTRIP**: The eighth bit (parity bit) is stripped on input characters.
- **IXOFF**: Stop/start characters are sent for input flow control enabled.
- **IXON**: Stop/start characters are recognized for output flow control.
- **IXANY**: Any char will restart output after stop.
- **IUCLC**: Map upper case to lower case on input.
- **PARMRK**: Parity errors are marked with a three character sequence.
- **IMAXBEL**: The bell is rung when the input queue fills.

The input mode mask bits can be combined for the following results:

- The setting of **IGNBRK** causes input break conditions to be ignored. If **IGNBRK** is not set, but **BRKINT** is set, the break condition has the same effect as if the **VINTR** control character had been typed. If neither **IGNBRK** nor **BRKINT** are set, then the break condition is input as a single character '\0'. If the **PARMRK** flag is set, then the input is read as three characters, '\377', '\0', and '\0'.

- The setting of **IGNPAR** causes a byte with a parity or framing error, except for breaks, to be ignored (that is, discarded). If **IGNPAR** is not set, but **PARMRK** is set, a byte with parity or framing error, except for breaks, is passed as the three characters '\377', '\0', and X, where X is the character data received in error. If the **ISTRIP** flag is not set, the valid character '\377' is passed as '\377', '377'. If both **PARMRK** and **IGNPAR** are not set, framing or parity errors, including breaks, are passed as the single character '\0'.

- The setting of **INPCK** enables input parity checking. If input parity checking is not enabled (**INPCK** not set), then characters with parity errors are simply passed through as is. The enabling/disabling of input parity checking is independent of the generation of parity on output.
Setting ISTRIP causes the eighth bit of the eight valid input bits to be stripped before processing. If this mask is not set, all eight bits are processed.

Setting INLCR causes a newline character to be read as a carriage return character. If the IGNCR flag is also set, the carriage return is ignored. If the IGNCR flag is not set, INLCR works as described earlier.

The STOP character (normally <Ctrl-S>) suspends output and the START character (normally <Ctrl-Q>) restarts output. Setting IXON enables stop/start output control, in which the START and STOP characters are not read, but rather perform flow control functions. Extra stop characters typed when output is already stopped have no effect, unless the start and stop characters are made the same, in which case output resumes. Disabling IXON causes the START and STOP characters to be read.

Setting IXOFF enables stop/start input control. When this flag is set, the terminal device will be sent STOP characters to halt the transmission of data when the input queue is in danger of overflowing (exceed MAX_INPUT). When enough characters have been read to reduce the amount of data queued to an acceptable level, a START character is sent to the device to allow it to continue transmitting data. This mode is useful when the terminal is actually another machine that obeys those conventions.

Input Echoing and Redisplay

The terminal driver has several modes for handling the echoing of terminal input, controlled by bits in the c_lflag field of the termios structure.

Hardcopy Terminals

When a hardcopy terminal is in use, the ECHOPRT bit is normally set in the local flags word. Characters which are logically erased are then printed out backwards preceded by \ (backslash) and followed by a / (slash) in this mode.

Erasing Characters from a CRT

When a CRT terminal is in use, the ECHOE bit may be set to cause input to be erased from the screen with a backspace-space-backspace sequence when character or word deleting sequences are used. The ECHOKE bit may be set as well, causing the input to be erased in this manner on line kill sequences as well.

Echoing of Control Characters

If the ECHOCTL bit is set in the local flags word, then nonprinting (control) characters are normally echoed as ^X (for some X) rather than being echoed unmodified; DELETE is echoed as ^?.
Output Processing

When one or more characters are written, they are actually transmitted to the terminal as soon as previously written characters have finished typing. (As noted above, input characters are normally echoed by putting them in the output queue as they arrive.) When a process produces characters more rapidly than the terminal can accept them, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold the program is resumed. Even parity is normally generated on output. If the NOEOT bit is set in the c_oflag word of the termios structure, the EOT character (\texttt{<Ctrl-D>}) is not transmitted, to prevent terminals that respond to it from hanging up.

The terminal drivers provide necessary processing for canonical and noncanonical mode output including delay generation for certain special characters and parity generation. Delays are available after backspaces (BSDLY), formfeeds (FFDLY), carriage returns (CRDLY), tabs (TABDLY) and newlines (NLDLY). The driver will also optionally expand tabs into spaces, where the tab stops are assumed to be set every eight columns, and optionally convert newlines to carriage returns followed by newline. Output process is controlled by bits in the c_oflag field of the termios structure. Refer to the write(2) reference manual page for a description of the O_NONBLOCK flag.

The terminal drivers provide for mapping from lowercase to uppercase (OLCUC) for terminals lacking lower case, and for other special processing on deficient terminals.

Finally, the terminal driver, supports an output flush character, normally \texttt{<Ctrl-O>}, which sets the FLUSHO bit in the local mode word, causing subsequent output to be flushed until it is cleared by a program or more input is typed. This character has effect in both canonical and noncanonical modes and causes any pending input to be retyped. An ioctl to flush the characters in the input or output queues, TIOCFUNFLUSH, is also available.

Uppercase Terminals

If the IUCLC bit in the c_iflag field is set in the tty flags, then all uppercase letters are mapped into the corresponding lowercase letter. The uppercase letter may be generated by preceding it by \texttt{\textbackslash} (backslash). Uppercase letters are preceded by a \texttt{\textbackslash} (backslash) when output. In addition, the following escape sequences will be generated on output and accepted on input if the XCASE bit is set in the c_iflag word:

For: \texttt{\textbackslash} \texttt{1} \texttt{~} \texttt{[} \texttt{]}

Use: \texttt{\textbackslash\textbackslash} \texttt{!} \texttt{\textbackslash} \texttt{\textbackslash}
Line Control and Breaks

There are several ioctl calls available to control the state of the terminal line. The TIOCSBRK ioctl will set the break bit in the hardware interface causing a break condition to exist; this can be cleared (usually after a delay with sleep(3)) by TIOCCBRK. The tcsendbreak() can also be used to cause a break condition for a specified amount of time. Break conditions in the input are handled according to the c_iflag field settings for the termios structure. Refer to the section "Input Modes" for a complete listing of the c_iflag field settings. The TIOCCDTR ioctl will clear the data terminal ready condition; it can be set again by TIOCSDTR.

When the carrier signal from the dataset drops (usually because the user has hung up his terminal) a SIGHUP hangup signal is sent to the processes in the distinguished process group of the terminal; this usually causes them to terminate. The sending of SIGHUP does not take place if the CLOCAL bit is set in c_cflag field of the driver. Access to the terminal by other processes is then normally revoked, so any further reads will fail, and programs that read a terminal and test for End-of-File on their input will terminate appropriately.

Interrupt Characters

When the ISIG bit is set in the c_iflag word, there are several characters that generate signals in both canonical and noncanonical mode; all are sent to the processes in the foreground process group of the terminal. If the NOFLSH bit is not set in c_iflag, these characters also flush pending input and output when typed at a terminal. The characters shown here are the defaults; the symbolic names of the indices of these characters in the c_cc array of the termios structure are also shown. The characters may be changed.

^C  VINTR (in c_cc) generates a SIGINT signal. This is the normal way to stop a process which is no longer interesting, or to regain control in an interactive program.

\  VQUIT (in c_cc) generates a SIGQUIT signal. This is used to cause a program to terminate and produce a core image, if possible, in the file core in the current directory.

^Z  VSUSP (in c_cc) generates a SIGTSTP signal, which is used to suspend the current process group.

^Y  VDSUSP (in c_cc) generates a SIGTSTP signal as <Ctrl-Z> does, but the signal is sent when a program attempts to read the <Ctrl-Y>, rather than when it is typed.
Terminal Access Control

If a process attempts to read from its controlling terminal when the process is not in the foreground process group of the terminal, that background process group is sent a SIGTTIN signal. This signal normally causes the members of that process group to stop. If, however, the process is ignoring SIGTTIN, has SIGTTIN blocked, or if the reading process’ process group is orphaned, the read will return -1 and set `errno` to [EIO]. The operation will then not send a signal.

If a process attempts to write to its controlling terminal when the process is not in the foreground process group of the terminal, and the TOSTOP bit is set in the `c_lflag` word of the `termios` structure, that background process group is sent a SIGTTOU signal and the process is prohibited from writing. If TOSTOP is not set, or if TOSTOP is set and the process is blocking or ignoring the SIGTTOU signal, process writes to the terminal are allowed, and the SIGTTOU signal is not sent. If TOSTOP is set, if the writing process’ process group is orphaned, and if SIGTTOU is not blocked by the writing process, the write operation returns a -1 with `errno` set to [EIO], and does not a send a signal.

Terminal/Window Sizes

To accommodate terminals and workstations with variable-sized windows, the terminal driver provides a mechanism for obtaining and setting the current terminal size. The driver does not use this information internally, but only stores it and provides a uniform access mechanism. When the size is changed, a SIGWINCH signal is sent to the terminal’s process group so that knowledgeable programs may detect size changes.

tty Parameters

In contrast to earlier versions of the `tty` driver, the POSIX terminal parameters and structures are contained in a single structure, the `termios` structure defined in the `sys/termios.h` file. Refer to the `termios.h(0)` reference manual page for a complete summary of this file.

Basic ioctls Calls

A large number of `ioctl(2)` calls apply to terminals. Some have the general form:

```c
#include <sys/termios.h>

ioctl(fildes, code, arg)
struct termios *arg;
```
The applicable codes are:

**TIOCGETA**: Gets the termios structure and all its associated parameters. The interface delays until output is quiescent, then throws away any unread characters.

**TIOCSETA**: Sets the parameters according to the termios structure.

**TIOCSETAW**: Drains the output before setting the parameters according to the termios structure. Sets the parameters like TIOCSETA.

**TIOCSETAF**: Drains the output and flushes the input before setting the parameters according to the termios structure. Sets the parameters like TIOCSETA.

With the following codes *arg* is ignored:

**TIOCEXCL**: Set exclusive-use mode: no further opens are permitted until the file has been closed.

**TIOCNXCL**: Turn off exclusive-use mode.

With the following codes *arg* is a pointer to an int:

**TIOCFLUSH**: If the int pointed to by *arg* has a zero value, all characters waiting in input or output queues are flushed. Otherwise, the value of the int is for the FREAD and FWRITE bits defined in the *sys/file.h* file; if the FREAD bit is set, all characters waiting in input queues are flushed, and if the FWRITE bit is set, all characters waiting in output queues are flushed.

Setting and Unsetting Controlling Terminals

**TIOCSCTTY**: Sets the terminal as the controlling terminal for the calling process.

**TIOCNOTTY**: Voids the terminal as a controlling terminal.

The following are miscellaneous ioctl terminal commands. In cases where arguments are required, they are described; *arg* should otherwise be given as 0.

**TIOCSTI**: The argument points to a character which the system pretends had been typed on the terminal.

**TIOCSBRK**: The break bit is set in the terminal.

**TIOCCBRK**: The break bit is cleared.

**TIOCSDTR**: Data terminal ready is set.

**TIOCCDTR**: Data terminal ready is cleared.

**TIOCSTOP**: Output is stopped as if the "stop" character had been typed.

**TIOCSTART**: Output is restarted as if the "start" character had been typed.
TIOCSPGRP The arg parameter is a pointer to an int which is the value to which the process group ID for this terminal will be set.

TIOCOUTQ Returns in the int pointed to by arg the number of characters queued for output to the terminal.

TIOCREMOTE Sets the terminal for remote input editing.

FIONREAD returns in the int pointed to by arg the number of characters immediately readable from the argument descriptor. This works for files, pipes, and terminals.

Controlling Terminal Modems

The following ioctls apply to modems:

TIOCMODG The arg parameter is a pointer to an int, which is the value of the modem control state.

TIOCMODS The arg parameter is a pointer to an int, which is the value to which the modem control state is to be set.

