Your comments on our products and publications are welcome. A postage-paid form is provided for this purpose on the last page of this manual.
### 2. System Calls

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept(2)</td>
<td>accept a connection on a socket</td>
</tr>
<tr>
<td>access(2)</td>
<td>determine accessibility of file</td>
</tr>
<tr>
<td>acct(2)</td>
<td>turn accounting on or off</td>
</tr>
<tr>
<td>adjtime(2)</td>
<td>correct the time to allow synchronization of the system clock</td>
</tr>
<tr>
<td>bind(2)</td>
<td>bind a name to a socket</td>
</tr>
<tr>
<td>brk(2)</td>
<td>change data segment size</td>
</tr>
<tr>
<td>cachecl(2)</td>
<td>mark pages cacheable or uncacheable</td>
</tr>
<tr>
<td>cacheflush(2)</td>
<td>flush contents of instruction and, or data cache</td>
</tr>
<tr>
<td>chdir(2)</td>
<td>change current working directory</td>
</tr>
<tr>
<td>chown(2)</td>
<td>change owner and group of a file</td>
</tr>
<tr>
<td>close(2)</td>
<td>delete a descriptor</td>
</tr>
<tr>
<td>connect(2)</td>
<td>initiate a connection on a socket</td>
</tr>
<tr>
<td>creat(2)</td>
<td>create a new file</td>
</tr>
<tr>
<td>dup(2)</td>
<td>duplicate a descriptor</td>
</tr>
<tr>
<td>execve(2)</td>
<td>execute a file</td>
</tr>
<tr>
<td>exit(2)</td>
<td>terminate a process</td>
</tr>
<tr>
<td>fcntl(2)</td>
<td>file control</td>
</tr>
<tr>
<td>fixaxd(2)</td>
<td>fix address exceptions (unaligned references)</td>
</tr>
<tr>
<td>flock(2)</td>
<td>apply or remove an advisory lock on an open file</td>
</tr>
<tr>
<td>fork(2)</td>
<td>create a new process</td>
</tr>
<tr>
<td>fp_setintr(2)</td>
<td>generate a SIGFPE signal on floating-point interrupts</td>
</tr>
<tr>
<td>fsync(2)</td>
<td>synchronize a file's in-core state with that on disk</td>
</tr>
<tr>
<td>getdirentries(2)</td>
<td>gets directory entries in a filesystem independent format</td>
</tr>
<tr>
<td>getdomainname(2)</td>
<td>get, set name of current domain</td>
</tr>
<tr>
<td>getdtablesize(2)</td>
<td>get descriptor table size</td>
</tr>
<tr>
<td>getgid(2)</td>
<td>get group identity</td>
</tr>
<tr>
<td>getgroups(2)</td>
<td>get group access list</td>
</tr>
<tr>
<td>gethostid(2)</td>
<td>get, set unique identifier of current host</td>
</tr>
<tr>
<td>gethostname(2)</td>
<td>get, set name of current host</td>
</tr>
<tr>
<td>getitimer(2)</td>
<td>get, set value of interval timer</td>
</tr>
<tr>
<td>getpagesize(2)</td>
<td>get system page size</td>
</tr>
<tr>
<td>getpeername(2)</td>
<td>get name of connected peer</td>
</tr>
<tr>
<td>getpgid(2)</td>
<td>get process group</td>
</tr>
<tr>
<td>getpid(2)</td>
<td>get process identification</td>
</tr>
<tr>
<td>getpriority(2)</td>
<td>get, set program scheduling priority</td>
</tr>
<tr>
<td>getrlimit(2)</td>
<td>control maximum system resource consumption</td>
</tr>
<tr>
<td>getrusage(2)</td>
<td>get information about resource utilization</td>
</tr>
<tr>
<td>getsockname(2)</td>
<td>get socket name</td>
</tr>
<tr>
<td>getsockopt(2)</td>
<td>get and set options on sockets</td>
</tr>
<tr>
<td>gettimeofday(2)</td>
<td>get, set date and time</td>
</tr>
<tr>
<td>getuid(2)</td>
<td>get user identity</td>
</tr>
<tr>
<td>hwconf(2)</td>
<td>get or set hardware configuration</td>
</tr>
<tr>
<td>intro(2)</td>
<td>introduction to system calls and error numbers</td>
</tr>
<tr>
<td>ioctl(2)</td>
<td>ioctl</td>
</tr>
<tr>
<td>kill(2)</td>
<td>send signal to a process</td>
</tr>
<tr>
<td>killpg(2)</td>
<td>send signal to a process group</td>
</tr>
<tr>
<td>kopt(2)</td>
<td>get or set kernel options</td>
</tr>
<tr>
<td>link(2)</td>
<td>make a hard link to a file</td>
</tr>
<tr>
<td>listen(2)</td>
<td>listen for connections on a socket</td>
</tr>
<tr>
<td>lseek(2)</td>
<td>move read, write pointer</td>
</tr>
<tr>
<td>mipsfpunit(2)</td>
<td>enabling and dissabling the floating-point unit</td>
</tr>
<tr>
<td>mkdir(2)</td>
<td>make a directory file</td>
</tr>
<tr>
<td>mknod(2)</td>
<td>make a special file</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mmap(2)</td>
<td>map or unmap pages of memory</td>
</tr>
<tr>
<td>mount(2)</td>
<td>mount file system</td>
</tr>
<tr>
<td>nfssvc(2)</td>
<td>async_daemon NFS daemons</td>
</tr>
<tr>
<td>open(2)</td>
<td>open a file for reading or writing, or create a new file</td>
</tr>
<tr>
<td>pipe(2)</td>
<td>create an interprocess communication channel</td>
</tr>
<tr>
<td>profil(2)</td>
<td>process trace</td>
</tr>
<tr>
<td>ptrace(2)</td>
<td>manipulate disk quotas</td>
</tr>
<tr>
<td>quota(2)</td>
<td>manipulate disk quotas</td>
</tr>
<tr>
<td>quotactl(2)</td>
<td>read input</td>
</tr>
<tr>
<td>read(2)</td>
<td>read value of a symbolic link</td>
</tr>
<tr>
<td>readlink(2)</td>
<td>reboot system or halt processor</td>
</tr>
<tr>
<td>recv(2)</td>
<td>receive a message from a socket</td>
</tr>
<tr>
<td>rename(2)</td>
<td>change the name of a file</td>
</tr>
<tr>
<td>rmdir(2)</td>
<td>remove a directory file</td>
</tr>
<tr>
<td>select(2)</td>
<td>synchronous I,O multiplexing</td>
</tr>
<tr>
<td>send(2)</td>
<td>send a message from a socket</td>
</tr>
<tr>
<td>setgroups(2)</td>
<td>set group access list</td>
</tr>
<tr>
<td>setpgid(2)</td>
<td>set process group</td>
</tr>
<tr>
<td>setquota(2)</td>
<td>enable,disable quotas on a file system</td>
</tr>
<tr>
<td>setregid(2)</td>
<td>set real and effective group ID</td>
</tr>
<tr>
<td>setreuid(2)</td>
<td>set real and effective user ID’s</td>
</tr>
<tr>
<td>shutdown(2)</td>
<td>shut down part of a full-duplex connection</td>
</tr>
<tr>
<td>sigblock(2)</td>
<td>block signals</td>
</tr>
<tr>
<td>sigpause(2)</td>
<td>atomically release blocked signals and wait for interrupt</td>
</tr>
<tr>
<td>sigreturn(2)</td>
<td>return from signal</td>
</tr>
<tr>
<td>sigsetmask(2)</td>
<td>set current signal mask</td>
</tr>
<tr>
<td>sigstack(2)</td>
<td>set and,or get signal stack context</td>
</tr>
<tr>
<td>sigvec(2)</td>
<td>software signal facilities</td>
</tr>
<tr>
<td>socket(2)</td>
<td>create an endpoint for communication</td>
</tr>
<tr>
<td>socketpair(2)</td>
<td>create a pair of connected sockets</td>
</tr>
<tr>
<td>stat(2)</td>
<td>get file status</td>
</tr>
<tr>
<td>stats(2)</td>
<td>get file system statistics</td>
</tr>
<tr>
<td>swapon(2)</td>
<td>add a swap device for interleaved paging,swapping</td>
</tr>
<tr>
<td>symlink(2)</td>
<td>make symbolic link to a file</td>
</tr>
<tr>
<td>sync(2)</td>
<td>update super-block</td>
</tr>
<tr>
<td>syscall(2)</td>
<td>indirect system call</td>
</tr>
<tr>
<td>truncate(2)</td>
<td>truncate a file to a specified length</td>
</tr>
<tr>
<td>umask(2)</td>
<td>set file creation mode mask</td>
</tr>
<tr>
<td>uname(2)</td>
<td>get general system information</td>
</tr>
<tr>
<td>unlink(2)</td>
<td>remove directory entry</td>
</tr>
<tr>
<td>unmount(2)</td>
<td>remove a file system</td>
</tr>
<tr>
<td>utimes(2)</td>
<td>set file times</td>
</tr>
<tr>
<td>vfork(2)</td>
<td>spawn new process in a virtual memory efficient way</td>
</tr>
<tr>
<td>vhangup(2)</td>
<td>virtually “hangup” the current control terminal</td>
</tr>
<tr>
<td>wait(2)</td>
<td>wait for process to terminate</td>
</tr>
<tr>
<td>write(2)</td>
<td>write output</td>
</tr>
</tbody>
</table>