TIOCMSET Sets all modem bits.

TIOCMBIS The arg parameter is a pointer to an int, which specifies the modem bits to be set.

TIOCMBIC arg is a pointer to an int, which specifies the modem bits to be cleared.

TIOCMGET Gets all the modem bits and returns them in the int pointed to by arg.

Window/Terminal Sizes

Each terminal has provision for storage of the current terminal or window size in a winsize structure, with format:

```c
struct winsize {
    unsigned short ws_row;    /* rows, in characters */
    unsigned short ws_col;    /* columns, in characters */
    unsigned short ws_xpixel; /* horizontal size, pixels */
    unsigned short ws_ypixel; /* vertical size, pixels */
};
```
tty(7)

A value of 0 (zero) in any field is interpreted as ‘‘undefined;’’ the entire structure is zeroed on final close.

The applicable ioctl functions are:

TIOCGWINSZ

The arg parameter is a pointer to a struct winsize into which will be placed the current terminal or window size information.

TIOCSWINSZ

The arg parameter is a pointer to a struct winsize, which will be used to set the current terminal or window size information. If the new information is different than the old information, a SIGWINCH signal will be sent to the terminal’s process group.

Files

/dev/tty Special file for tty.
/dev/tty* Special files for ttys, where the * (asterisk) sign represents the tty number.
/dev/console Device special file for console.

Related Information

Functions: ioctl(2), sigvec(2), tcgetattr(3), tcsetattr(3), tcdrain(3), tcflush(3), tcsendbreak(3), tcgetpgrp(3), tcsetpgrp(3)

Commands: csh(1), tset(1), getty(8)

IEEE Std POSIX 1003.1-1988
Application Environment Specification - Operating System/Programming Interfaces Volume
udp

Purpose  Internet user datagram protocol (UDP)

Synopsis  
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

Description  
UDP is a simple, unreliable datagram protocol that is used to support the
SOCK_DGRAM abstraction for the Internet Protocol family. UDP sockets are
connectionless, and are normally used with the sendto() and recvfrom() functions, though the connect() function may also be used to fix the destination for
future packets, in which case the recv() or read() and send() or write() functions
may be used.

UDP address formats are identical to those used by TCP. In particular, UDP
provides a port identifier in addition to the normal Internet address format. Note
that the UDP port space is separate from the TCP port space (that is, a UDP port
may not be connected to a TCP port). In addition, broadcast packets may be sent
(assuming the underlying network supports this) by using a reserved broadcast
address; this address is network interface dependent.

Options at the IP transport level may be used with UDP; see the ip() reference
page.

Errors  
If a socket operation fails, errno may be set to one of the following values:

[EISCONN]  The socket is already connected. This error occurs when trying to
establish connection on a socket or when trying to send a datagram
with the destination address specified.

[ENOTCONN]  The destination address of a datagram was not specified, and the
socket has not been connected.

[ENOBUFFS]  The system ran out of memory for an internal data structure.
An attempt was made to create a socket with a port that has already been allocated.

An attempt was made to create a socket with a network address for which no network interface exists.

Related Information

Functions: `getsockopt(2), recv(2), send(2), socket(2)`

Files: `netintro(7), inet(7), ip(7)`
This chapter contains reference pages for OSF/1 miscellaneous functions. The reference pages from the man5 directory are sorted alphabetically in this chapter.
Purpose

Octal, hexadecimal, and decimal ASCII character sets

Description

The octal character set is:

```
000 nul 001 soh 002 stx 003 etx 004 eot 005 enq 006 ack
007 bel 010 bs 011 ht 012 nl 013 vt 014 np 015 cr
016 so 017 si 020 dle 021 dc1 022 dc2 023 dc3 024 dc4
025 nak 026 syn 027 etb 030 can 031 em 032 sub 033 esc
034 fs 035 gs 036 rs 037 us 040 sp 041 ! 042 *
043 # 044 $ 045 % 046 & 047 ' 050 ( 051 )
052 + 053 , 054 - 056 . 057 / 060 0
061 1 062 2 063 3 064 4 065 5 066 6 067 7
070 8 071 9 072 : 073 ; 074 < 075 = 076 >
077 ? 100 @ 101 A 102 B 103 C 104 D 105 E
106 F 107 G 110 H 111 I 112 J 113 K 114 L
115 M 116 N 117 O 120 P 121 Q 122 R 123 S
124 T 125 U 126 V 127 W 130 X 131 Y 132 Z
133 [ 134 \ 135 ] 136 ^ 137 _ 140 ` 141 a
142 b 143 c 144 d 145 e 146 f 147 g 150 h
151 i 152 j 153 k 154 l 155 m 156 n 157 o
160 p 161 q 162 r 163 s 164 t 165 u 166 v
167 w 170 x 171 y 172 z 173 { 174 | 175 }
176 ~ 177 del
```

The hexadecimal character set is:

```
00 nul 01 soh 02 stx 03 etx 04 eot 05 enq 06 ack
07 bel 08 bs 09 ht 0a nl 0b vt 0c np 0d cr
0e so 0f si 10 dle 11 dc1 12 dc2 13 dc3 14 dc4
15 nak 16 syn 17 etb 18 can 19 em 1a sub 1b esc
1c fs 1d gs 1e rs 1f us 20 sp 21 ! 22 "
23 # 24 $ 25 % 26 & 27 ' 28 ( 29 )
2a * 2b + 2c , 2d - 2e . 2f / 30 0
31 1 32 2 33 3 34 4 35 5 36 6 37 7
3f ? 38 8 39 9 3a : 3b ; 3c < 3d =
3e > 40 @ 41 A 42 B 43 C 44 D 45 E
46 F 47 G 48 H 49 I 4a J 4b K 4c L
4d M 4e N 4f O 50 P 51 Q 52 R 53 S
54 T 55 U 56 V 57 W 58 X 59 Y 5a Z
```
The decimal character set is:

```
0 nul    1 soh    2 stx    3 etx    4 eot    5 enq    6 ack
 7 bel    8 bs     9 ht     10 nl     11 vt     12 np     13 cr
14 so    15 si    16 dle    17 dcl    18 dc2    19 dc3    20 dc4
21 nak    22 syn    23 etb    24 can    25 em     26 sub    27 esc
28 fs    29 gs     30 rs     31 us     32 sp     33 !     34 "
35 #      36 $      37 %      38 &      39 '      40 (      41 )
42 *      43 +      44 ,      45 -      46 .      47 /      48 0
49 1      50 2      51 3      52 4      53 5      54 6      55 7
56 8      57 9      58 :      59 ;      60 <      61 =      62 >
63 ?      64 @      65 A      66 B      67 C      68 D      69 E
70 F      71 G      72 H      73 I      74 J      75 K      76 L
77 M      78 N      79 O      80 P      81 Q      82 R      83 S
84 T      85 U      86 V      87 W      88 X      89 Y      90 Z
91 [      92 \      93 ]      94 ^      95 _      96 '      97 a
98 b      99 c     100 d     101 e     102 f     103 g     104 h
105 i     106 j     107 k     108 l     109 m     110 n     111 o
112 p     113 q     114 r     115 s     116 t     117 u     118 v
119 w     120 x     121 y     122 z     123 {     124 |     125 }
126 ~     127 del
```

Files

```
/usr/share/misc/ascii
```
end, etext, edata

Purpose  Defines the last location of a program

Synopsis
  extern end;
  extern etext;
  extern edata;

Description

The external names end, etext, and edata are defined for all programs. They are
not functions, but identifiers associated with the following addresses:
  etext  The first address following the program text.
  edata  The first address following the initialized data region.
  end   The first address following the data region that is not initialized.

The break value of the program is the first location beyond the data. When a
program begins running, this location coincides with end. However, many factors
can change the break value, including:
  • The brk( ) function
  • The malloc( ) function
  • The standard I/O functions
  • The -p flag on the cc command

Therefore, use sbrk(0), not end, to determine the break value of the program.

Related Information

Functions: brk(2), malloc(3)
Commands: cc(1)
environ

**Purpose**  
User environment

**Synopsis**  
extern char **environ ;

**Description**  
An array of strings called the environment is made available by the execve( ) function when a process begins. By convention these strings have the form name=value. The following names are used by various commands:

- **EXINIT**  
  A startup list of commands read by ex, edit, and vi.

- **HOME**  
  A user’s login directory, set by login from the password file passwd.

- **PATH**  
  The sequence of directories, separated by colons, searched by csh, sh, system, execvp, etc, when looking for an executable file. PATH is set to :/usr/ucb:/bin:/usr/bin initially by login.

- **PRINTER**  
  The name of the default printer to be used by lpr, lpq, and lprm.

- **SHELL**  
  The full pathname of the user’s login shell.

- **TERM**  
  The kind of terminal for which output is to be prepared. This information is used by commands, such as nroff or plot which may exploit special terminal capabilities. See /usr/share/misc/termcap for a list of terminal types.

- **TERMCAP**  
  The string describing the terminal in the TERM environment variable, or, if it begins with a / (slash), the name of the termcap file. See TERMPATH below.

- **TERMPATH**  
  A sequence of pathnames of termcap files, separated by colons or spaces, which are searched for terminal descriptions in the order listed. Having no TERMPATH is equivalent to a TERMPATH of $HOME/termcap:/etc/termcap. TERMPATH is ignored if TERMCAP contains a full pathname.

- **USER**  
  The login name of the user.

Further names may be placed in the environment by the export command and name=value arguments in sh, or by the setenv command if you use csh. It is unwise to change certain sh variables that are frequently exported by .profile files, such as MAIL, PS1, PS2, and IFS.
Related Information

Functions: exec(2), system(3)
Commands: csh(1), ex(1), login(1), sh(1)
Purpose  Layout of file systems

Description
This page describes the file system hierarchy. As a general rule, it lists only directories.

/ The root directory of the file system
/dev/ Block and character device files
/etc/ System configuration files and databases. These are nonexecutable files.
    nls/  National Language Support databases
/lost+found/ Files located by fsck
/net/ Mounted network directories
/opt/ Optional application packages
/sbin/ Commands essential for the system to boot. These commands do not depend on shared libraries or the loader and can have other versions in /usr/bin or /usr/sbin.
    init.d/ System state rc files
    rc0.d/ rc files executed for system-state 0
    rc2.d/ rc files executed for system-state 2
    rc3.d/ rc files executed for system-state 3
/stand/ Standalone programs
/tmp/ System generated temporary files (the contents of /tmp are usually not preserved across a system reboot)
/usr/ Contains the majority of user utilities and applications
    bin/ Common utilities and applications
    ccs/ C compilation system; tools and libraries used to generate C programs
        bin/ Development binaries (includes cc, ld, make, etc.)
        lib/ Development libraries and backends
        lex/ lex data
include/
Program header (include) files; not all subdirectories are listed below

mach/ Mach specific C include files
machine/
     Machine specific C include files
net/ Miscellaneous network C include files
netimp/
     C include files for IMP protocols
netinet/
     C include files for Internet standard protocols
netns/ C include files for XNS standard protocols
nfs/ C include files for Network File System (NFS)
protocols/
     C include files for Berkeley service protocols
rpc/ C include files for remote procedure calls
servers/
     C include files for servers
streams/
     C include files for Streams
sys/ System C include files (kernel data structures)
tli/ C include files for Transport Layer Interface
udp/ C include files for User Datagram Protocol
ufs/ C include files for UFS
lbin/ Back-end executables
spell/ Spell back-end
uucp/ UUCP programs
lib/ Consists entirely of links to libraries located elsewhere (/usr/ccs/lib, /usr/libin, /usr/share/lib, /X11/lib); included for compatibility
sbin/ System administration utilities and system utilities
Miscellaneous Functions

hier(5)

share/ Architecture-independent ASCII text files
dict/ Word lists
lib/
  me/ Macros for use with the ME macro package
  ms/ Macros for use with the MS macro package
tabset/
  Tab description files for a variety of terminals; used in /etc/termcap
terminfo/
  Terminal information database
tmac/ Text processing macros
man/ Online reference pages
  man1/ Source for user command reference pages
  man2/ Source for system call reference pages
  man3/ Source for library routine reference pages
  man4/ Source for file format reference pages
  man5/ Source for miscellaneous reference pages
  man7/ Source for device reference pages
  man8/ Source for administrator command reference pages
  cat1-8 Formatted versions of reference pages in the
        man1 through man8 directories
shlib/ Binary loadable shared libraries; shared versions of libraries
      in /usr/ccs/lib
/var/ Multipurpose log, temporary, transient, varying, and spool files
adm/ Common administrative files and databases
cron/ Files used by cron
  crash/ For saving kernel crash dumps
sendmail/
  sendmail configuration and database files
syslog/ Files generated by syslog
spool/  Miscellaneous printer and mail system spooling directories

lpd/   Line printer spooling directories

mail/  Incoming mail messages

mqueue/ Undelivered mail queue

uucp/  UUCP spool directory

tmp/   Application-generated temporary files that are kept between system reboots

run/   Files created when daemons are running

/vmunix Pure kernel executable (the operating system loaded into memory at boot time)

Related Information

Commands: ls(1), apropos(1), whatis(1), whereis(1), finger(1), which(1), find(1), grep(1), fsck(8)
Hostname resolution description

Hostnames are domains, where a domain is a hierarchical, dot-separated list of subdomains; for example, the machine monet, in the Berkeley subdomain of the EDU subdomain of the Internet would be represented as follows:

\texttt{monet.Berkeley.EDU}

Notice that there is no trailing dot.