### 3. Library Subroutines

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort(3)</td>
<td>generate a fault</td>
</tr>
<tr>
<td>abort(3f)</td>
<td>terminate</td>
</tr>
<tr>
<td>abs(3)</td>
<td>integer absolute value</td>
</tr>
<tr>
<td>access(3f)</td>
<td>determine accessibility of a file</td>
</tr>
<tr>
<td>alarm(3c)</td>
<td>schedule signal after specified time</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>alarm(3f)</td>
<td>execute</td>
</tr>
<tr>
<td>asinh(3m)</td>
<td>inverse</td>
</tr>
<tr>
<td>assert(3)</td>
<td>program verification</td>
</tr>
<tr>
<td>atof(3)</td>
<td>convert ASCII to numbers</td>
</tr>
<tr>
<td>bstring(3)</td>
<td>bit and byte string operations</td>
</tr>
<tr>
<td>byteorder(3n)</td>
<td>convert values between host and network byte order</td>
</tr>
<tr>
<td>chdir(3f)</td>
<td>change</td>
</tr>
<tr>
<td>chmod(3f)</td>
<td>change</td>
</tr>
<tr>
<td>crypt(3)</td>
<td>DES encryption</td>
</tr>
<tr>
<td>ctime(3)</td>
<td>convert date and time to ASCII</td>
</tr>
<tr>
<td>ctype(3)</td>
<td>character classification macros</td>
</tr>
<tr>
<td>curses(3x)</td>
<td>screen functions with “optimal” cursor motion</td>
</tr>
<tr>
<td>dbm(3x)</td>
<td>data base subroutines</td>
</tr>
<tr>
<td>directory(3)</td>
<td>directory operations</td>
</tr>
<tr>
<td>disassembler(3x)</td>
<td>disassemble</td>
</tr>
<tr>
<td>dr(3n)</td>
<td>library routines for external data representation</td>
</tr>
<tr>
<td>ecvt(3)</td>
<td>output conversion</td>
</tr>
<tr>
<td>emulate_branch(3)</td>
<td>MIPS branch emulation</td>
</tr>
<tr>
<td>end(3)</td>
<td>last locations in program</td>
</tr>
<tr>
<td>end(3)</td>
<td>first locations in program</td>
</tr>
<tr>
<td>end(3)</td>
<td>runtime procedure table</td>
</tr>
<tr>
<td>erf(3m)</td>
<td>error</td>
</tr>
<tr>
<td>ethers(3n)</td>
<td>Ethernet address mapping operations</td>
</tr>
<tr>
<td>etime(3f)</td>
<td>return</td>
</tr>
<tr>
<td>examples(3)</td>
<td>library of sample programs</td>
</tr>
<tr>
<td>exec(3)</td>
<td>execute a file</td>
</tr>
<tr>
<td>exit(3)</td>
<td>terminate a process after flushing any pending output</td>
</tr>
<tr>
<td>exp(3m)</td>
<td>exponential, close or flush a stream</td>
</tr>
<tr>
<td>fclose(3s)</td>
<td>return</td>
</tr>
<tr>
<td>fdate(3f)</td>
<td>stream status inquiries</td>
</tr>
<tr>
<td>floor(3m)</td>
<td>absolute</td>
</tr>
<tr>
<td>flush(3f)</td>
<td>flush</td>
</tr>
<tr>
<td>fopen(3s)</td>
<td>open a stream</td>
</tr>
<tr>
<td>fork(3f)</td>
<td>create</td>
</tr>
<tr>
<td>fp_class(3)</td>
<td>classes of IEEE floating-point values</td>
</tr>
<tr>
<td>fpc(3)</td>
<td>floating-point control registers</td>
</tr>
<tr>
<td>fpi(3)</td>
<td>floating-point interrupt analysis</td>
</tr>
<tr>
<td>fread(3s)</td>
<td>buffered binary input, output</td>
</tr>
<tr>
<td>frexp(3)</td>
<td>split into mantissa and exponent</td>
</tr>
<tr>
<td>fseek(3f)</td>
<td>reposition</td>
</tr>
<tr>
<td>fseek(3s)</td>
<td>reposition a stream</td>
</tr>
<tr>
<td>getarg(3f)</td>
<td>return</td>
</tr>
<tr>
<td>getc(3f)</td>
<td>get character or word from stream</td>
</tr>
<tr>
<td>getcwd(3f)</td>
<td>get</td>
</tr>
<tr>
<td>getdisk(3)</td>
<td>get disk description by its name</td>
</tr>
<tr>
<td>getenv(3)</td>
<td>manipulate environmental variables</td>
</tr>
<tr>
<td>getenv(3f)</td>
<td>get</td>
</tr>
<tr>
<td>getsent(3)</td>
<td>get file</td>
</tr>
<tr>
<td>getgrent(3)</td>
<td>get group file entry</td>
</tr>
<tr>
<td>gethostbyname(3n)</td>
<td>get network</td>
</tr>
<tr>
<td>getlog(3f)</td>
<td>get</td>
</tr>
<tr>
<td>getlogin(3)</td>
<td>get login name</td>
</tr>
<tr>
<td>getmntent(3)</td>
<td>get file system</td>
</tr>
<tr>
<td>getnetent(3n)</td>
<td>get network</td>
</tr>
</tbody>
</table>
getnetgrent(3n) .................................................... get network group entry
getopt(3) .............................................................. get option letter from argv
getpass(3) .............................................................. read a password
getpid(3f) .............................................................. get
getprotoent(3n) ....................................................... get
getpw(3c) .............................................................. get name from uid
getpwnent(3) ............................................................ get
getrpcent(3n) ........................................................... get RPC entry
getrpcport(3r) .......................................................... get RPC port number
gets(3s) ................................................................. get a string from a stream
getsevrent(3n) ......................................................... get
gettyent(3) .............................................................. get ttys file entry
getuid(3f) .............................................................. get
getusershells(3) ....................................................... get legal user shells
getwd(3) ................................................................. get current working directory pathname
hypot(3m) .............................................................. Euclidean return
date(3f) ................................................................. copysign,
dtctr(3m) .............................................................. Internet
setgroups(3) ............................................................ initialize group access list
insq(3) ................................................................. insert,remove element from a queue
intro(3) ................................................................. introduction to C library functions
intro(3f) ................................................................. introduction
j0(3m) ................................................................. bessel
kill(3f) ................................................................. send
ldapread(3x) .......................................................... read
ldclose(3x) ............................................................. close
ldfread(3x) ............................................................ read
ldgetaux(3x) ........................................................... retrieve
ldgetenv(3x) ........................................................... retrieve
ldgetpd(3x) ............................................................ retrieve
ldlread(3x) ............................................................ manipulate
ldlseek(3x) ............................................................. seek
ldohseek(3x) ........................................................... seek
ldopen(3x) ............................................................. open
ldrseek(3x) ............................................................. seek
lshread(3x) ............................................................ read
ldsseek(3x) ............................................................. seek
ldtbread(3x) ............................................................ read
ldtbsseek(3x) .......................................................... seek
len(3f) ................................................................. return
lgamma(3m) ........................................................... log
lib26el48(3x) .......................................................... subroutines for the HP 2648 graphics terminal
libraries(3) ............................................................ overview of VADS libraries
link(3f) ................................................................. make
loc(3f) ................................................................. return
lockf(3) ................................................................. advisory record locking on files
malloc(3) .............................................................. memory allocator
math(3m) .............................................................. introduction
memory(3) ............................................................. memory operations
mktemp(3) .............................................................. make a unique file name
monitor(3) .............................................................. prepare execution profile
mount(3r) .............................................................. keep track of remotely mounted filesystems
mp(3r) ................................................................. multiple precision integer arithmetic
ndbm(3) ................................................................. data base subroutines
nice(3c) ............................................................... set program priority
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlist(3x)</td>
<td>get</td>
</tr>
<tr>
<td>ns(3n)</td>
<td>Xerox NS(tm)</td>
</tr>
<tr>
<td>pause(3c)</td>
<td>stop until signal</td>
</tr>
<tr>
<td>perror(3)</td>
<td>system error messages</td>
</tr>
<tr>
<td>perror(3f)</td>
<td>get</td>
</tr>
<tr>
<td>plot(3x)</td>
<td>graphics interface</td>
</tr>
<tr>
<td>popen(3)</td>
<td>initiate I/O fo, from a process</td>
</tr>
<tr>
<td>printf(3s)</td>
<td>formatted output conversion</td>
</tr>
<tr>
<td>psignal(3)</td>
<td>system signal messages</td>
</tr>
<tr>
<td>publiclib(3)</td>
<td>public domain packages written in Ada</td>
</tr>
<tr>
<td>putc(3f)</td>
<td>write</td>
</tr>
<tr>
<td>puts(3s)</td>
<td>put character or word on a stream</td>
</tr>
<tr>
<td>putc(3s)</td>
<td>put a string on a stream</td>
</tr>
<tr>
<td>qsort(3)</td>
<td>quicker sort</td>
</tr>
<tr>
<td>qsort(3f)</td>
<td>quick</td>
</tr>
<tr>
<td>rand(3c)</td>
<td>random number generator</td>
</tr>
<tr>
<td>rand(3f)</td>
<td>random</td>
</tr>
<tr>
<td>random(3)</td>
<td>better random number generator</td>
</tr>
<tr>
<td>ranhash(3x)</td>
<td>access</td>
</tr>
<tr>
<td>rcmd(3)</td>
<td>routines for returning a stream to a remote command</td>
</tr>
<tr>
<td>regex(3)</td>
<td>regular expression handler</td>
</tr>
<tr>
<td>resolver(3)</td>
<td>resolver routines</td>
</tr>
<tr>
<td>rex(3r)</td>
<td>remote execution protocol</td>
</tr>
<tr>
<td>rexec(3)</td>
<td>return stream to a remote command</td>
</tr>
<tr>
<td>rmusers(3r)</td>
<td>return information about users on remote machines</td>
</tr>
<tr>
<td>rpc(3n)</td>
<td>library routines for remote procedure calls</td>
</tr>
<tr>
<td>rquota(3r)</td>
<td>implement quotas on remote machines</td>
</tr>
<tr>
<td>rwall(3r)</td>
<td>write to specified remote machines</td>
</tr>
<tr>
<td>scandir(3)</td>
<td>scan a directory</td>
</tr>
<tr>
<td>scanf(3s)</td>
<td>formatted input conversion</td>
</tr>
<tr>
<td>setbuf(3s)</td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>setjmp(3)</td>
<td>non-local goto</td>
</tr>
<tr>
<td>setuid(3)</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>siginterrupt(3)</td>
<td>allow signals to interrupt system calls</td>
</tr>
<tr>
<td>signal(3c)</td>
<td>simplified software signal facilities</td>
</tr>
<tr>
<td>signal(3f)</td>
<td>change</td>
</tr>
<tr>
<td>sin(3m)</td>
<td>trigonometric</td>
</tr>
<tr>
<td>sinh(3m)</td>
<td>hyperbolic</td>
</tr>
<tr>
<td>sleep(3f)</td>
<td>suspend</td>
</tr>
<tr>
<td>sleep(3)</td>
<td>suspend execution for interval</td>
</tr>
<tr>
<td>sqrt(3m)</td>
<td>cube</td>
</tr>
<tr>
<td>standard(3)</td>
<td>VADS standard library</td>
</tr>
<tr>
<td>stat(3f)</td>
<td>get</td>
</tr>
<tr>
<td>staux(3)</td>
<td>routines</td>
</tr>
<tr>
<td>stcu(3)</td>
<td>routines</td>
</tr>
<tr>
<td>stdio(3s)</td>
<td>standard buffered input, output package</td>
</tr>
<tr>
<td>stfd(3)</td>
<td>routines</td>
</tr>
<tr>
<td>stfe(3)</td>
<td>routines</td>
</tr>
<tr>
<td>stio(3)</td>
<td>routines</td>
</tr>
<tr>
<td>strprint(3)</td>
<td>routines to print the symbol table</td>
</tr>
<tr>
<td>string(3)</td>
<td>string operations</td>
</tr>
<tr>
<td>tty(3c)</td>
<td>stty,</td>
</tr>
<tr>
<td>swab(3)</td>
<td>swap bytes</td>
</tr>
<tr>
<td>syslog(3)</td>
<td>control system bytes</td>
</tr>
<tr>
<td>system(3)</td>
<td>issue a shell command</td>
</tr>
<tr>
<td>system(3f)</td>
<td>execute</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>termcap(3x)</td>
<td>terminal independent</td>
</tr>
<tr>
<td>time(3c)</td>
<td>get date and time</td>
</tr>
<tr>
<td>times(3c)</td>
<td>get process times</td>
</tr>
<tr>
<td>timezone(3)</td>
<td>supply timezone string</td>
</tr>
<tr>
<td>tbyname(3)</td>
<td>find name of a terminal</td>
</tr>
<tr>
<td>ualarm(3)</td>
<td>schedule signal after specified time</td>
</tr>
<tr>
<td>unaligned(3)</td>
<td>gather statistics on unaligned references</td>
</tr>
<tr>
<td>ungetc(3s)</td>
<td>push character back into input stream</td>
</tr>
<tr>
<td>unlink(3f)</td>
<td>remove</td>
</tr>
<tr>
<td>usleep(3)</td>
<td>suspend execution for interval</td>
</tr>
<tr>
<td>utime(3c)</td>
<td>set file times</td>
</tr>
<tr>
<td>valloc(3c)</td>
<td>aligned memory allocator</td>
</tr>
<tr>
<td>varargs(3)</td>
<td>variable argument list</td>
</tr>
<tr>
<td>verdivxlib(3)</td>
<td>MIPS-supported Ada library packages</td>
</tr>
<tr>
<td>vlimit(3c)</td>
<td>control maximum system resource consumption</td>
</tr>
<tr>
<td>vtimes(3c)</td>
<td>get information about resource utilization</td>
</tr>
<tr>
<td>ypcInit(3n)</td>
<td>yellow pages client interface</td>
</tr>
<tr>
<td>yp passwd(3r)</td>
<td>update user password in yellow pages</td>
</tr>
</tbody>
</table>

5. Miscellaneous

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acct(5)</td>
<td>execution accounting file</td>
</tr>
<tr>
<td>csconf(5)</td>
<td>startup file for csh command</td>
</tr>
<tr>
<td>printcap(5)</td>
<td>printer capability database</td>
</tr>
<tr>
<td>termcap(5)</td>
<td>terminal capability data base</td>
</tr>
<tr>
<td>gmtime, asctime, timezone, tset</td>
<td>convert date and time to ASCII</td>
</tr>
<tr>
<td>crypt, setkey, encrypt</td>
<td>DES encryption</td>
</tr>
<tr>
<td>ether_hoston, ether_line</td>
<td>Ethernet address mapping</td>
</tr>
<tr>
<td>absolute value hypot, fabs</td>
<td>Euclidean distance, complex</td>
</tr>
<tr>
<td>inet_ifaddrs, inet_netid</td>
<td>Internet inet_makeaddr</td>
</tr>
<tr>
<td>emulate_branch</td>
<td>MIPS branch emulation</td>
</tr>
<tr>
<td>emulate_branch</td>
<td>MIPS branch emulation</td>
</tr>
<tr>
<td>packages_visibility</td>
<td>MIPS-supported Ada library</td>
</tr>
<tr>
<td>nssvc, async_daemon</td>
<td>NFS daemons</td>
</tr>
<tr>
<td>standard</td>
<td>VADS standard library</td>
</tr>
<tr>
<td>conversion ns_addr, ns_nameto</td>
<td>Xerox NS(tm) address</td>
</tr>
<tr>
<td>and fabs, floor, ceil, rint</td>
<td>absolute value, floor, ceiling</td>
</tr>
<tr>
<td>accept</td>
<td>accept a connection on a socket</td>
</tr>
<tr>
<td>ranhshinit, ranhash, ranlookup</td>
<td>access routine for the symbol</td>
</tr>
<tr>
<td>interleaved swapoff</td>
<td>add a swap device for</td>
</tr>
<tr>
<td>files lockf</td>
<td>advisory record locking on</td>
</tr>
<tr>
<td>valloc</td>
<td>aligned memory allocator</td>
</tr>
<tr>
<td>system calls siginterrupt</td>
<td>allow signals to interrupt</td>
</tr>
<tr>
<td>lock on an open file flock</td>
<td>apply or remove an advisory</td>
</tr>
<tr>
<td>output a.out</td>
<td>assembler and link editor</td>
</tr>
<tr>
<td>setbuf, setbuffer, setlinebuf</td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>signals and wait for sigpause</td>
<td>atomically release blocked</td>
</tr>
<tr>
<td>j0, j1, jn, y0, y1, yn</td>
<td>bessel functions</td>
</tr>
<tr>
<td>random, initstate, setstate</td>
<td>better random number generator;</td>
</tr>
<tr>
<td>bind</td>
<td>bind a name to a socket</td>
</tr>
<tr>
<td>bcopy, bcmip, bzero, file</td>
<td>bit and byte string operations</td>
</tr>
<tr>
<td>sigblock</td>
<td>block signals</td>
</tr>
<tr>
<td>fread, fwrite</td>
<td>buffered binary input, output</td>
</tr>
<tr>
<td>directory chdir</td>
<td>change current working directory</td>
</tr>
<tr>
<td>brk, sbrk</td>
<td>change data segment size</td>
</tr>
<tr>
<td>chdir</td>
<td>change default directory</td>
</tr>
<tr>
<td>chmod</td>
<td>change mode of a file</td>
</tr>
<tr>
<td>file chown</td>
<td>change owner and group of a</td>
</tr>
<tr>
<td>signal</td>
<td>change the action for a signal</td>
</tr>
<tr>
<td>rename</td>
<td>change the name of a file</td>
</tr>
<tr>
<td>toupper, tolower, toascii</td>
<td>character classification macros</td>
</tr>
<tr>
<td>values fp_class</td>
<td>classes of IEEE floating-point</td>
</tr>
<tr>
<td>values fp_class</td>
<td>classes of IEEE floating-point</td>
</tr>
<tr>
<td>ldldclose, ldaclose</td>
<td>close a common object file</td>
</tr>
<tr>
<td>fclose, flush</td>
<td>close or flush a stream</td>
</tr>
<tr>
<td>ioctl</td>
<td>control device</td>
</tr>
<tr>
<td>consumption getrlimit, setrlimit</td>
<td>control maximum system resource</td>
</tr>
<tr>
<td>consumption vlimit</td>
<td>control maximum system resource</td>
</tr>
<tr>
<td>openlog, closelog, setlogmask</td>
<td>control system log syslog</td>
</tr>
<tr>
<td>atof, atoi, atol</td>
<td>convert ASCII to numbers</td>
</tr>
<tr>
<td>htonl, htonl, ntohl, ntohs</td>
<td>convert values between host and</td>
</tr>
<tr>
<td>drem, finite, llog, scall</td>
<td>copy signs, remainder, copysign,</td>
</tr>
<tr>
<td>synchronization of the adjtime</td>
<td>correct the time to allow</td>
</tr>
<tr>
<td>fork</td>
<td>create a copy of this process</td>
</tr>
<tr>
<td>creat</td>
<td>create a new file</td>
</tr>
<tr>
<td>fork</td>
<td>create a new process</td>
</tr>
<tr>
<td>sockets socketpair</td>
<td>create a pair of connected</td>
</tr>
<tr>
<td>communication socket</td>
<td>create an endpoint for</td>
</tr>
<tr>
<td>communication channel pipe</td>
<td>create an interprocess</td>
</tr>
<tr>
<td>sqrt, sqrt</td>
<td>cube root, square root</td>
</tr>
<tr>
<td>store, delete, firstkey, nextkey</td>
<td>data base subroutines fetch,</td>
</tr>
<tr>
<td>dbm_error, dbm_clearerr</td>
<td>data base subroutines</td>
</tr>
<tr>
<td>close</td>
<td>delete a descriptor</td>
</tr>
<tr>
<td>file access</td>
<td>determine accessibility of a</td>
</tr>
<tr>
<td>file access</td>
<td>determine accessibility of a</td>
</tr>
<tr>
<td>file access</td>
<td>determine accessibility of file</td>
</tr>
<tr>
<td>seekdir, rewinddir, closedir</td>
<td>directory operations telldir,</td>
</tr>
<tr>
<td>and print the disassembler</td>
<td>disassemble a MIPS instruction</td>
</tr>
<tr>
<td>dup, dup2</td>
<td>duplicate a descriptor</td>
</tr>
<tr>
<td>system setquota</td>
<td>enable, disable quotas on a file</td>
</tr>
<tr>
<td>floating-point unit mipsfpu</td>
<td>enabling and disabling the</td>
</tr>
<tr>
<td>erf, erfc</td>
<td>error functions</td>
</tr>
<tr>
<td>system</td>
<td>execute a UNIX command</td>
</tr>
<tr>
<td>exec, execve, exact, environ</td>
<td>execute a file execvp,</td>
</tr>
<tr>
<td>execute</td>
<td>execute a file</td>
</tr>
<tr>
<td>specified time alarm</td>
<td>execute a subroutine after a</td>
</tr>
<tr>
<td>Permutated Index</td>
<td>RISC/os Programmer's Reference</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>space, closepl</td>
<td>graphics interface</td>
</tr>
<tr>
<td>sinh, cosh, tanh</td>
<td>hyperbolic functions</td>
</tr>
<tr>
<td>machines rquota</td>
<td>implement quotas on remote</td>
</tr>
<tr>
<td>syscall</td>
<td>indirect system call</td>
</tr>
<tr>
<td>initgroups</td>
<td>initialize group access list</td>
</tr>
<tr>
<td>popen, pclose</td>
<td>initiate I/O to/from a process</td>
</tr>
<tr>
<td>socket connect</td>
<td>initiate a connection on a</td>
</tr>
<tr>
<td>queue insue, remque</td>
<td>insert/remove element from a</td>
</tr>
<tr>
<td>abs</td>
<td>integer absolute value</td>
</tr>
<tr>
<td>functions intro</td>
<td>introduction to C library</td>
</tr>
<tr>
<td>functions intro</td>
<td>introduction to FORTRAN library</td>
</tr>
<tr>
<td>library functions</td>
<td>math introduction to mathematical</td>
</tr>
<tr>
<td>and error numbers intro</td>
<td>introduction to system calls</td>
</tr>
<tr>
<td>asinh, acosh, atanh</td>
<td>inverse hyperbolic functions</td>
</tr>
<tr>
<td>system</td>
<td>issue a shell command</td>
</tr>
<tr>
<td>file systems mount</td>
<td>keep track of remotely mounted</td>
</tr>
<tr>
<td>end, etext, edata</td>
<td>last locations in program</td>
</tr>
<tr>
<td>end, etext, edata</td>
<td>last locations in program</td>
</tr>
<tr>
<td>examples</td>
<td>library of sample programs</td>
</tr>
<tr>
<td>data representation xdr</td>
<td>library routines for external</td>
</tr>
<tr>
<td>procedure calls rpc</td>
<td>library routines for remote</td>
</tr>
<tr>
<td>socket listen</td>
<td>listen for connections on a</td>
</tr>
<tr>
<td>gamma</td>
<td>log gamma function</td>
</tr>
<tr>
<td>mkdir</td>
<td>make a directory file</td>
</tr>
<tr>
<td>link</td>
<td>make a hard link to a file</td>
</tr>
<tr>
<td>link</td>
<td>make a link to an existing file</td>
</tr>
<tr>
<td>mkodn</td>
<td>make a special file</td>
</tr>
<tr>
<td>mktmp</td>
<td>make a unique file name</td>
</tr>
<tr>
<td>symlink</td>
<td>make symbolic link to a file</td>
</tr>
<tr>
<td>quota</td>
<td>manipulate disk quotas</td>
</tr>
<tr>
<td>quotactl</td>
<td>manipulate disk quotas</td>
</tr>
<tr>
<td>getenv, setenv, unsetenv</td>
<td>manipulate environmental</td>
</tr>
<tr>
<td>of a ldread, ldlimit, lditem</td>
<td>manipulate line number entries</td>
</tr>
<tr>
<td>mmap, munmap</td>
<td>map or unmap pages of memory</td>
</tr>
<tr>
<td>uncachable cachecl</td>
<td>mark pages cacheable or</td>
</tr>
<tr>
<td>free, realloc, calloc</td>
<td>memory allocator malloc,</td>
</tr>
<tr>
<td>memchr, memcmp, memcpy, memset</td>
<td>memory operations memcmp,</td>
</tr>
<tr>
<td>mount</td>
<td>mount file system</td>
</tr>
<tr>
<td>lseek</td>
<td>move read, write pointer</td>
</tr>
<tr>
<td>arithmetic</td>
<td>multiple precision integer</td>
</tr>
<tr>
<td>setjmp, longjmp</td>
<td>non-local goto</td>
</tr>
<tr>
<td>reading fdopen, ldopen</td>
<td>open a common object file for</td>
</tr>
<tr>
<td>writing, or create a new open</td>
<td>open a file for reading or</td>
</tr>
<tr>
<td>fopend, freopen, fdopen</td>
<td>open a stream</td>
</tr>
<tr>
<td>ecvt, vct, cvt</td>
<td>output conversion</td>
</tr>
<tr>
<td>monitor, monstartup, moncontrol</td>
<td>prepare execution profile</td>