Hostnames are often used with network client and server programs, which must generally translate the name to an address for use. (This function is generally performed by the \texttt{gethostbyname()} function.) Hostnames are resolved by the Internet name resolver in the following fashion.

If the name consists of a single component (that is, contains no dot), and if the \texttt{HOSTALIASES} environment variable is set to the name of a file, that file is searched for a string matching the input hostname. The file should consist of lines made up of two white-space separated strings, the first of which is the hostname alias, and the second of which is the complete hostname to be substituted for that alias. If a case-insensitive match is found between the hostname to be resolved and the first field of a line in the file, the substituted name is looked up with no further processing.

If the input name ends with a trailing dot, the trailing dot is removed, and the remaining name is looked up with no further processing.

If the input name does not end with a trailing dot, it is looked up by searching through a list of domains until a match is found. The default search list includes first the local domain, then its parent domains with at least 2 name components (longest first). For example, in the domain CS.Berkeley.EDU, the name lithium.CChem will be checked first as lithium.CChem.CS.Berkeley.EDU and then as lithium.CChem.Berkeley.EDU. Lithium.CChem.EDU will not be tried, as there is only one component remaining from the local domain. The search path can be changed from the default by a system-wide configuration file.

Related Information

Functions: \texttt{gethostbyname(3)}

Commands: \texttt{named(8)}
A

abort, 1-16, 1-30
abort function, 1-16
abs, 1-17
abs function, 1-17, 1-18
absolute value
  complex, 1-299
  function, 1-17
accept connect, 1-782
accept function, 1-19, 1-20
access
  changing for a file, 1-61
  file, 1-21
access function, 1-21, 1-22
access modes
  changing for a mapped file, 1-406
  changing for a shared memory region, 1-406
  retrieving and setting for a file, 1-155
accounting
  enabling and disabling, 1-23
  process, 1-23
accounting record, expanding, 1-151
acct function, 1-23, 1-24
acos function, 1-739, 1-741
acosh function, 1-29
adjtime function, 1-25, 1-26
advance function, 1-601, 1-605
alarm function, 1-27, 1-28
alloca function, 1-364, 1-367
allocate memory, 1-786
alphasort function, 1-630, 1-631
any function, 1-342, 1-344
anysr function, 1-342, 1-344
ar, archive library file format, 2-2
arc cosine, hyperbolic function, 1-29
archive library file format, 2-2
arc sine, hyperbolic function, 1-29
arc tangent, hyperbolic function, 1-29
argument vector, returning flag letters from, 1-244
asctime function, 1-83, 1-88
asctime_r function, 1-83, 1-88
asin function, 1-739, 1-741
asinh function, 1-29
assert macro, 1-30, 1-31
assigning buffers, 1-660
async_daemon function, 1-32
asynchronous server, creating in NFS, 1-32
atan function, 1-739, 1-741
atan2 function, 1-739, 1-741
atanh function, 1-29
atexit function, 1-145, 1-147
atof function, 1-33, 1-34
atoi function, 1-35, 1-38
atol function, 1-35, 1-38
attributes object, creating for threads, 1-510
authenticating clients for servers, 1-628

B
balbrk function, 1-342, 1-344
baud rate
  returning input from termios, 1-55
  returning output from termios, 1-56
  setting input in termios, 1-57
  setting output in termios, 1-58
bcmp function, 1-39, 1-40
bcopy function, 1-39, 1-40
bessel functions, 1-41, 1-42
binary search function, 1-47
binary search trees, managing, 1-893
bind, socket name, 1-43
bind address, 1-790
bind function, 1-43, 1-44
bit strings, functions, 1-39
blocking signals, 1-718
break, changing data segment size, 1-45
breaking data transmission, 1-878
brk function, 1-45, 1-46
bsearch function, 1-47, 1-48
buffer, assigning, 1-660
byte quantities
  long, 1-559
  short, 1-562
bytes, swapping, 1-767
byte stream
  retrieving long quantities from, 1-237
  retrieving short quantities from, 1-275
byte strings, functions, 1-39
bzero function, 1-39, 1-40

C
cabs function, 1-299, 1-300
calloc function, 1-364, 1-367
cancelability of threads, 1-542, 1-545
cancellation points in threads, 1-549
capabilities of terminals, 2-127
case conversion, 2-4
catclose function, 1-49, 1-50
cat function, 1-342, 1-344
catgets function, 1-51, 1-52
catopen function, 1-53, 1-54
cbrt function, 1-750, 1-751
ceil function, 1-168, 1-170
cfgetispeed function, 1-55
cfgetospeed function, 1-56
cfsetispeed function, 1-57
cfsetospeed function, 1-58
character
classification functions, 1-89, 1-313
converting multibyte to wide, 1-372
finding length of multibyte, 1-368
going from input stream, 1-292, 1-912
pushing back, 1-906
translating to 7-bit ASCII, 1-78
translating to lowercase, 1-78
translating to uppercase, 1-78
writing to output stream, 1-913
characteristics of file implementation, 1-458
characters
classification, 2-4
writing out, 1-555
character string
converting multibyte to wide, 1-370
converting to floating point, 1-33
converting to integer, 1-35
character strings, wide, operations on, 1-941
character translation functions, 1-78, 1-80
chdir function, 1-59, 1-60
child process
creating via fork, 1-176
waiting for it to stop or terminate, 1-923
chmod function, 1-61, 1-64
chown function, 1-65, 1-67
chroot function, 1-68, 1-69
clean_up function, 1-342, 1-344
cleanup stack
adding routines, 1-496
removing a routine from, 1-494
clearenv function, 1-70
clearerr function, 1-71
clients, authenticating for servers, 1-628
clock, 1-756
getting time, 1-884
setting value, 1-662
clock function, 1-72
closedir function, 1-453, 1-457
close endpoint, 1-795
close function, 1-73, 1-74
closelog function, 1-776, 1-779
closing a pipe, 1-464
collating sequence, 2-4
commands
executing, 1-780
executing on remote host, 1-580, 1-620
comparing thread identifiers, 1-518
compatibility
with old UNIX systems, 1-734
with other UNIX systems, 1-710
compatibility interfaces for signals, 1-726
compile function, 1-601, 1-605
complementary error function, computing, 1-134
configuring system variables, 1-774
connect, 1-797
connect function, 1-75, 1-77
connection
accepting on a socket, 1-19
establishing between two sockets, 1-75
listening for on a socket, 1-347
protocol, 1-808
constructing a name for a temporary file, 1-888
context, execution, saving and restoring, 1-670
control, flow of, 1-870
controlling terminal, generating pathname for, 1-81
control operations, on a file, 1-155
convention tables for locale, 2-15
converting a wide character, 1-930
converting dates and times, 1-757
converting formatted input, 1-632, 1-939
converting wide characters, 1-928
core memory image, 2-3
cos function, 1-739, 1-741
cosh function, 1-742, 1-743
CPU time, returning, 1-72
creat function, 1-447, 1-452
creating a temporary file, 1-887
creating a thread, 1-514
creating keys for threads, 1-524
creating mutexes for threads, 1-528
creating signal masks, 1-711
ctab command, 2-4, 2-8
ctermid function, 1-81, 1-82
ctime function, 1-83, 1-88
ctime_r function, 1-83, 1-88
ctype functions, 1-89, 1-91
cube root function, 1-750
curdir function, 1-342, 1-344
current directory, changing, 1-59
curses
curses routines, 1-93
minicurses package, 1-92
screen dimensions, 1-92
termcap compatibility functions, 1-106
terminfo level functions, 1-104
curses library, 1-92, 1-106
cuserid function, 1-107, 1-108

database
manipulating entry in user, 1-259
user, 1-219
database management, dbm library, 1-109
databases
disktab, 2-12
group, 2-24
protocols, 2-98
ROUTE, 2-104
services for Internet, 2-109
shell, 2-110
terminfo, 2-127

data segment, changing size for
break, 1-45
data sink, 2-67
data, thread_specific, binding
values to keys, 1-547
date and time, returning, 1-285
date conversion, 1-757
dbm_clearerr function, 1-434,
1-436
dbm_close function, 1-434, 1-436
dbm_delete, function, 1-434, 1-436
dbm_error function, 1-434, 1-436
dbm_fetch function, 1-434, 1-436
dbm_firstkey function, 1-434,
1-436
dbm_forder function, 1-434, 1-436
dbm_init function, 1-109, 1-110
dbm_nextkey function, 1-434,
1-436
dbm_open function, 1-434, 1-436
dbm_store function, 1-434, 1-436
deallocate memory, 1-805
delete function, 1-109, 1-110
deleting attribute object from
threads, 1-512
deleting mutexes from threads,
1-526
descriptors, file, 2-20
descriptor table, returning size of,
1-210
detaching a thread, 1-516
device
adding swap device for
interleaved paging
and swapping, 1-768
allocating paging and
swapping space,
1-768
device file, control operations on,
1-310
devices, null, 2-67
diagnostics, inserting in programs,
1-30
difftime function, 1-83, 1-88
dir, format of directories, 2-9
directories
scanning, 1-630
sorting, 1-630
directory
changing current, 1-59
changing root, 1-68
creating, 1-378, 1-383
effective root, 1-68
mounting a filesystem on,
1-395, 1-400
removing, 1-623
removing entry of, 1-908
renaming, 1-610
returning entries in file-
system independent
format, 1-207
returning pathname for
current, 1-293
returning pathname of
current, 1-205
umounting a filesystem
from, 1-902
walking a file tree, 1-192
directory operations, 1-453
discon endpoint, 1-838
disconnect, 1-838, 1-858
disconnect, 1-855
disk, getting description of, 1-209
disklabel, 2-10, 2-11
disk packe label, 2-10
disk quotas
   enabling and disabling, 1-680
   manipulating, 1-570
disktab database, 2-14
disktab database, 2-12
div function, 1-17, 1-18
division function, 1-17
dname function, 1-342, 1-344
dn_comp function, 1-113, 1-114
dn_expand function, 1-115, 1-116
dn_find function, 1-117, 1-118
dn_skipname function, 1-119, 1-120
domain name, 2-102
   compressing, 1-113
   expanding, 1-115
   searching for default, 1-613
   searching for expanded, 1-117
   skipping over compressed, 1-119
drand48 function, 1-121, 1-124
drivers
   for terminals, 2-151
   pseudo terminal, 2-99
dup function, 1-155, 1-160
dup2 function, 1-155, 1-160