</tr>
<tr>
<td>printcap</td>
<td>printer capability database</td>
</tr>
<tr>
<td>ptrace</td>
<td>process trace</td>
</tr>
<tr>
<td>assert</td>
<td>program verification</td>
</tr>
<tr>
<td>in Ada publiclib</td>
<td>public domain packages written</td>
</tr>
<tr>
<td>stream ungetc</td>
<td>push character back into input</td>
</tr>
<tr>
<td>puts, fput, fputc</td>
<td>put a string on a stream</td>
</tr>
<tr>
<td>putc, putc, putwchar</td>
<td>put character or word on a</td>
</tr>
<tr>
<td>quicksort</td>
<td>quick sort</td>
</tr>
<tr>
<td>qsort</td>
<td>quickier sort</td>
</tr>
<tr>
<td>rand, srand</td>
<td>random number generator</td>
</tr>
<tr>
<td>rand, irand, srand</td>
<td>random number generator</td>
</tr>
<tr>
<td>getpass</td>
<td>read a password</td>
</tr>
<tr>
<td>entry of a common ldthread</td>
<td>read an indexed symbol table</td>
</tr>
<tr>
<td>header of a ldthread, ldshread</td>
<td>read an indexed section</td>
</tr>
<tr>
<td>read, readv</td>
<td>read input</td>
</tr>
<tr>
<td>member of an archive ldthread</td>
<td>read the archive header of a</td>
</tr>
<tr>
<td>common object file ldthread</td>
<td>read the file header of a</td>
</tr>
<tr>
<td>reallink</td>
<td>read value of a symbolic link</td>
</tr>
<tr>
<td>reboot</td>
<td>reboot system or halt processor</td>
</tr>
<tr>
<td>recv, recvfrom, rcvmsg</td>
<td>receive a message from a socket</td>
</tr>
<tr>
<td>re_exec, re_exec</td>
<td>regular expression handler</td>
</tr>
<tr>
<td>re_-exec</td>
<td>remote execution protocol</td>
</tr>
<tr>
<td>unlink</td>
<td>remove a directory entry</td>
</tr>
<tr>
<td>rmfile</td>
<td>remove a file system</td>
</tr>
<tr>
<td>umount</td>
<td>remove a file system</td>
</tr>
<tr>
<td>unlink</td>
<td>remove directory entry</td>
</tr>
<tr>
<td>unit fseek, ftell</td>
<td>reposition a file on a logical</td>
</tr>
<tr>
<td>fseek, ftell, rewind</td>
<td>reposition a stream</td>
</tr>
<tr>
<td>res_init, dn_comp, dn_expand</td>
<td>resolver routines res_send,</td>
</tr>
<tr>
<td>Procedure</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>ldgetaux</td>
<td>retrieve an auxiliary entry</td>
</tr>
<tr>
<td>ldgetpd</td>
<td>retrieve procedure descriptor</td>
</tr>
<tr>
<td>ldgetname</td>
<td>retrieve symbol name for object</td>
</tr>
<tr>
<td>getarg, larg</td>
<td>return command line arguments</td>
</tr>
<tr>
<td>ASCII string</td>
<td>return date and time in an</td>
</tr>
<tr>
<td>fd</td>
<td>routines for returning a stream</td>
</tr>
<tr>
<td>idate, ttime</td>
<td>return date or time in</td>
</tr>
<tr>
<td>etime, dtime</td>
<td>return elapsed execution time</td>
</tr>
<tr>
<td>sigreturn</td>
<td>return from signal</td>
</tr>
<tr>
<td>on remote</td>
<td>return information about users</td>
</tr>
<tr>
<td>rusers</td>
<td>return length of Fortran string</td>
</tr>
<tr>
<td>len</td>
<td>return system time</td>
</tr>
<tr>
<td>rexec</td>
<td>return stream to a remote</td>
</tr>
<tr>
<td>loc</td>
<td>return the address of an object</td>
</tr>
<tr>
<td>routines for returning a stream</td>
<td>routines that provide a binary</td>
</tr>
<tr>
<td>routines that provide a</td>
<td>routines that provide access to</td>
</tr>
<tr>
<td>routines that provide scalar</td>
<td>routines that provide symbol table</td>
</tr>
<tr>
<td>routines to print the symbol table</td>
<td>scan a directory</td>
</tr>
<tr>
<td>scandir, alphasort</td>
<td>schedule signal after specified</td>
</tr>
<tr>
<td>time, alarm</td>
<td>schedule signal after specified</td>
</tr>
<tr>
<td>time, ualarm</td>
<td>“optimal” cursor motion cursors</td>
</tr>
<tr>
<td>of a common</td>
<td>seek to a indexednamed section</td>
</tr>
<tr>
<td>Idseek, Ldseek</td>
<td>seek to line number entries of</td>
</tr>
<tr>
<td>section of a</td>
<td>seek to relocation entries of</td>
</tr>
<tr>
<td>Ldseek, Ldnsseek</td>
<td>seek to the optional file</td>
</tr>
<tr>
<td>header of a common</td>
<td>seek to the symbol table of a</td>
</tr>
<tr>
<td>Idhseek</td>
<td>send a message from a socket</td>
</tr>
<tr>
<td>send, sendto, sendmsg</td>
<td>send a signal to a process</td>
</tr>
<tr>
<td>kill</td>
<td>send signal to a process</td>
</tr>
<tr>
<td>kill</td>
<td>send signal to a process group</td>
</tr>
<tr>
<td>killpg</td>
<td>send signal to a process group</td>
</tr>
<tr>
<td>(defunct) stty, gty</td>
<td>set and get terminal state</td>
</tr>
<tr>
<td>context sigstack</td>
<td>set and, or get signal stack</td>
</tr>
<tr>
<td>sigsetmask, sigmask</td>
<td>set current signal mask</td>
</tr>
<tr>
<td>umask</td>
<td>set file creation mode mask</td>
</tr>
<tr>
<td>utime</td>
<td>set file times</td>
</tr>
<tr>
<td>utimes</td>
<td>set file times</td>
</tr>
<tr>
<td>setgroups</td>
<td>seek group access list</td>
</tr>
<tr>
<td>setppr</td>
<td>set group process group</td>
</tr>
<tr>
<td>nice</td>
<td>set program priority</td>
</tr>
<tr>
<td>setregid</td>
<td>set real and effective group ID</td>
</tr>
<tr>
<td>ID’s</td>
<td>set real and effective user</td>
</tr>
<tr>
<td>setuid, setgid, setegid, setregid</td>
<td>set user and group ID setuid, setregid and setegid</td>
</tr>
<tr>
<td>connection shutdown</td>
<td>shut down part of a full-duplex</td>
</tr>
<tr>
<td>facilities signal</td>
<td>simplified software signal</td>
</tr>
<tr>
<td>sigvec</td>
<td>software signal facilities</td>
</tr>
<tr>
<td>memory efficient way</td>
<td>spawn new process in a virtual</td>
</tr>
<tr>
<td>vfork</td>
<td>split into manilla and</td>
</tr>
<tr>
<td>exponent frexp, ldexp, modf</td>
<td>standard buffered input, output</td>
</tr>
<tr>
<td>package stdio</td>
<td>startup file for csh command</td>
</tr>
<tr>
<td>cshrc</td>
<td>stop until signal</td>
</tr>
<tr>
<td>pause</td>
<td>string operations strchr,</td>
</tr>
<tr>
<td>stprbrk, strspn, streqstr, strtok</td>
<td>subroutines for the HP 2460</td>
</tr>
<tr>
<td>graphics terminal lib2648</td>
<td>supply timezone string</td>
</tr>
<tr>
<td>time</td>
<td>sleep</td>
</tr>
<tr>
<td>interval sleep</td>
<td>suspend execution for an</td>
</tr>
<tr>
<td>sleep</td>
<td>suspend execution for interval</td>
</tr>
<tr>
<td>usleep</td>
<td>suspend execution for interval</td>
</tr>
<tr>
<td>swab</td>
<td>swap bytes</td>
</tr>
<tr>
<td>state with that on disk fsync</td>
<td>synchronize a file's in-core</td>
</tr>
<tr>
<td>select</td>
<td>synchronous I/O multiplexing</td>
</tr>
<tr>
<td>for</td>
<td>system error messages</td>
</tr>
<tr>
<td>sys_errno, sys_nerr</td>
<td>system signal messages</td>
</tr>
<tr>
<td>psignal, sys_siglist</td>
<td>terminal capability data base</td>
</tr>
<tr>
<td>termcap</td>
<td>terminal independent timebase</td>
</tr>
<tr>
<td>tgetflag, tgetstr, tgoto, tputs</td>
<td>terminate Fortran program</td>
</tr>
<tr>
<td>abort</td>
<td>terminate a process</td>
</tr>
<tr>
<td>flushing any pending output</td>
<td>terminate a process after</td>
</tr>
<tr>
<td>exit2</td>
<td>trigonometric functions sin,</td>
</tr>
<tr>
<td>cos, tan, asin, acos, atan, atan2</td>
<td>turn accounting on or off</td>
</tr>
<tr>
<td>acct</td>
<td>update super-block</td>
</tr>
</tbody>
</table>
get file, getdbyte, setdbyte, n_sys
ns_addr, ns_nsto : Xerox NS(tm)
references) fixdixe : fix
ether_line : Ethernet
loc : return the
allow synchronization of the
flock : apply or remove an
lockf
after a specified time
specified time
alarm : execute a subroutine
valloc :
malloc, free, realloc, calloc,
realloc, calloc, alloca : memory
valloc : aligned memory
calls siginterrupt:
adjtime : correct the time to
scandir
fpi : floating-point interrupt
fpi : floating-point interrupt
analysis
: flush contents of instruction
and, or data cache cacheflush
sigstack : set
output
on an open file
archive header of a member of an
archive file: lddread : read the
varargs : variable
large : return command line
getopt : get option letter from
: multiple precision integer
time, localtime, gmtime
trigonometric sin, cos, tan,
hyperbolic functions
a.out
setbuf, setbuffer, setlinebuf :
nfsvc
sin, cos, tan, asin, acos
sin, cos, tan, asin, acos, atan
functions asinh, acosh,
to numbers
numbers atof, atof, atoi
signals and wait for sigpause:
that provide scalar interfaces to
idgex : retrieve an
: terminal capability data
delete, firstkey, nextkey : data
db_merr, dbm_clearrer : data
provide a high-level interface to
string operations bcopy, byte string operations
j0, j1, jn, y0, y1, yn :
random, initstate, setstate :
read, fwrite : buffered
stio : routines that provide a
bind :
bcopy, bcmp, bzero, ffs :
sigblock
sigpause : atomically release
emulate_branch : MIPS
emulate_branch : MIPS
size
fread, fwrite
stdio : standard
setbuffer, setlinebuf : assign
values between host and network
bcopy, bcmp, bzero, ffs : bit and
swab : swap
operations bcopy, bcmp,
complex absolute value hypot,
of instruction and, or data
check : mark pages
or uncacheable
instruction and, or data cache
syscall : indirect system

getdbyte(3)
ns(3n)
fixdixe(2)
ethers(3n)
loc(3f)
adjtime(2)
flock(2)
lockf(3)
alarm(3f)
alarm(3c)
valloc(3c)
malloc(3)
malloc(3)
malloc(3)
malloc(3)
malloc(3)
siginterrupt(3)
adjtime(2)
scandir(3)
fpi(3)
fp(3)
sigstack(2)
a.out(4)
apply or remove an advisory lock
archive file: lddread : read the
archive header of a member of an
gpart : arguments getarg
argv : argument list
math : arithmetic
asctime, time zone, tzset :
asin, acos, atan, atan2 :
asinh, acosh, atanh : inverse
asinh, acosh
assert : program verification
assign buffering to a stream
asyncdaemon : NFS daemons
atan, atan2 : trigonometric
atan2 : trigonometric functions
atanh : inverse hyperbolic
asinh, acosh
atof, atoi, atol : convert ASCII to
atof, atoi, atol : convert ASCII to
atomic release blocked
auxiliary entry, given an index
termcap
base subroutines fetch, store,
base subroutines dbm_nextkey,
basic functions needed that
bcopy, bcmp, bzero, ffs : bit and byte
bcopy, bcmp, bzero, ffs : bit and byte
better random number generator;
binary input, output
binary read write interface to
bind : bind a name to a socket
bind : bind a name to a socket
bit and byte string operations
block signals
block signals and wait for
branch emulation
branch emulation
byte order ntohs : convert
byte string operations
buffered binary input, output
buffered binary input, output package
buffering to a stream setbuf,
byte order ntohs : convert
byte string operations
bzero, ffs : bit and byte string

cabs : Euclidean distance

cache : flush contents

cache : flush contents

cache flush contents of

calls bcopy, bcmp,
complex absolute value hypot,
of instruction and, or data
check : mark pages
or uncacheable
check : mark pages cacheable
check : mark pages cacheable
check : mark pages cacheable
check : mark pages cacheable
check flush contents of