edata identifier, 3-4
edata identifier, 3-4
en, locale convention tables, 2-15, 2-17
encode_mach_o_hdr function, 1-128, 1-129
endsent function, 1-214, 1-215
endgrent function, 1-219, 1-221
endhostent function, 1-130
_end identifier, 3-4
end identifier, 3-4
endnetent function, 1-131
endpoint
   close, 1-795
   discon, 1-838
   disconnect, 1-855
   establish, 1-822
   event, 1-818
endprotoent function, 1-132
endpwent function, 1-259, 1-261
endservent function, 1-133
endusershell function, 1-288
endutent function, 1-289, 1-291
environment variable, setting of, 1-558
environment file, 2-15
environment variable, returning value of, 1-211
erand48 function, 1-121, 1-124
erfc function, 1-134, 1-135
erf function, 1-134, 1-135
error, 1-803
error function, computing, 1-134
_etext identifier, 3-4
etext identifier, 3-4
Euclidean distance function, 1-299
event, look, 1-818

e cvt function, 1-125, 1-127

Index-6
exec function, 1-136, 1-141
execle function, 1-136, 1-141
execl function, 1-136, 1-141
execlp function, 1-136, 1-141
executing commands on remote host, 1-580, 1-620
executing shell commands, 1-780
execution
starting and stopping profiling, 1-483
suspending, 1-914
execution context, saving and restoring, 1-670
execution of a process, suspending, 1-744
execve function, 1-136, 1-141
execv function, 1-136, 1-141
execvp function, 1-136, 1-141
exec_with_loader function, 1-142, 1-144
_exit, 1-16
_exit function, 1-145, 1-147
exit function, 1-145, 1-147
expacct function, 1-151
exp function, 1-148, 1-150
exponential function, 1-148
exports file, 2-18, 2-19
expressions, regular, 1-582, 1-601
external variable, optarg, 1-244

f
fchdir function, 1-59, 1-60
fchmod function, 1-61, 1-64
fchown function, 1-65, 1-67
fclose function, 1-152, 1-154
fcntl function, 1-155, 1-160
fcvt function, 1-125, 1-127
FD_CLR macro, 1-638, 1-641
fd file descriptor, 2-20
fdfopen function, 1-342, 1-344
FD_ISSET macro, 1-638, 1-641
dopen function, 1-171, 1-175
FD_SET macro, 1-638, 1-641
FD_ZERO macro, 1-638, 1-641
feof macro, 1-161
ferror macro, 1-162
fetch macro, 1-109, 1-110
fflush function, 1-152, 1-154
fis function, 1-39, 1-40
fgetc function, 1-201, 1-202
fgetpos function, 1-184, 1-187
fgets function, 1-267, 1-268
fgetwc function, 1-292
fgetws function, 1-294
FIFO, creating, 1-381, 1-383
file
access, 1-61, 1-915
access flags, 1-447
access modes, 1-155
advisory lock, 1-164
changing access, 1-61
changing length of, 1-890
changing owner and group IDs, 1-65
checking I/O status of file objects, 1-638
closing, 1-73
controlling a device file, 1-310
controlling locking on file sections, 1-355

Index-7
control operations, 1-155
creating, 1-383, 1-447
creating a directory, 1-378, 1-383
creating a FIFO, 1-383
creating a link for, 1-345
creating a pipe, 1-467
creating a special file, 1-383
creation mask, 1-901
determining accessibility, 1-21
device file control, 1-310
executable, 1-136
executable with loader, 1-142
executing, 1-136
executing with loader, 1-142
locking, 1-355
locks, 1-155
making symbolic links, 1-770
mapping a file system object into virtual memory, 1-390
modification time, 1-915
moving read-write offset, 1-360
opening and positioning on first record, 1-214
opening for reading or writing, 1-447
owner and group IDs, 1-65
polling, 1-471
providing information about, 1-752
providing information about an open file, 1-752
providing information about, including links, 1-752
reading from, 1-584
reading from a symbolic link, 1-588
reading next line of, 1-214
removing, 1-608
removing a directory, 1-623
renaming, 1-610
retrieving and setting access modes, 1-155
retrieving and setting locks, 1-155
retrieving and setting status information, 1-155
returning the handle for, 1-212
searching for file system type, 1-214
searching for special filename, 1-214
searching for system filename, 1-214
setting access and modification times, 1-915
setting and getting creation mask value, 1-901
setting or removing a lock, 1-164
shared library requirement, 1-142
status flags, 1-447
system statistics, 1-754
writing changes to disk, 1-188
writing to, 1-932
file descriptor
checking I/O status of, 1-638
closing, 1-73
monitoring conditions on multiple, 1-471
sets for checking I/O status, 1-638
file descriptors, 2-20
file handle, returning, 1-212
file implementatation,
 characteristics of, 1-458
file locking, 1-164
filename, constructing unique, 1-386
file, network, opening and rewinding, 1-676
fileno macro, 1-163
file, protocols, setting and rewinding, 1-679
files
archive library, 2-2
core memory image, 2-3
directory format, 2-9
exports, 2-18
/fB/dev/tty/fR, 2-151
file system volume, 2-21
name for temporary, 1-888
password, 2-96
resolver configuration, 2-102
shells, 2-110
signal, 2-113
stab, 2-120
temporary, 1-887
terminfo, 2-127
termios, 2-139
utmp, 1-898
filesystem
enabling and disabling disk quotas, 1-680
mapping an object into virtual memory, 1-390
mounted, 1-216
mounting, 1-395, 1-400
renaming files and directories, 1-610
returning list of all mounted, 1-216
umounting, 1-395, 1-902
file system
information about mounted, 1-754
manipulating disk quotas, 1-570
returning information about, 1-214
updating, 1-773
file-system independent format, returning directory entries in, 1-207
firstkey function, 1-109, 1-110
flag letters, returning from argument vector, 1-244
floating-point integer
absolute value function, 1-168
modulo remainder function, 1-168
round functions, 1-168
floating-point number
converting to a string, 1-125
converting to fraction and integral power of 2, 1-181
converting to integral and fractional parts, 1-181
multiplying by integral power of 2, 1-181
flockfile function, 1-167
flock function, 1-164, 1-166
floor function, 1-168, 1-170
flow control functions, 1-870
flushing input data, 1-872
flushing output data, 1-872
fmin function, 1-402, 1-405
fmod function, 1-168, 1-170
fmout function, 1-402, 1-405
fopen function, 1-168, 1-171
forder function, 1-109, 1-110
foreground process, group ID, 1-876
fork function, 1-176, 1-178
format of directories, 2-9
format of file system volume, 2-21
formatted input, converting, 1-632, 1-939
formatting output, 1-476
formatting output parameters, 1-921
formatting printed output, 1-937
fpathconf function, 1-458, 1-461
fprintf function, 1-476, 1-482
fputc function, 1-555, 1-557, 1-564, 1-565
fputs function, 1-560, 1-561
fputc function, 1-555, 1-557, 1-564, 1-565
fputws function, 1-566, 1-567
fread function, 1-179, 1-180
free function, 1-364, 1-367
freeing process timers, 1-625
freopen function, 1-171, 1-175
frexp function, 1-181, 1-183
fs, file system volume, 2-21, 2-23
fscanf function, 1-632, 1-637
fseek function, 1-184, 1-187
fsetpos function, 1-184, 1-187
fstatfs function, 1-754, 1-755
fstat function, 1-752, 1-753
fsync function, 1-188, 1-189
ftell function, 1-184, 1-187
ftime function, 1-283, 1-284
ftok function, 1-190, 1-191
ftruncate function, 1-890, 1-892
ftw function, 1-192, 1-194
function, 1-219
function errors, 1-466
functions, interrupting with signals, 1-714
funlockfile function, 1-195
fwrite function, 1-179, 1-180

G

gamma function, 1-196, 1-197
gcd function, 1-402, 1-405
gcvt function, 1-125, 1-127
generating random numbers, 1-574, 1-577
geometry of disks, 2-10, 2-12
getaddressconf function, 1-198, 1-200
getchar macro, 1-201, 1-202
getclock function, 1-203, 1-204
getc macro, 1-201, 1-202
getcwd function, 1-205, 1-206
getdirentries function, 1-207, 1-208
getdiskbyname function, 1-209
getdtablesize function, 1-210
getegid function, 1-218
getenv function, 1-211
geteuid function, 1-287
getfh function, 1-212, 1-213
getfsent function, 1-214, 1-215
getfsfile function, 1-214, 1-215
getfsspec function, 1-214, 1-215
getfsstat function, 1-216, 1-217
getfstype function, 1-214, 1-215
getgid function, 1-218
getgrent function, 1-219, 1-221
getgrgid function, 1-219, 1-221
getgrnam function, 1-219, 1-221
getgroups function, 1-222, 1-223
gethostbyaddr function, 1-224, 1-225
gethostbyname function, 1-226, 1-227
gethostent function, 1-228, 1-229
gethostid function, 1-230
gethostname function, 1-231
getitimer function, 1-232, 1-234
getlogin function, 1-235, 1-236
getlogin_r function, 1-235, 1-236
_getlong function, 1-237, 1-238
genetbyaddr function, 1-239, 1-240
gnetbyname function, 1-241, 1-242
genetent function, 1-243
getopt function, 1-244, 1-245
getpagesize function, 1-246
getpass function, 1-247, 1-248
getpeername function, 1-249, 1-250
getpgid function, 1-251
getpid function, 1-251
getppid function, 1-251
getpriority function, 1-252, 1-253
getprotobynumber function, 1-256, 1-257
getprotobyname function, 1-254, 1-255
getprotoent function, 1-258
getpwent function, 1-259, 1-261
getpwnam function, 1-259, 1-261
getpwuid function, 1-259, 1-261
getrlimit function, 1-262, 1-264
getrusage function, 1-265, 1-266
getservbyname function, 1-269, 1-270
getservbyport function, 1-271, 1-272
getservent function, 1-273, 1-274
gets function, 1-267, 1-268
_getshort function, 1-275, 1-276
getsockname function, 1-277, 1-278
getsockopt function, 1-279, 1-282
getstate, 1-812
gmtimeofday function, 1-283, 1-284
gettimeofday function, 1-283, 1-284
getting service file entries, 1-688
getting user limits, 1-899
getuid function, 1-287
getusershell function, 1-288
getutent function, 1-289, 1-291
getutid function, 1-289, 1-291
getutline function, 1-289, 1-291
getwc function, 1-292
getwchar function, 1-292
getwd function, 1-293
getw function, 1-201, 1-202
getws function, 1-294
giveup function, 1-342, 1-344
gmtime function, 1-83, 1-88
gmtime_r function, 1-83, 1-88
group access list, setting, 1-666

group database, 2-24

group ID
  changing for a file, 1-65
  foreground process, 1-876
  real and effective, 1-682
  real, effective, and saved set, 1-664
  returning effective, 1-218
  returning for a process, 1-251
  returning real, 1-218
  setting, 1-664, 1-684, 1-689
  setting for process, 1-677, 1-882
  setting real and effective, 1-682

group information, accessing in user database, 1-219

group set
  initializing concurrent, 1-307
  returning for current process, 1-222

H

hash tables
  creating, 1-295
  deleting, 1-295
  searching, 1-295
  hcreate function, 1-295, 1-296

hdestroy function, 1-295, 1-296

host
  returning ID for current, 1-230
  returning name of current, 1-231
  setting ID for current, 1-668
  setting name of current, 1-669

host address, converting to byte-ordered address integer, 1-303

host-byte order
  converting long integer, 1-297, 1-445
  converting short integer, 1-298, 1-446

host entries, ending retrieval of, 1-130

host entry
  returning by address, 1-224
  returning by name, 1-226

host ID
  returning for current host, 1-230
  setting for current host, 1-668

hostname
  returning for current host, 1-231
  setting for current host, 1-669

hosts file
  opening, 1-228
  reading next line, 1-228
  resetting file marker, 1-228
  retrieving entries, 1-228

hosts name files
  searching by address, 1-224
searching by name, 1-226
hsearch function, 1-295, 1-296
htonl function, 1-297
htons function, 1-298
hyperbolic functions, 1-742
acosh, 1-29
asinh, 1-29
atanh, 1-29
hypot function, 1-299
hypot function, 1-299, 1-300

I

icmp, Internet Control Message Protocol, 2-25, 2-26
idp, Xerox Internet Protocol, 2-27, 2-29
IDs of threads, 1-541
imatch function, 1-342, 1-344
index function, 1-342, 1-344
Inet, Internet Protocol family, 2-30, 2-31
inet_addr function, 1-301
inet_lnaof function, 1-302
inet_makeaddr function, 1-303
inet_netof function, 1-304
inet_network function, 1-305
inet_ntoa function, 1-306
initgroups function, 1-307, 1-308
initializing routine for threads, 1-539
initstate function, 1-577, 1-579
inodes, 2-21
input
converting, 1-939
converting formatted, 1-632
flushing, 1-872
pushing back character, 1-906
input stream
getting character from, 1-201, 1-292
getting characters from, 1-912
getting word from, 1-201, 1-292
insque function, 1-309
integer arithmetic functions, 1-402, 1-405
integers
absolute value, 1-17
division, 1-17
interface
to the sigaction function, 1-734
to the sigprocmask function, 1-710
interface for terminals, 2-139
interfaces
loopback, 2-34
LVM, 2-35
interfaces for terminals, 2-151
interfaces to networks, 2-66
Internet
domain name, 2-102
protocols database, 2-98
services available, 2-109
Internet address, searching for, 1-613
Internet Control Message Protocol, 2-25
Internet ports, 1-626
Internet Protocol, 2-30, 2-32
Internet Protocol family, 2-165
interprocess communication key, generating, 1-190
interrupting functions with signals, 1-714
interval timers
  changing timeout, 1-27
  setting and returning, 1-232
  setting timeout, 1-27
introduction to networking, 2-58
inverse trigonometric functions, 1-739
invert function, 1-402, 1-405
ioctl function, 1-310, 1-311
IO functions, standard, 1-555, 1-560, 1-564
I/O status, checking file descriptor sets for, 1-638
ip, Internet Protocol, 2-32, 2-33
isalnum function, 1-89, 1-91
isalpha function, 1-89, 1-91
isascii function, 1-89, 1-91
isatty function, 1-896, 1-897
isctrnl function, 1-89, 1-91
isdigit function, 1-89, 1-91
isgraph function, 1-89, 1-91
isjalnum function, 1-313, 1-314
isjalpha function, 1-313, 1-314
isjdigit function, 1-313, 1-314
isjpunct function, 1-313, 1-314
isspace function, 1-33, 1-36, 1-89, 1-91
isupper function, 1-89, 1-91
isxdigit function, 1-89, 1-91
itom function, 1-402, 1-405

J

Japanese Language Support, 1-292
j0 function, 1-41, 1-42
j1 function, 1-41, 1-42
jn function, 1-41, 1-42
jrand48 function, 1-121, 1-124
jump point, setting, 1-729

K

kernel packet forwarding, database, 2-104
kill function, 1-315, 1-316
killpg function, 1-316

L

labels for disk packs, 2-10
labs function, 1-17, 1-18
lcg48 function, 1-121, 1-124
ldexp function, 1-181, 1-183
ldiv function, 1-17, 1-18
ldr_entry function, 1-317
ldr_inq_module function, 1-318, 1-319
ldr_inq_region function, 1-320, 1-321
ldr_install function, 1-322, 1-323
ldr_lookup_package function, 1-324, 1-325
ldr_next_module function, 1-326, 1-327
ldr_remove function, 1-328
ldr_xattach function, 1-329, 1-330
ldr_xdetach function, 1-331, 1-332
ldr_xentry function, 1-333, 1-334
ldr_xload function, 1-335, 1-337
ldr_xlookup_package function, 1-338, 1-339
ldr_xunload function, 1-340, 1-341
lfind function, 1-358, 1-359
lgamma function, 1-196, 1-197
libPW, 1-342
limits, for users, 1-899
linear search, of table, 1-358
link
  creating, 1-345
  decrementing count, 1-908
  making symbolic link to a file, 1-770
  providing information about symbolic links, 1-752
  reading from symbolic, 1-588
  removing directory entry, 1-908
link function, 1-345, 1-346
listen, 1-814
listen function, 1-347, 1-348
loaded module
  returning entry point for, 1-317
returning entry point for in another process, 1-333
returning information about, 1-318
returning next for a process, 1-326
returning region information for, 1-320
unloading in another process, 1-340
loader
  attaching to another process, 1-329
defined external names for program locations, 3-4
detaching from an attached process, 1-331
executing a file with, 1-142
installing module, 1-322
loading module, 1-349
loading module in another process, 1-335
returning address of symbol name in another process package, 1-338
returning address of symbol name in a package, 1-324
returning a module from process package table, 1-328
returning entry point for loaded module, 1-317
returning entry point for loaded module in another process, 1-333
returning information about loaded module, 1-318
returning next module ID for a process, 1-326
returning region information for loaded module, 1-320
unloading a module, 1-910
unloading module in another process, 1-340
loader module
installing, 1-322
loading, 1-349
loading in another process, 1-335
removing from process package table, 1-328
load function, 1-349, 1-350
locale
convention tables, 2-15
setting and querying, 1-672
localeconv function, 1-351, 1-354
localeconv_r function, 1-351, 1-354
locale-dependent parameters, 1-351
localtime function, 1-83, 1-88
localtime_r function, 1-83, 1-88
lock
advisory on a file, 1-164
setting or removing on a file, 1-164
lockf function, 1-355, 1-357
locking mutexes for threads, 1-530, 1-532
lockit function, 1-342, 1-344
locks
enforced versus arbitrary, 1-355
on process’ text and/or data segments in memory, 1-469
on sections of an open file, 1-355
read versus write, 1-355
shared and exclusive on a file, 1-155
logarithm functions, 1-148
log function, 1-148, 1-150
log10 function, 1-148, 1-150
Logical Volume Manager, 2-35
login name, returning and setting, 1-235
logname function, 1-342, 1-344
lo interface, 2-34
long byte quantities, placing in byte stream, 1-559
long integer
converting to host-byte order, 1-445
converting to network-byte order, 1-297
longjmp function, 1-670, 1-671
loopback network interface, 2-34
lrand48 function, 1-121, 1-124
lsearch function, 1-358, 1-359
lseek function, 1-360, 1-361
lstat function, 1-752, 1-753
lvm, Logical Volume Manager, 2-35, 2-55
LVM_ACTIVATEVM command, 2-36
LVM_ATTACHPV command, 2-37
LVM_CHANGELEV command, 2-37
LVM_CHANGEPV command, 2-37
Index-16
LVM_CREATELV command, 2-38
LVM_CREATEVG command, 2-38
LVM_DEACTIVATEVG command, 2-39
LVM_DELETELV command, 2-39
LVM_DELETEEPV command, 2-39
LVM_EXTENDLV command, 2-39
LVM_INSTALLPV command, 2-40
LVM_OPTIONGET command, 2-40
LVM_QUERYLV command, 2-41
LVM_QUERYLVMAP command, 2-41
LVM_QUERYPV command, 2-42
LVM_QUERYPVMAP command, 2-42
LVM_QUERYPVSPATH command, 2-43
LVM_QUERYPVSS command, 2-43
LVM_QUERYVG command, 2-43
LVM_REALLOCLV command, 2-44
LVM_REDUCELV command, 2-44
LVM_REMOVEPV command, 2-44
LVM_RESYNCLV command, 2-45
LVM_RESYNCLX command, 2-45
LVM_RESYNCPV command, 2-45
LVM_SETVGID command, 2-45
madvise function, 1-362, 1-363
mallinfo function, 1-364, 1-367
malloc function, 1-364, 1-367
mallopt function, 1-364, 1-367
management, 1-827
managing binary search trees, 1-893
managing signals, 1-726
manipulating strings, 1-760
mapped file
changing access modes, 1-406
initializing semaphore in, 1-409
synchronizing, 1-428
unmapping, 1-430
writing changes to disk, 1-428
mask, setting and getting value of for file, 1-901
mathematical functions, 1-739, 1-742
mblen function, 1-368, 1-369
mbstowcs function, 1-370, 1-371
mbtowc function, 1-372, 1-373
mcmp function, 1-402, 1-405
mdiv function, 1-402, 1-405
memccpy function, 1-374, 1-377
memchr function, 1-374, 1-377
memcmp function, 1-374, 1-377
memcpy function, 1-374, 1-377
memmove function, 1-374, 1-377
memory, 1-786
allocating, 1-364
allocating space for an array, 1-364
changing size of allocated, 1-364
free, 1-805
freeing, 1-364
  tuning allocation algorithm, 1-364
memory allocator functions, 1-364, 1-367
memory area, manipulating strings in, 1-374
memory image, 2-3
memory operations, 1-374, 1-377
memory region, checking validity of, 1-432
memset function, 1-374, 1-377
message
  receiving from a message queue, 1-422
  retrieving from message catalog, 1-51
  sending to a message queue, 1-425
message catalog
  closing, 1-49
  opening, 1-53
  retrieving a message from, 1-51
message queue
  creating, 1-420
  performing control operations on, 1-417
  receiving a message from, 1-422
  removing, 1-417
  returning the ID for, 1-420
  sending a message to, 1-425
messages
  for function errors, 1-466
  receiving from connected or unconnected sockets, 1-598
  receiving from connected sockets, 1-593
  receiving from unconnected sockets, 1-595
  sending messages using a message structure, 1-655
  sending through connected sockets, 1-653
  sending through unconnected sockets, 1-657
m_in function, 1-402, 1-405
min function, 1-402, 1-405
mkdir function, 1-378, 1-380
mkfifo function, 1-381, 1-382
mknod function, 1-383, 1-385
mkstemp function, 1-386, 1-387
mktemp function, 1-386, 1-387
mktime function, 1-83, 1-88
mktimer function, 1-388, 1-389
mmap function, 1-390, 1-394
modf function, 1-181, 1-183
module, unloading, 1-910
mount function, 1-395, 1-399, 1-400, 1-401
mount points, remote', 2-18
m_out function, 1-402, 1-405
mout function, 1-402, 1-405
move function, 1-342, 1-344, 1-402, 1-405
mprotect function, 1-406, 1-408
mrand48 function, 1-121, 1-124
msem_init function, 1-409, 1-410
msem_lock function, 1-411, 1-412
msem_remove function, 1-413, 1-414
msem_unlock function, 1-415, 1-416
msgctl function, 1-417, 1-419
msgget function, 1-420, 1-421
msgrecv function, 1-422, 1-424
msgsnd function, 1-425, 1-427
msqid_ds, 2-56
msgrt function, 1-402, 1-405
msub function, 1-402, 1-405
msync function, 1-428, 1-429
mult function, 1-402, 1-405
multibyte character, converting from wide, 1-930
multibyte character string, converting from wide, 1-928
munmap function, 1-430, 1-431
mutex attribute object
creating, 1-536
deleting, 1-538
mvalid function, 1-432, 1-433