Page xiv
February 1989
MIPS Computer Systems, Inc.
<table>
<thead>
<tr>
<th>RISC/OS Programmer's Reference</th>
<th>Permutated Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>callee getuid, getgid</td>
<td>getuid(3F)</td>
</tr>
<tr>
<td>malloc, free, realloc</td>
<td>malloc(3)</td>
</tr>
<tr>
<td>routines for remote procedure</td>
<td>rpc(3n)</td>
</tr>
<tr>
<td>allow signals to interrupt system</td>
<td>siginterrupt(3)</td>
</tr>
<tr>
<td>intro : introduction to system</td>
<td>intro(2)</td>
</tr>
<tr>
<td>termcap : terminal</td>
<td>termcap(5)</td>
</tr>
<tr>
<td>printcap : printer</td>
<td>printcap(5)</td>
</tr>
<tr>
<td>chdir, sqrt : cube root, square</td>
<td>sqrt(3m)</td>
</tr>
<tr>
<td>floor, ceiling, and fabs, floor,</td>
<td>floor(3m)</td>
</tr>
<tr>
<td>rint : absolute value, floor</td>
<td>rint(3m)</td>
</tr>
<tr>
<td>chdir : change current working directory</td>
<td>chdir(2)</td>
</tr>
<tr>
<td>brk, sbbrk : change data segment size</td>
<td>brk(2)</td>
</tr>
<tr>
<td>chmod : change mode of a file</td>
<td>chmod(3f)</td>
</tr>
<tr>
<td>chown : change owner and group of a file</td>
<td>chown(2)</td>
</tr>
<tr>
<td>signal : change the action for a signal</td>
<td>signal(3f)</td>
</tr>
<tr>
<td>rename : change the name of a file</td>
<td>rename(2)</td>
</tr>
<tr>
<td>an interprocess communication</td>
<td>pipe(2)</td>
</tr>
<tr>
<td>ungetc : push character back into input stream</td>
<td>ungetc(3s)</td>
</tr>
<tr>
<td>toupper, tolower, toascii : character classification macros</td>
<td>cctype(3)</td>
</tr>
<tr>
<td>getc, getchar, fgetc, getw : get a character from a logical unit</td>
<td>getc(3)</td>
</tr>
<tr>
<td>getc, fgetc, getf : get a character or word from stream</td>
<td>getf(3)</td>
</tr>
<tr>
<td>putc, putchar, fputc, putw : put a character or word on a stream</td>
<td>putw(3)</td>
</tr>
<tr>
<td>unit putc, fputc : write a character to a fortran logical</td>
<td>unit(3)</td>
</tr>
<tr>
<td>can not be longer than NCARGS:50</td>
<td>wait(3f)</td>
</tr>
<tr>
<td>directory : characters, as defined in</td>
<td>wait(3f)</td>
</tr>
<tr>
<td>chdir : change current working directory</td>
<td>chdir(2)</td>
</tr>
<tr>
<td>chmod : change default directory</td>
<td>chmod(3)</td>
</tr>
<tr>
<td>chmod : change mode of a file</td>
<td>chmod(3f)</td>
</tr>
<tr>
<td>chown : change owner and group of a file</td>
<td>chown(2)</td>
</tr>
<tr>
<td>values fp_class : classes of IEEE floating-point</td>
<td>values(3f)</td>
</tr>
<tr>
<td>tolower, toascii : character classification macros</td>
<td>toascii(3)</td>
</tr>
<tr>
<td>inquiries ferror, feco, yyprev : yellow pages</td>
<td>inquiries(3)</td>
</tr>
<tr>
<td>synchronization of the system clearerr, fileno : stream status</td>
<td>synchronization(3)</td>
</tr>
<tr>
<td>ldclose, ldaclose : close a clock : correct the time to allow</td>
<td>ldclose(3)</td>
</tr>
<tr>
<td>ldclose, ldaclose : close a common object file</td>
<td>ldclose(3)</td>
</tr>
<tr>
<td>fclos, flush : close or flush a stream</td>
<td>fclos(3)</td>
</tr>
<tr>
<td>telldir, seekdir, rewinddir, current directory operations</td>
<td>telldir(3)</td>
</tr>
<tr>
<td>system log syslog, openlog, closeopen : graphics interface</td>
<td>system(3)</td>
</tr>
<tr>
<td>syslog : 5120 space,</td>
<td>syslog(2)</td>
</tr>
<tr>
<td>cshrc : startup file for csh command</td>
<td>cshrc(3)</td>
</tr>
<tr>
<td>returning a stream to a remote command</td>
<td>returning(3)</td>
</tr>
<tr>
<td>reexec : return stream to a remote command</td>
<td>reexec(3)</td>
</tr>
<tr>
<td>system : issue a shell command</td>
<td>system(3)</td>
</tr>
<tr>
<td>system : execute a UNIX command</td>
<td>system(3)</td>
</tr>
<tr>
<td>getarg, larg : return common line arguments</td>
<td>getarg(3)</td>
</tr>
<tr>
<td>ldclose, ldaclose : close a common object file</td>
<td>ldclose(3)</td>
</tr>
<tr>
<td>ldopen, ldopen : open a common object file ldthread</td>
<td>ldopen(3x)</td>
</tr>
<tr>
<td>read the file header of a common object file seek</td>
<td>read(3x)</td>
</tr>
<tr>
<td>number entries of a section of a common object file to line</td>
<td>number(3x)</td>
</tr>
<tr>
<td>to the optional file header of a common object file to relocation</td>
<td>to(3x)</td>
</tr>
<tr>
<td>entries of a section of a common object file read</td>
<td>entries(3x)</td>
</tr>
<tr>
<td>indexed named section header of a common object file seek</td>
<td>indexed(3x)</td>
</tr>
<tr>
<td>to an indexed named section of a communication channel</td>
<td>communications(3)</td>
</tr>
<tr>
<td>indexed symbol table entry of a common object file function</td>
<td>indexed(3x)</td>
</tr>
<tr>
<td>: seek to the symbol table of a common object file ldthread</td>
<td>: seek(3x)</td>
</tr>
<tr>
<td>ldopen, ldopen : open a common object file ldthread</td>
<td>ldopen(3x)</td>
</tr>
<tr>
<td>line number entries of a common object file function</td>
<td>line(3x)</td>
</tr>
<tr>
<td>socket : create an endpoint for communication channel</td>
<td>socket(2)</td>
</tr>
<tr>
<td>pipe : create an interprocess communication compiler unit symbol table</td>
<td>pipe(2)</td>
</tr>
<tr>
<td>stouc(3)</td>
<td>stouc(3)</td>
</tr>
<tr>
<td>hypot, cabs : Euclidean distance, complex absolute value</td>
<td>hypot(3m)</td>
</tr>
<tr>
<td>hwconf : get or set hardware configuration information</td>
<td>hwconf(2)</td>
</tr>
<tr>
<td>on a socket connect : initiate a connection</td>
<td>connect(2)</td>
</tr>
<tr>
<td>getpeername : get name of connected peer</td>
<td>getpeername(2)</td>
</tr>
<tr>
<td>: getsockoptpair : create a pair of connected sockets</td>
<td>connected(2)</td>
</tr>
<tr>
<td>: shut down part of a full-duplex connection shutdown</td>
<td>connected(2)</td>
</tr>
<tr>
<td>accept : accept a connection on a socket</td>
<td>accept(2)</td>
</tr>
<tr>
<td>connect : initiate a connection on a socket</td>
<td>connect(2)</td>
</tr>
<tr>
<td>listen : listen for connections on a socket</td>
<td>listen(2)</td>
</tr>
<tr>
<td>: control maximum system resource</td>
<td>consumplimit(3)</td>
</tr>
<tr>
<td>: control maximum system resource getrlimit, setrlimit</td>
<td>consumplimit(3)</td>
</tr>
<tr>
<td>data cache cacheflush : flush</td>
<td>datacache(3)</td>
</tr>
<tr>
<td>fcntl : file context sigstack</td>
<td>fsni(2)</td>
</tr>
<tr>
<td>ioctl : control device</td>
<td>ioctl(2)</td>
</tr>
<tr>
<td>iocl : control</td>
<td>device : device for interleaved</td>
</tr>
<tr>
<td>swapon : add a swap</td>
<td>device :</td>
</tr>
<tr>
<td>chdir : change current working directory</td>
<td>device :</td>
</tr>
<tr>
<td>chdir : change default directory</td>
<td>device :</td>
</tr>
<tr>
<td>get current working directory</td>
<td>device :</td>
</tr>
<tr>
<td>scandir, alphasort : scan a directory</td>
<td>device :</td>
</tr>
<tr>
<td>getpathname of current working directory</td>
<td>device :</td>
</tr>
<tr>
<td>getdiskbyname : get directory entries in a filesystem</td>
<td>device :</td>
</tr>
<tr>
<td>un link : remove a directory entry</td>
<td>device :</td>
</tr>
<tr>
<td>mk dir : make a directory file</td>
<td>device :</td>
</tr>
<tr>
<td>rmdir : remove a directory file</td>
<td>device :</td>
</tr>
<tr>
<td>seekdir, rew indir, closedir : get directory operations telldir, get current working directory</td>
<td>device :</td>
</tr>
<tr>
<td>getwd : get current working directory</td>
<td>device :</td>
</tr>
<tr>
<td>and print the disassembler : instruction and print the file's in-core state that on getdiskbyname : get disk description by its name</td>
<td>device :</td>
</tr>
<tr>
<td>get quot a : manipulate disk quotes</td>
<td>device :</td>
</tr>
<tr>
<td>getquot acl : manipulate disk quotes</td>
<td>device :</td>
</tr>
<tr>
<td>unit mipsfp : enabling and disabling the floating-point</td>
<td>device :</td>
</tr>
<tr>
<td>hypot, cabs : Euclidean distance, complex absolute value</td>
<td>device :</td>
</tr>
<tr>
<td>res_nquery, res send, res init, res send, res_init, dn_comp, dn_exp : resolver routines</td>
<td>device :</td>
</tr>
<tr>
<td>in core name of current domain setdomainname</td>
<td>device :</td>
</tr>
<tr>
<td>publiclib : public domain packages written in Ada</td>
<td>device :</td>
</tr>
<tr>
<td>copy sign, remainder, copy sign, dtime : return elapsed execution time</td>
<td>device :</td>
</tr>
<tr>
<td>time etime, descriptor : get from a queue</td>
<td>device :</td>
</tr>
<tr>
<td>dup, dup2 : duplicate a a.out : assembler and link effective group ID</td>
<td>device :</td>
</tr>
<tr>
<td>setregid : set real and effective user ID's</td>
<td>device :</td>
</tr>
<tr>
<td>setreuid : set real and end, ete x, edata : last locations in program</td>
<td>device :</td>
</tr>
<tr>
<td>new process in a virtual memory</td>
<td>device :</td>
</tr>
<tr>
<td>efficient way vfork : spawn</td>
<td>device :</td>
</tr>
<tr>
<td>insque, remque : insert, remove emulation</td>
<td>device :</td>
</tr>
<tr>
<td>emulation : MIPS branch</td>
<td>device :</td>
</tr>
<tr>
<td>emulate_branch : MIPS branch</td>
<td>device :</td>
</tr>
<tr>
<td>new process in a virtual memory</td>
<td>device :</td>
</tr>
<tr>
<td>efficient way vfork : spawn</td>
<td>device :</td>
</tr>
<tr>
<td>set quota : enable, disable quotas on a file</td>
<td>device :</td>
</tr>
<tr>
<td>floating-point unit mipsfp : enabling and disabling the</td>
<td>device :</td>
</tr>
<tr>
<td>crypt, setkey, encrypt : DES encryption</td>
<td>device :</td>
</tr>
<tr>
<td>crypt, setkey, encrypt : DES encryption</td>
<td>device :</td>
</tr>
<tr>
<td>end, etex, edata : last locations in program</td>
<td>device :</td>
</tr>
<tr>
<td>getfsfil e, getf type, setfent, getfsent, getg rind, getg rnam, set grent, end grent, gethostent, gethostent, getmntent, getsetent, add ment, getnetbyname, set netent, getnetent, setnetent, set socket : create an</td>
<td>device :</td>
</tr>
<tr>
<td>getprotobyname, setprot enet, get names : set names</td>
<td>device :</td>
</tr>
<tr>
<td>getpwuid, getpwnam, setpwent, getg rnam, get setgrent, end grent, set netent, get tendrypt, gett ypsin, set t ypsin, endusershell, setusershell, nlist : get</td>
<td>device :</td>
</tr>
<tr>
<td>getdirentries : get directory</td>
<td>device :</td>
</tr>
<tr>
<td>ld litem : manipulate linear number</td>
<td>device :</td>
</tr>
<tr>
<td>ld lseek : seek to line number</td>
<td>device :</td>
</tr>
<tr>
<td>ldnrseek : seek to relocation</td>
<td>device :</td>
</tr>
<tr>
<td>end grent, get group file</td>
<td>device :</td>
</tr>
<tr>
<td>ins net : get network group</td>
<td>device :</td>
</tr>
<tr>
<td>get g pchynumber : get RPC</td>
<td>device :</td>
</tr>
<tr>
<td>endgrent, getttyid, get g rnam, getttyid, getg rnam, set grent, end grent, gett ypsin, set t ypsin, endusershell, setusershell, nlist : get</td>
<td>device :</td>
</tr>
<tr>
<td>getdirentries : get directory</td>
<td>device :</td>
</tr>
<tr>
<td>ld litem : manipulate linear number</td>
<td>device :</td>
</tr>
<tr>
<td>ld lseek : seek to line number</td>
<td>device :</td>
</tr>
<tr>
<td>ldnrseek : seek to relocation</td>
<td>device :</td>
</tr>
<tr>
<td>end grent, get group file</td>
<td>device :</td>
</tr>
<tr>
<td>ins net : get network group</td>
<td>device :</td>
</tr>
<tr>
<td>get g pchynumber : get RPC</td>
<td>device :</td>
</tr>
<tr>
<td>endgrent, getttyid, get g rnam, getttyid, getg rnam, set grent, end grent, gett ypsin, set t ypsin, endusershell, setusershell, nlist : get</td>
<td>device :</td>
</tr>
<tr>
<td>getdirentries : get directory</td>
<td>device :</td>
</tr>
<tr>
<td>ld litem : manipulate linear number</td>
<td>device :</td>
</tr>
<tr>
<td>ld lseek : seek to line number</td>
<td>device :</td>
</tr>
<tr>
<td>ldnrseek : seek to relocation</td>
<td>device :</td>
</tr>
<tr>
<td>end grent, get group file</td>
<td>device :</td>
</tr>
<tr>
<td>ins net : get network group</td>
<td>device :</td>
</tr>
<tr>
<td>get g pchynumber : get RPC</td>
<td>device :</td>
</tr>
<tr>
<td>endgrent, getttyid, get g rnam, getttyid, getg rnam, set grent, end grent, gett ypsin, set t ypsin, endusershell, setusershell, nlist : get</td>
<td>device :</td>
</tr>
<tr>
<td>getdirentries : get directory</td>
<td>device :</td>
</tr>
<tr>
<td>ld litem : manipulate linear number</td>
<td>device :</td>
</tr>
<tr>
<td>ld lseek : seek to line number</td>
<td>device :</td>
</tr>
<tr>
<td>ldnrseek : seek to relocation</td>
<td>device :</td>
</tr>
<tr>
<td>end grent, get group file</td>
<td>device :</td>
</tr>
<tr>
<td>ins net : get network group</td>
<td>device :</td>
</tr>
<tr>
<td>get g pchynumber : get RPC</td>
<td>device :</td>
</tr>
<tr>
<td>endgrent, getttyid, get g rnam, getttyid, getg rnam, set grent, end grent, gett ypsin, set t ypsin, endusershell, setusershell, nlist : get</td>
<td>device :</td>
</tr>
<tr>
<td>getdirentries : get directory</td>
<td>device :</td>
</tr>
<tr>
<td>ld litem : manipulate linear number</td>
<td>device :</td>
</tr>
<tr>
<td>ld lseek : seek to line number</td>
<td>device :</td>
</tr>
<tr>
<td>ldnrseek : seek to relocation</td>
<td>device :</td>
</tr>
<tr>
<td>end grent, get group file</td>
<td>device :</td>
</tr>
<tr>
<td>ins net : get network group</td>
<td>device :</td>
</tr>
<tr>
<td>get g pchynumber : get RPC</td>
<td>device :</td>
</tr>
<tr>
<td>endgrent, getttyid, get g rnam, getttyid, getg rnam, set grent, end grent, gett ypsin, set t ypsin, endusershell, setusershell, nlist : get</td>
<td>device :</td>
</tr>
<tr>
<td>getdirentries : get directory</td>
<td>device :</td>
</tr>
</tbody>
</table>
table entry of a common object
table of a common object file
setusershell, endusershell : get
string : truncate a file to a specified
len : return length of Fortran
getopt : get option