N

name, terminal, 1-896
name servers
querying, 1-618
query messages for, 1-615
NaN, checking, 1-312
national language, returning
information about, 1-441
ndbm library, 1-434, 1-436
neg function, 1-437
netintro, 2-58, 2-63
network address
converting dot-formatted string to integer, 1-305
converting integer form to host (local) address, 1-302

network-byte order
converting long integer, 1-297, 1-445
converting short integer, 1-298, 1-446

network entry
returning by address, 1-239
returning by name, 1-241
network file, opening and rewinding, 1-676
networking
getstate, 1-812
introduction to, 2-58
rcvrel, 1-842
sndrel, 1-858
sync, 1-863
t_accept, 1-782
t_alloc, 1-786
t_bind, 1-790
t_close, 1-795
t_connect, 1-797
t_error, 1-803
t_free, 1-805
t_getinfo, 1-808
t_list, 1-814
t_look, 1-818
t_open, 1-822
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_optmgmt</td>
<td>1-827</td>
</tr>
<tr>
<td>t_rcvconnect</td>
<td>1-834</td>
</tr>
<tr>
<td>t_rcvdis</td>
<td>1-838</td>
</tr>
<tr>
<td>t_rcvudata</td>
<td>1-844</td>
</tr>
<tr>
<td>t_rcvuderr</td>
<td>1-848</td>
</tr>
<tr>
<td>t_snd</td>
<td>1-851</td>
</tr>
<tr>
<td>t_snddis</td>
<td>1-855</td>
</tr>
<tr>
<td>t_sndudata</td>
<td>1-860</td>
</tr>
<tr>
<td>t_unbind</td>
<td>1-866</td>
</tr>
<tr>
<td>networking</td>
<td>1-831</td>
</tr>
<tr>
<td>loopback interface</td>
<td>2-34</td>
</tr>
<tr>
<td>software interface</td>
<td>2-66</td>
</tr>
<tr>
<td>networks</td>
<td></td>
</tr>
<tr>
<td>networking</td>
<td></td>
</tr>
<tr>
<td>loopback interface</td>
<td>2-34</td>
</tr>
<tr>
<td>software interface</td>
<td>2-66</td>
</tr>
<tr>
<td>networks</td>
<td></td>
</tr>
<tr>
<td>closing</td>
<td>1-131</td>
</tr>
<tr>
<td>opening</td>
<td>1-243</td>
</tr>
<tr>
<td>reading next line</td>
<td>1-243</td>
</tr>
<tr>
<td>searching by address</td>
<td>1-239</td>
</tr>
<tr>
<td>searching by name</td>
<td>1-241</td>
</tr>
<tr>
<td>nextkey function</td>
<td>1-109</td>
</tr>
<tr>
<td>nextkey function</td>
<td>1-110</td>
</tr>
<tr>
<td>networks</td>
<td></td>
</tr>
<tr>
<td>closed</td>
<td></td>
</tr>
<tr>
<td>opening</td>
<td></td>
</tr>
<tr>
<td>reading next line</td>
<td></td>
</tr>
<tr>
<td>searching by address</td>
<td></td>
</tr>
<tr>
<td>searching by name</td>
<td></td>
</tr>
</tbody>
</table>

**networks file**

- closing, 1-131
- opening, 1-243
- reading next line, 1-243
- searching by address, 1-239
- searching by name, 1-241

**nextkey function**

- 1-109
- 1-110

**NFS**

- creating a remote server, 1-438
- creating local asynchronous I/O server, 1-32
- nfssvc function, 1-438
- nice function, 1-439, 1-440
- nl_langinfo function, 1-441, 1-442
- nl_langinfo_r function, 1-441, 1-442
- nonlocal goto, 1-716
- setting jump point, 1-729
- nrand48 function, 1-121, 1-124
- ns, Xerox Network Systems
  - protocol family, 2-64, 2-65
- NS address
  - converting character strings to binary, 1-443

**converting to ASCII**

- 1-443

**ns_addr function**

- 1-443, 1-444

**nsip interface**

- 2-66

**ns_ntoa function**

- 1-443, 1-444

**ntohl function**

- 1-445

**ntohs function**

- 1-446

**NULL**

- 1-33

**null**

- 2-67

**null character**

- 1-33, 1-36

**O**

- Object file format, converting
  - osf_rose header, 1-111, 1-128
- omin function, 1-402, 1-405
- omout function, 1-402, 1-405
- opendir function, 1-453, 1-457
- open function, 1-447, 1-452
- opening a network file, 1-676
- opening a pipe to a process, 1-474
- openlog function, 1-776, 1-779
- operations on directories, 1-453
- operations on strings, 1-760
- operations on wide character strings, 1-941
- optarg, external variable, 1-244
- OSF/ROSE, 2-68
- osf_rose
  - converting header from canonical to readable form, 1-111
  - converting header from readable to canonical
form, 1-128
output
  completing, 1-868
  flushing, 1-872
  formatting, 1-937
  formatting parameters, 1-921
  printing and formatting, 1-476
output stream, writing characters to, 1-913
owner, changing for a file, 1-65

P

package
  returning address of symbol name in, 1-324
  returning address of symbol name in another process, 1-338
packet forwarding database, 2-104
page size, system versus hardware, 1-246
paging
  adding device for interleaved paging, 1-768
  expected behavior for a process, 1-362
parameters
  formatting for output, 1-921
  locale-dependent, 1-351
  variable length, 1-918
parameters, terminal, setting, 1-880
parent process ID, returning, 1-251
passwd, password files, 2-96, 2-97
password, reading, 1-247
pathconf function, 1-458, 1-461
patoi function, 1-342, 1-344
patol function, 1-342, 1-344
pause function, 1-462, 1-463
pclose function, 1-464, 1-465
peer name, returning for a socket, 1-249
permission, changing for a file, 1-61
perror function, 1-466
physical volumes, 2-35
pipe, 1-464, 1-474
  creating, 1-467
pipe function, 1-467, 1-468
plock function, 1-469, 1-470
poll function, 1-471, 1-473
popen function, 1-474, 1-475
power function, 1-148
pow function, 1-148, 1-150, 1-402, 1-405
printf function, 1-476, 1-482
printing formatted output, 1-937
printing output, 1-476
process
  accounting, 1-23
  advising the system of paging behavior, 1-362
  allocating timers for, 1-388
  attaching shared memory region, 1-696
  changing scheduling priority, 1-439
  cleanup on exit, 1-145
  clearing environment, 1-70
closing a pipe to, 1-464
creating a session, 1-689
creating via fork, 1-176
descriptor table size, 1-210
effective user ID, 1-287
examining and changing
actions, 1-706, 1-734
examining and changing
signal mask, 1-721
exiting, 1-145
forking, 1-176
generating signal to end,
1-16
group ID, 1-882
ID group, 1-876
locking text and/or data
segments in memory,
1-469
opening a pipe to, 1-474
pathname for controlling
terminal, 1-81
performing shared memory
control operations,
1-699
real user ID, 1-287
receiving signals, 1-706,
1-734
replacing signal mask, 1-732
restoring processor state,
1-724
return associated username,
1-107
returning and setting
scheduling priority,
1-252
returning CPU time used,
1-72
returning group ID, 1-251
returning ID, 1-251
returning ID of next loaded
module, 1-326
returning parent process ID,
1-251
returning real and effective
group IDs, 1-218
returning resource
utilization for, 1-265
returning supplementary
group set, 1-222
returning the effective user
ID, 1-287
returning the real user ID,
1-287
sending a signal to, 1-315
setting concurrent group set
for current, 1-307
setting group ID, 1-677,
1-684
setting ID, 1-689
setting real and effective
group ID, 1-682
setting real and effective
user ID's, 1-683
setting real, effective, and
saved set group ID,
1-664
setting real, effective, and
save set user ID,
1-694
setting the group access list,
1-666
setting user ID, 1-686
suspending, 1-462, 1-923,
1-932
suspending execution, 1-
732, 1-744, 1-914
terminating, 1-145
tracing execution of a child
process, 1-551
unloading specified module, 1-915
waiting for caught signals, 1-923
process group
  returning and setting scheduling priority, 1-252
  sending a signal to, 1-315
process ID, returning, 1-251
process image
  current, 1-136, 1-142
  new, 1-136, 1-142
processor, halting, 1-590
process timer, 1-606
  freeing, 1-625
profil function, 1-483, 1-484
profiling, starting and stopping, 1-483
Programmers Workbench Library, 1-342, 1-344
protocol, 1-808
  connection, 1-808
  endpoint, 1-822
  supporting sockets, 1-745
protocol entry
  retrieving, 1-258
  returning by name, 1-254
  returning by number, 1-256
protocols
  ICMP, 2-25
  IDP, 2-27
  IP, 2-30, 2-32
  NS, 2-64
  SPP, 2-118
  TCP, 2-125
  UDP, 2-165
protocols file
  closing, 1-132
opening, 1-258
reading, 1-258
searching by name, 1-254
searching by number, 1-256
setting and rewinding, 1-679
protocols name database, 2-98
pseudo-random numbers, generator functions, 1-121, 1-124
pseudo terminal driver, 2-99
pthread_attr_create function, 1-485, 1-486
pthread_attr_delete function, 1-487
pthread_attr_getstacksize function, 1-488, 1-489
pthread_attr_setstacksize function, 1-490, 1-491
pthread_cancel function, 1-492, 1-493
pthread_cleanup_pop function, 1-494, 1-495
pthread_cleanup_push function, 1-496, 1-497
pthread_condattr_create function, 1-510, 1-511
pthread_condattr_delete function, 1-512, 1-513
pthread_cond_broadcast function, 1-498, 1-499
pthread_cond_destroy function, 1-500, 1-501
pthread_cond_init function, 1-502, 1-503
pthread_cond_signal function, 1-504, 1-505
pthread_cond_timedwait function, 1-506, 1-507
pthread_cond_wait function, 1-
OSF/1 Programmer's Reference

508, 1-509
pthread_create function, 1-514, 1-515
pthread_detach function, 1-516, 1-517
pthread_equal function, 1-518
pthread_exit function, 1-519
pthread_getspecific function, 1-520, 1-521
pthread_join function, 1-522, 1-523
pthread_keycreate function, 1-524, 1-525
pthread_mutexattr_create function, 1-536, 1-537
pthread_mutexattr_delete function, 1-538
pthread_mutex_destroy function, 1-526, 1-527
pthread_mutex_init function, 1-528, 1-529
pthread_mutex_lock function, 1-530, 1-531
pthread_mutex_trylock function, 1-532, 1-533
pthread_mutex_unlock function, 1-534, 1-535
pthread_once function, 1-539, 1-540
pthread_self function, 1-541
pthread_setspecific function, 1-542, 1-544
pthread_setcancel function, 1-545, 1-546
pthread_setspecific function, 1-547, 1-548
pthread_testcancel function, 1-549
pthread_yield function, 1-550
ptrace function, 1-551, 1-554
pty, pseudo terminal driver, 2-99, 2-101
pushing a routine onto the cleanup stack, 1-496
pushing character back into input, 1-906
putc function, 1-555, 1-557
putchar function, 1-555, 1-557
putenv function, 1-558
putlong function, 1-559
putpwent function, 1-259, 1-261
puts function, 1-560, 1-561
putshort function, 1-562, 1-563
putw function, 1-564, 1-565
putwchar function, 1-564, 1-565
putw function, 1-555, 1-557
putws function, 1-566, 1-567
qsort function, 1-568, 1-569
querying locale, 1-672
querying name servers, 1-618
query messages for name servers, 1-615
queue
inserting element in, 1-309
removing element from, 1-309
quotactl function, 1-570, 1-572
R

raise function, 1-573
rand function, 1-574, 1-576
random function, 1-577, 1-579
random numbers, generating, 1-574, 1-577
rand_r function, 1-574, 1-576
rcmd function, 1-580, 1-581
rcvrel, 1-842
rcvdata, 1-844
readdir function, 1-453, 1-457
read function, 1-584, 1-587
readlink function, 1-588, 1-589
readv function, 1-584, 1-587
read-write offset, moving for a file, 1-360
realloc function, 1-364, 1-367
reboot function, 1-364, 1-367
rename function, 1-610, 1-612
repeat function, 1-342, 1-344
repl function, 1-342, 1-344
res_init function, 1-613, 1-614
res_mkquery function, 1-615, 1-617
resolver configuration file, 2-102, 2-103
resource utilization, returning information on, 1-265
res_send function, 1-618, 1-619
restoring execution context, 1-670
retrieving sockets, 1-626
retrieving terminal name, 1-896
retrieving values of system variables, 1-774
rewinddir function, 1-453, 1-457
rewind function, 1-184, 1-187
rewinding a network file, 1-676
rewinding the protocols file, 1-679
reexec function, 1-620, 1-622
rint function, 1-168, 1-170
rmdir function, 1-623, 1-624
rmtimer function, 1-625
root directory, changing effective, 1-68
route database, 2-104, 2-105
rpow function, 1-402, 1-405
rresvport function, 1-626, 1-627
ruserok function, 1-628, 1-629
removing a file, 1-608
removing routines from the cleanup stack, 1-494
rename function, 1-309
repeat function, 1-610, 1-612
rewind function, 1-342, 1-344
repl function, 1-342, 1-344
res_init function, 1-613, 1-614
res_mkquery function, 1-615, 1-617
resolver configuration file, 2-102, 2-103
resource utilization, returning information on, 1-265
res_send function, 1-618, 1-619
restoring execution context, 1-670
retrieving sockets, 1-626
retrieving terminal name, 1-896
retrieving values of system variables, 1-774
rewinddir function, 1-453, 1-457
rewind function, 1-184, 1-187
rewinding a network file, 1-676
rewinding the protocols file, 1-679
reexec function, 1-620, 1-622
rint function, 1-168, 1-170
rmdir function, 1-623, 1-624
rmtimer function, 1-625
root directory, changing effective, 1-68
route database, 2-104, 2-105
rpow function, 1-402, 1-405
rresvport function, 1-626, 1-627
ruserok function, 1-628, 1-629
S

satoi function, 1-342, 1-344
saving execution context, 1-670
sbrk function, 1-45, 1-46
scandir function, 1-630, 1-631
scanf function, 1-632, 1-637
scanning directory contents, 1-630
scheduler on threads, 1-550
scheduling priority
  returning and setting, 1-252
  setting, 1-439
sdiv function, 1-402, 1-405
searching for default domain name, 1-613
searching for Internet address, 1-613
search trees, 1-893
seed48 function, 1-121, 1-124
seekdir function, 1-453, 1-457
select function, 1-638, 1-641
semaphore
  initializing in mapped file, 1-409
  initializing in shared memory region, 1-409
  locking binary, 1-411
  removing binary, 1-413
  unlocking binary, 1-415
semaphores
  performing control operations on, 1-642
  performing operations on, 1-649
semaphore set
  creating, 1-646
  performing operations on, 1-649
removing, 1-642
  returning the ID for, 1-646
semctl function, 1-642, 1-645
semget function, 1-646, 1-648
semid_ds, 2-106
semop function, 1-649, 1-652
send
data, 1-851
data unit, 1-860
release, 1-858
send disconnect, 1-855
send function, 1-653, 1-654
sending signals, 1-573
sendmsg function, 1-655, 1-656
sendto function, 1-657, 1-659
sequence, collating, 2-4
server, authenticating clients, 1-628
service entry
  returning by name, 1-269
  returning by port, 1-271
service file entry, 1-688
services, service name database, 2-109
services file
  closing, 1-133
  opening, 1-273
  reading next line, 1-273
  searching by name, 1-269
  searching by port, 1-271
session, creating a new one, 1-689
setbuffer function, 1-660, 1-661
setbuf function, 1-660, 1-661
setclock function, 1-662, 1-663
setegid function, 1-684, 1-685
seteuid function, 1-686, 1-687
setfsent function, 1-214, 1-215

Index-26
setting process group ID, 1-882
setting system clock, 1-662
setting terminal parameters, 1-880
setting the protocols file, 1-679
setting the system clock, 1-756
setting user ID, 1-686
setting user limits, 1-899
setuid function, 1-694, 1-695
setusershell function, 1-288
setutent function, 1-289, 1-291
setvbuf function, 1-660, 1-661
shared library, executing a file with loader, 1-142
shared memory
attaching, 1-696
control operations, 1-699
detaching, 1-701
ID, 1-702
performing control operations, 1-699
returning and creating ID, 1-702
shared memory region
changing access modes, 1-406
initializing semaphore in, 1-409
unmapping, 1-430
shell commands, executing, 1-780
shell database, 2-110
shells file
closing, 1-288
reading, 1-288
rewinding, 1-288
shmat function, 1-696, 1-698
shmctl function, 1-699, 1-700
shmct function, 1-701
setting environment variables, 1-558
setting group ID, 1-684
setting locale, 1-672
setgrent function, 1-219, 1-221
setgroups function, 1-666, 1-667
sethostent function, 1-228, 1-229
sethostid function, 1-668
sethostname function, 1-669
setitimer function, 1-232, 1-234
setjmp function, 1-670, 1-671
setlinebuf function, 1-660, 1-661
setlocale function, 1-672, 1-675
setlocale_r function, 1-672, 1-675
setlogin function, 1-235, 1-236
setlogmask function, 1-776, 1-779
setnetent function, 1-676
setpgrp function, 1-677, 1-678
setpriority function, 1-252, 1-253
setprotoent function, 1-679
setpwent function, 1-259, 1-261
setquota function, 1-680, 1-681
setregid function, 1-682
setreuid function, 1-683
setrgid function, 1-684, 1-685
setrlimit function, 1-262, 1-264
setruid function, 1-686, 1-687
setsid function, 1-689
setsig function, 1-342, 1-344
setsig1 function, 1-342, 1-344
setsockopt function, 1-690, 1-693
setstate function, 1-577, 1-579
settimeofday function, 1-283, 1-284
setting process group ID, 1-882
setting system clock, 1-662
setting terminal parameters, 1-880
setting the protocols file, 1-679
setting the system clock, 1-756
setting user ID, 1-686
setting user limits, 1-899
setuid function, 1-694, 1-695
setusershell function, 1-288
setutent function, 1-289, 1-291
setvbuf function, 1-660, 1-661
shared library, executing a file with loader, 1-142
shared memory
attaching, 1-696
control operations, 1-699
detaching, 1-701
ID, 1-702
performing control operations, 1-699
returning and creating ID, 1-702
shared memory region
changing access modes, 1-406
initializing semaphore in, 1-409
unmapping, 1-430
shell commands, executing, 1-780
shell database, 2-110
shells file
closing, 1-288
reading, 1-288
rewinding, 1-288
shmat function, 1-696, 1-698
shmctl function, 1-699, 1-700
shmct function, 1-701
setting environment variables, 1-558
setting group ID, 1-684
setting locale, 1-672
shmget function, 1-702, 1-704
shm_id_ds, 2-111
short byte quantities, placing in byte stream, 1-562
short integer
  converting to host-byte order, 1-446
  converting to network-byte order, 1-298
shutdown function, 1-705
SIGABRT, 1-16
sigaction function, 1-706, 1-709
sigaddset function, 1-711, 1-713
sigblock function, 1-710
sigdelset function, 1-711, 1-713
sigemptyset function, 1-711, 1-713
sigfillset function, 1-711, 1-713
sighold function, 1-726, 1-728
sigignore function, 1-726, 1-728
siginterrupt function, 1-714, 1-715
sigismember function, 1-711, 1-713
siglongjmp function, 1-716, 1-717
signal
  adding to set of blocked signals, 1-710
  atomically changing set of blocked signals, 1-732
  blocked, 1-710, 1-720, 1-721
  defining alternate stacks, 1-730
  examining pending, 1-720
  returning from, 1-724
  sending to a process or process group, 1-315
  setting and getting stack context, 1-730
  setting mask, 1-721
  suspending process execution, 1-732
taking action upon receipt, 1-706, 1-734
to abort current process, 1-16
signal file, 2-113, 2-117
signal function, 1-706, 1-709
signal handling for nonlocal goto, 1-716
signal management, compatibility interfaces, 1-726
signal masks, creating and manipulating, 1-711
signals
  blocking, 1-718
  interrupting functions, 1-714
  sending, 1-573
sigpause function, 1-718, 1-719
sigpending function, 1-720
sigprocmask function, 1-721, 1-723
sigrelse function, 1-726, 1-728
sigreturn function, 1-724, 1-725
sigset function, 1-726, 1-728
sigsetjmp function, 1-729
sigsetmask function, 1-721, 1-723
sigstack function, 1-730, 1-731
sigsuspend function, 1-732, 1-733
sigvec function, 1-734, 1-736
sigwait function, 1-737, 1-738
sin function, 1-739, 1-741
sinh function, 1-742, 1-743
sleep function, 1-744
sname function, 1-342, 1-344
sndrel, 1-858
socket
  accepting a connection, 1-19
  binding a name, 1-43
  controlling socket communication, 1-690
creating, 1-745
creating a connected pair, 1-748
creating by accepting a connection, 1-19
creating end points, 1-745
disabling receive and send operations, 1-705
establishing a connection, 1-75
inherited, 1-277
inherited by a process, 1-249
listening for connections, 1-347
locally bound address, 1-277
name, 1-43
options on, 1-279
receive and send operations, 1-705
receiving messages from connected, 1-593
receiving messages from connected or unconnected, 1-598
receiving messages from unconnected, 1-595
retrieving, 1-626
returning name, 1-277
returning options on, 1-279
returning peer name, 1-249
sending messages through connected, 1-653
sending messages through unconnected, 1-657
sending messages using a message structure, 1-655
setting options, 1-690
socket function, 1-745, 1-747
socketpair, creating, 1-748
socketpair function, 1-748, 1-749
sockets, 2-25, 2-58, 2-118, 2-125, 2-165
sorting directory contents, 1-630
sorting tables, 1-568
special file, creating, 1-383
spp, Xerox Sequenced Packet protocol, 2-118, 2-119
sprintf function, 1-476, 1-482
sqrt function, 1-750, 1-751
square root function, 1-750
srand function, 1-574, 1-576
srand48 function, 1-121, 1-124
srandom function, 1-577, 1-579
sscanf function, 1-632, 1-637
stab file, 2-120, 2-122
stack
defining alternates, 1-730
setting and getting context, 1-730
stack size attribute
finding value of, 1-488
setting value of, 1-490
standard IO functions, 1-555
statfs function, 1-754, 1-755
stat function, 1-752, 1-753
status, controlling for a file, 1-155
stderr file descriptor, 2-20
stdin file descriptor, 2-20
stdout file descriptor, 2-20
stdlib.h, 1-17
step function, 1-601, 1-605
stime function, 1-756
store function, 1-109, 1-110
strcat function, 1-760, 1-766
strchr function, 1-760, 1-766
strcoll function, 1-760, 1-766
strepy function, 1-760, 1-766
strcsps function, 1-760, 1-766
strdup function, 1-760, 1-766

stream
  clearing errors, 1-71
  closing, 1-152
  flushing, 1-152
  getting a string from, 1-294
  getting a string from stdin, 1-267
  locking stdio, 1-167
  mapping pointer to file descriptor, 1-163
  opening, 1-171
  performing binary
    input/output, 1-179
  returning file pointer for, 1-184
  setting file pointer for, 1-184
  testing EOF on, 1-161
  testing error indicator on, 1-162
  unlocking stdio, 1-195
strend function, 1-342, 1-344
strerror function, 1-760, 1-766
strftime function, 1-757, 1-759

string
  converting character to
    floating point, 1-33
  converting character to
    integer, 1-35
  getting from a stream, 1-267, 1-294