2648 graphics terminal
VADS libraries : overview of VADS
libraries VADS
standard : VADS standard
library functions
library functions math
library of sample programs
examples
verdixib : MIPS-supported Ada
library packages
data representation xdr :
procedure calls rpc :
getarg, large : return command
ldlinit, ldliiten : manipulate of a ldlseek, ldlseek : seek to
:
read value of a symbolic
link : readlink
link : make a hard link to a file
link : make a link to an existing
a.out : assembler and
link : make a hard
link to a file
link to a file
link to an existing file
symlink : make symbolic
link : make a
getgroups : get group access
initialize group access
nlist : get entries from name
setgroups : set group access
setvargs : variable argument
on a socket
listen : listen for connections
object
timezone, tzset : clime,
end, etcext, edata : last
exp, expm1, log
log10, log10p, pow : exponential,
log : scalar : compute
flush : flush output to a
fseek : reposition a file on a
fgetc : get a character from a
fgetc : get a character to a Fortran
fget : get user's
getlogin : get

to terminate system(3) can not be
setjmp,

stat,
time, time, clime,
information about users on remote
implement quotas on remote
rwall : write to specified remote
: character classification
alloca : memory allocator
quote : quotactl
getenv, setenv, unsetenv :
a ldlread, ldlinit, ldliiten :
fread, ldexp, modf : split into
mmap, munmap :
ether_line : Ethernet address
uncacheable cachectl :

given an index ldgetaux
  retrieve an auxiliary entry
  ldgetaux(3x)

given a procedure ldgetpd
  retrieve procedure descriptor
  ldgetpd(3x)

file ldgetname
  retrieve symbol name for object
  ldgetname(3x)

getarg, iargc
  return command line arguments
  getarg(3f)

string fdate
  return date and time in an ASCII
  fdate(3f)

form idate, itime
  return date or time in numerical
  idate(3f)

etime, dtime
  return elapsed execution time
  etime(3f)

remote rusers, rusers:
  return information about users
  rusers(3f)

len
  return length of Fortran string
  len(3f)

rexec
  return stream to a remote command
  rexec(3c)

time, ctime, ltime, gtime:
  return system time
  time(3f)

loc
  return the address of an object
  loc(3f)

rcmd
  return the address of a remote
  rcmd(3)

reviser, ruserok : routines for
  returning a stream to a remote
  ruserok(3)

by the super-user(user:
  returning a stream to a remote
  user(3)

seek, tell,
  rewind, reposition a stream
  seek(3e)

readdir, telldir, seekdir,
  rewinddir, closedir : directory
  readdir(3)

command
  rcmd(3)

strncpy, strlcn, index, strchr,
  ceiling, and fabs, floor, ceil,
  rmdir : remove a directory file
  string(3)

information about users on
  rmdir(3)

chrt, sqtr : cube root, square
  square

sqrt, sqtr : cube
  sqrt(3m)

ranhash, ranlookup : access
  rand(3)