string conversion
  character to floating point, 1-33

character to integer, 1-35
string manipulation, 1-760
string operations, 1-760
strings
  manipulating in memory area, 1-374
  writing out, 1-560
strlen function, 1-760, 1-766
strncat function, 1-760, 1-766
strncpy function, 1-760, 1-766
strpbrk function, 1-760, 1-766
strrchr function, 1-760, 1-766
strspn function, 1-760, 1-766
strstr function, 1-760, 1-766
strtok function, 1-760, 1-766
strtok_r function, 1-760, 1-766
strtol function, 1-35, 1-36, 1-38
strtoul function, 1-35, 1-38
structures, synchronize, 1-863
strxfrm function, 1-760, 1-766
substr function, 1-342, 1-344
suspending a process, 1-462
suspending process execution, 1-744, 1-914
suspending threads, 1-737
swab function, 1-767
swapon function, 1-768, 1-769
swapping, adding device for, 1-768
swapping bytes, 1-767
symbolic link, reading from, 1-588
symbol name
  returning address in another
    process package, 1-338
  returning address in
    package, 1-324
symbol table types, 2-120
symlink function, 1-770, 1-772
core, 1-863
core function, 1-773
synchronize, library, 1-863
sysconf function, 1-774, 1-775
syslog function, 1-776, 1-779
system
  getting name of, 1-904
  identifying, 1-904
  rebooting, 1-590
system address space, returning configuration of, 1-198
system clock
  getting time, 1-884
  returning current value, 1-203, 1-285
  setting, 1-662, 1-756
  synchronization, 1-25
  times of process and child process, 1-885
system function, 1-780, 1-781
system log, 1-776
system page size, returning, 1-246
system resources, returning and setting limits for, 1-262
system time
  adjusting, 1-25
  returning and setting, 1-283
system timezone, returning and setting, 1-283
system variables, retrieving values of, 1-774

table
  performing linear search and update, 1-358
  sorting, 1-568
  tables, collating, 2-4
t_accept function, 1-782, 1-785
t_alloc function, 1-786, 1-789
tan function, 1-739, 1-741
tanh function, 1-742, 1-743
t_bind function, 1-790, 1-794
tcdrain function, 1-868, 1-869
tcflow function, 1-870, 1-871
tcflush function, 1-872, 1-873
tcgetattr function, 1-874, 1-875
tcgetpgrp function, 1-876, 1-877
t_close function, 1-795, 1-796
t_connect function, 1-797, 1-802
tcp, Transmission Control Protocol, 2-125, 2-126
tcsendbreak function, 1-878, 1-879
tcsetattr function, 1-880, 1-881
tcsetpgrp function, 1-882, 1-883
tdelete function, 1-893, 1-895
telldir function, 1-453, 1-457
tempnam function, 1-888, 1-889
temporary file
  creating, 1-887
  name, 1-888
  terminal drivers, 2-151
  terminal interface, 2-139, 2-151
  terminal name, 1-896
  terminal parameters, 1-874
  setting, 1-880
  terminals, capabilities of, 2-127
  terminating threads, 1-492, 1-519, 1-522
terminfo file, 2-127, 2-138
termios file, 2-139
t_error function, 1-803, 1-804
tfind function, 1-893, 1-895
t_free function, 1-805, 1-807
t_getinfo function, 1-808, 1-811
t_getstate function, 1-812, 1-813
thread
  asynchronous cancelability of, 1-542
  binding value to a key, 1-547
  calling initializing routine, 1-539
  cleanup stack
    adding a routine onto, 1-496
    removing a routine from, 1-494
  comparing identifiers, 1-518
  creating, 1-514
  creating a cancellation point, 1-549
  creating a key, 1-524
  creating a mutex, 1-528
  creating attributes object, 1-510
  creating mutex attribute object, 1-536
  creating variable, 1-502
  deleting a mutex, 1-526
  deleting attribute objects, 1-512
  deleting mutex attribute object, 1-538
  destroying variable, 1-500
  detaching, 1-516
  general cancelability of, 1-545
  ID of, 1-541
  locking a mutex, 1-530, 1-532
  returning key value, 1-520
  scheduler on, 1-550
  suspending, 1-737
  termination of, 1-492, 1-519
  unlocking a mutex, 1-534
  waiting for, 1-522
  waiting on, 1-506, 1-508
  waking up, 1-498, 1-504
thread attribute object
  deletion of, 1-487
  setting stack size, 1-490
thread attributes object
  creation of, 1-485
  stack size attribute, 1-488
time conversion, 1-757
time conversion functions, 1-83, 1-88
time function, 1-884
timeout
  for interval timers, 1-27
  setting and returning for interval timers, 1-232
timeout intervals for processes, 1-606
timer, allocating per-process, 1-388
timers, 1-625
times function, 1-885, 1-886
times of processes, 1-885
time units
  converting to other time units, 1-83
  converting to strings, 1-83
  storing for later processing, 1-83
TI OCGWIN SZ function, 2-164
TI OPC P KT function, 2-99
TI OCREMOTE function, 2-101
TI OCSTART function, 2-99
TI OCSTOP function, 2-99
TI OC SWIN SZ function, 2-164
TI OCU CNTL function, 2-100
T _listen function, 1-814, 1-817
T _look function, 1-818, 1-821
tmpfile function, 1-887
tmpnam function, 1-888, 1-889
toascii function, 1-78, 1-80
tolower function, 1-78, 1-80
tolower macro, 1-78, 1-80
t_open function, 1-822, 1-826
t_optmgmt function, 1-827, 1-830
toupper function, 1-78, 1-80
toupper macro, 1-78, 1-80
t tracing of child process execution, 1-551
Transmission Control Protocol, 2-125
transport endpoint, 1-822
transport endpoint"unbind, 1-866
t_rcvconnect function, 1-834,
1-837
t_rcvdis function, 1-838, 1-841
t_rcv function, 1-831, 1-833
t_rcvrel function, 1-842, 1-843
t_rcvudata function, 1-844, 1-847
t_rcvuderr function, 1-848, 1-850
trees, binary search, 1-893
trigonometric functions, 1-739
trnslat function, 1-342, 1-344
truncate function, 1-890, 1-892
tsearch function, 1-893, 1-895
t_snddis function, 1-855, 1-857
t_snd function, 1-851, 1-854
t_sndrel function, 1-858, 1-859
t_sndudata function, 1-860, 1-862
t_sync function, 1-863, 1-865
tty interface, 2-151, 2-164
ttyname function, 1-896, 1-897
tyslot function, 1-898
t_unbind function, 1-866, 1-867
twalk function, 1-893, 1-895
tzset function, 1-83, 1-88

U
ualarm function, 1-27, 1-28
udp, User Datagram Protocol, 2-
165, 2-166
ulimit function, 1-899, 1-900
umask function, 1-901
umount function, 1-395, 1-399, 1-
902, 1-903
uname function, 1-904
unbind, transport endpoint, 1-866
ungetc function, 1-906, 1-907
ungetwc function, 1-906, 1-907
unlink function, 1-908, 1-909
unload function, 1-910, 1-911
unloading modules, 1-910
unlocked_getc function, 1-912
unlocked_getchar function, 1-912
unlocked_putchar, 1-913
unlocked_putchar, 1-913
unlocking mutexes for threads,
1-534
unlockit function, 1-342, 1-344
user, returning and setting
scheduling priority, 1-252
user database
  accessing basic group information, 1-219
  defined, 1-259
  manipulating entry in, 1-259
User Datagram Protocol, 2-165
userdir function, 1-342, 1-344
userexit function, 1-342, 1-344
user ID
  real and effective, 1-683
  real, effective, and saved set, 1-694
  returning effective for a process, 1-287
  returning real for a process, 1-287
  setting, 1-686, 1-694
  setting real and effective, 1-683
  setting real, effective, and saved set, 1-694
user limits, setting and getting, 1-899
username, return for process, 1-107
username function, 1-342, 1-344
user password, 1-259
user's entry in utmp file, 1-898
user shell, returning name of, 1-288
usleep function, 1-914
ustatsfs function, 1-754, 1-755
utime function, 1-915, 1-917
utimes function, 1-915, 1-917
utmp file
  changing filename, 1-289
  closing, 1-289
  opening, 1-289
  positioning in, 1-289
  reading next entry, 1-289
resetting input stream, 1-289
writing to, 1-289
utmpname function, 1-289, 1-291

V

value, negating, 1-437
values in threads, 1-520
varargs function, 1-918, 1-920
variable length parameters, 1-918
verify function, 1-342, 1-344
vfprintf function, 1-921, 1-922
virtual address, unloading specified module, 1-915
virtual disks, 2-35
virtual memory
  attaching shared memory region, 1-696
  mapping an object into, 1-390
  shared memory region, 1-696
vprintf function, 1-921, 1-922
vsprintf function, 1-921, 1-922
vtimes function, 1-265, 1-266

W

wait, 1-16
wait function, 1-923
wait3 function, 1-923
waiting for output, 1-868
waiting on threads, 1-506, 1-508
waitpid, 1-16
waitpid function, 1-923
waking up threads, 1-498, 1-504
wcstombs function, 1-928, 1-929
wctomb function, 1-930, 1-931
wide character, converting to multibyte, 1-930
wide character string, converting to multibyte, 1-928
wide character strings, operations on, 1-941
word, getting from input stream, 1-292
write function, 1-932, 1-936
writev function, 1-932, 1-936
writing out a string, 1-560, 1-566
writing out characters, 1-555
writing out wide characters, 1-564
wsprintf function, 1-937, 1-938
wscanf function, 1-939, 1-940
wstrcat function, 1-941, 1-944
wstrchr function, 1-941, 1-944
wstrcmp function, 1-941, 1-944
wstrcpy function, 1-941, 1-944
wstrcspn function, 1-941, 1-944
wstrdup function, 1-941, 1-944
wstrlen function, 1-941, 1-944
wstrlen function, 1-941, 1-944
wstrncat function, 1-941, 1-944
wstrncpy function, 1-941, 1-944
wstrpbrk function, 1-941, 1-944
wstrrchr function, 1-941, 1-944
wstrspn function, 1-941, 1-944
wstok function, 1-941, 1-944

X
xalloc function, 1-342, 1-344
xcreat function, 1-342, 1-344
Xerox Internet Protocol, 2-27
Xerox Network Systems Protocol, 2-64
Xerox NS address
converting character strings to binary, 1-443
converting to ASCII, 1-443
Xerox Sequenced Packet protocol, 2-118
xfreeall function, 1-342, 1-344
xfree function, 1-342, 1-344
xlink function, 1-342, 1-344
xmsg function, 1-342, 1-344
xopen function, 1-342, 1-344
xpipe function, 1-342, 1-344
XTI
error, 1-803
t_accept, 1-782
t_alloc, 1-786
t_bind, 1-790
t_close, 1-795
t_connect, 1-797
t_error, 1-803
t_free, 1-805
t_getinfo, 1-808
t_getstate, 1-812
t_listen, 1-814
t_look, 1-818
t_open, 1-822
t_optmgmt, 1-827
t_rcv, 1-831
t_rcvconnect, 1-834
t_rcvdis, 1-838
t_rcvrel, 1-842
t_rcvudata, 1-844

Index-35
t_rcvuderr, 1-848
    t_snd, 1-851
    t_snddis, 1-855
    t_sndrel, 1-858
    t_sndudata, 1-860
    t_sync, 1-863
    t_unbind, 1-866
xunlink function, 1-342, 1-344
xwrite function, 1-342, 1-344

Y

y0 function, 1-41, 1-42
y1 function, 1-41, 1-42
yn function, 1-41, 1-42

Z

zero function, 1-342, 1-344
zeropad function, 1-342, 1-344
OPEN SOFTWARE FOUNDATION™
INFORMATION REQUEST FORM

Please send to me the following:

( ) OSF™ Membership Information
( ) OSF/1™ License Materials
( ) OSF/1 Training Information

Contact Name

Company Name

Street Address

Mail Stop

City State Zip

Phone FAX

Electronic Mail

MAIL TO:

Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142

Attn: OSF/1

For more information about OSF/1 call OSF Direct Channels at 617 621 7300.