Xerox NLS(um) address conversion
  runhash(3)

dncomp, dnexpand : resolver
  ranlookup(3)

representation xdr : library
  routines for the symbol table
  xdr(3)

calls rpc : library
  routines for external data
  rpc(3)

to a rcmd, reviser, ruserok:
  routines for a remote procedure
  rcmd(3)

read write interface to stio:
  routines that provide a binary
  stio(3)

compilation unit symbol stcu:
  routines that provide a
  stcu(3)

high-level interface to stfe
  routines that provide a
  stfe(3)

per file descriptor stdf : routines that provide access to
  routines that provide scalar
  stdf(3)

interfaces to staux
  routines that provide access
to
  staux(3)

table stprint:
  routines to print the symbol
  stprint(3)

procedure calls
  routine for the symbol table
  rpc(3)

remote machines
  routines for returning a stream
  rpc(3)

returning a stream to a rcmd
  routines that provide a binary
  rcmd(3)

...procedure_string_table:
  routines that provide a
  rcmd(3)

...procedure_string_table:
  routines that provide a
  rcmd(3)

...procedure_string_table:
  routines that provide a
  rcmd(3)

...procedure_string_table:
  routines that provide a
  rcmd(3)

a stream to a rcmd, reviser,
  routines for returning
  rcmd(3)

users on remote rusers,
  return information about
  rcmd(3)

machines
  sample programs
  rcmd(3)

examples : library of
  examples(3)

brk
  brk(2)

staux : routines that provide
  staux(3)

copy sign, drem, finite, logb,
  ieee(3m)

scandir, alphasort : scan a
directory
  scandir(3)

input conversion
  scanf, fscanf, sscanf : formatted
  scanf(3e)

time alarm
  scanf(3e)

time ualarm
  scanf(3e)

setpriority : get, set program
  scanf(3e)

cursor motion curses
  scanf(3e)

ldshread : read an indexednamed
  ldshread(3x)

seek to line number entries of a
  ldshread(3x)

: seek to relocation entries of a
  ldshread(3x)

: seek to an indexednamed
  ldshread(3x)

: seek to a common object
  ldshread(3x)

section of a common object
  ldshread(3x)

section of a common object
  ldshread(3x)

section of a that provide
  ldshread(3x)

seek to an indexednamed section
  ldshread(3x)

seek to line number entries of a
  ldshread(3x)

seek to relocation entries of a
  ldshread(3x)

seek to the optional field header
  ldshread(3x)

seek to the symbol table of a
  ldshread(3x)

seekdir, rewinddir, closedir:
  directory(3)

segment size
  brk(2)

multiplexing
  select : synchronous I/O
  select(2)

send, sendto, sendmsg :
  send a message from a socket
  send(2)

kill
  send signal to a process
  send(2)

message from a socket
  send signal to a process
  send(2)

kill
  send signal to a process group
  send(2)

killpg
  sendmsg : send a message from a
  send(2)

sendto, sendmsg : send a message
  send(2)
and disabling the floating-point
   a character to a totrax logical
   that provide a compilation
unit mpquot, fquot : enabling
    unit symbol table interface
    unit float output
    unlink : remove a directory entry
    unlink : remove directory entry
    umount : remove a file system
    unqun : manipulate
    until signal
    update super-block
    update user password in yellow pages
    user ID’s
    user and group ID setuid,
    user identity
    user or group ID of the caller
    user password in yellow pages
    user shell’s setusershell
    user’s login name
    users on remote machines
    usleep : suspend execution for
    utilization getusage
    utilization vtimes
    utime : set file times
    utimes : set file times
    valloc : aligned memory allocator
    value
    value hypot, cabs : Euclidean
    value, floor, ceiling, and fabs
    value of a symbolic link
    value of environment variables
    value of interval timer
    values fp_class
    values fp_class
    values between host and network
    wait : wait for a process to
terminatesystem(3f) can not be
   terminatesystem(3f) can not wait
    release blocked signals and
    wait, wait3 : terminate
    terminate
    terminate wait,
    fgetc, getw : get character or
    fputc, putw : put character or
    chdir : change current
getcwd : get path name of current
getwd : get current
logical unit pout, fputc :
   that provide a binary read
   write, writew
   machines rwall :
   write,
   open : open a file for reading or
   public domain packages
   external data representation
    j0, j1, jn, j0,
    j0, j1, jn, j0,
    : update user password in
    ypper_string, ypprot_err :
    y0, y1, yn : bessel functions
    y0, y1, yn : bessel functions
    yellow pages yppassword
    yellow pages client interface
    yc , yc functions
    yp_all, yp_order, yp_match,
    yp_all, yp_order, yp_match,
    yp_all, yp_match,
    yp_all, yp_match
    yp_first, yp_next, yp_all,
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yp_unbind, yp_match, ypcint,</td>
<td></td>
</tr>
<tr>
<td>yp_next, yp_all, yp_order,</td>
<td></td>
</tr>
<tr>
<td>yp_all, yp_bind, yp_unbind,</td>
<td></td>
</tr>
<tr>
<td>yp_unbind, yp_match, yp_first,</td>
<td></td>
</tr>
<tr>
<td>yp_first, yp_next, yp_all,</td>
<td>in yellow pages</td>
</tr>
<tr>
<td>yp_master, yperr_string,</td>
<td></td>
</tr>
<tr>
<td>yp_match, yp_first, yp_next,</td>
<td></td>
</tr>
<tr>
<td>yp_next, yp_all, yp_order,</td>
<td></td>
</tr>
<tr>
<td>yp_order, yp_master, yp_match,</td>
<td></td>
</tr>
<tr>
<td>yp_get_default_domain, yp_bind,</td>
<td></td>
</tr>
<tr>
<td>yp_unbind, yp_match, yp_first,</td>
<td></td>
</tr>
<tr>
<td>yppasswd : update user password</td>
<td></td>
</tr>
<tr>
<td>ypprot_err : yellow pages client</td>
<td></td>
</tr>
<tr>
<td>ypcint(3n)</td>
<td></td>
</tr>
</tbody>
</table>
NAME
accept — accept a connection on a socket

SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>

ns = accept(s, addr, addrlen)
int ns, s;
struct sockaddr *addr;
int *addrlen;

DESCRIPTION
The argument s is a socket that has been created with socket(2), bound to an address with bind(2), and is listening for connections after a listen(2). accept extracts the first connection on the queue of pending connections, creates a new socket with the same properties of s and allocates a new file descriptor, ns, for the socket. If no pending connections are present on the queue, and the socket is not marked as non-blocking, accept blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, accept returns an error as described below. The accepted socket, ns, may not be used to accept more connections. The original socket s remains open.

The argument addr is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the addr parameter is determined by the domain in which the communication is occurring. The addrlen parameter is a value-result parameter; it should initially contain the amount of space pointed to by addr; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types, currently with SOCK_STREAM.

It is possible to select(2) a socket for the purposes of doing an accept by selecting it for read.

RETURN VALUE
The call returns −1 on error. If it succeeds, it returns a non-negative integer that is a descriptor for the accepted socket.

ERRORS
The accept will fail if:
[EBAADF] The descriptor is invalid.
[ENOTSOCK] The descriptor references a file, not a socket.
[EOPNOTSUPP] The referenced socket is not of type SOCK_STREAM.
[EFAULT] The addr parameter is not in a writable part of the user address space.
[EWOULDBLOCK] The socket is marked non-blocking and no connections are present to be accepted.

SEE ALSO
bind(2), connect(2), listen(2), select(2), socket(2)
NAME
access — determine accessibility of file

SYNOPSIS
#include <sys/file.h>
#define R_OK 4 /* test for read permission */
#define W_OK 2 /* test for write permission */
#define X_OK 1 /* test for execute (search) permission */
#define F_OK 0 /* test for presence of file */

accessible = access(path, mode)
int accessible;
char *path;
int mode;

DESCRIPTION
access checks the given file path for accessibility according to mode, which is an inclusive or
of the bits R_OK, W_OK and X_OK. Specifying mode as F_OK (i.e., 0) tests whether the direc-
tories leading to the file can be searched and the file exists.

The real user -s1ID and the group access list (including the real group -s1ID) are used in verifying permission, so this call is useful to set-UID programs.

Notice that 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 may
look executable, but execve will fail unless it is in proper format.

RETURN VALUE
If path cannot be found or if any of the desired access modes would not be granted, then a -1
value is returned; otherwise a 0 value is returned.

ERRORS
Access to the file is denied if one or more of the following are true:

[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 255 characters, or an entire
path name exceeded 1023 characters.
[ENOENT] The named file 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 path-
name.

[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.

[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.

[EFAULT] path points outside the process's allocated address space.
[EIO]

SEE ALSO
chmod(2), stat(2)

An I/O error occurred while reading from or writing to the file system.
NAME
  acct — turn accounting on or off

SYNOPSIS
  acct(file)
  char *file;

DESCRIPTION
  The system is prepared to write a record in an accounting file for each
  process as it terminates. This call, with a null-terminated string naming
  an existing file as argument, turns on accounting; records for each
  terminating process are appended to file. An argument of 0 causes
  accounting to be turned off.

  The accounting file format is given in acct(5).

  This call is permitted only to the super-user.

NOTES
  Accounting is automatically disabled when the file system the accounting
  file resides on runs out of space; it is enabled when space once again
  becomes available.

RETURN VALUE
  On error −1 is returned. The file must exist and the call may be exercized
  only by the super-user. It is erroneous to try to turn on accounting when it
  is already on.

ERRORS
  acct will fail if one of the following is true:

  [EPERM]          The caller is not the super-user.
  [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 255 characters, or an
                   entire path name exceeded 1023 characters.
  [ENOENT]         The named file does not exist.
  [EACCES]         Search permission is denied for a component of the path
                   prefix, or the path name is not a regular file.
  [ELOOP]          Too many symbolic links were encountered in translating the
                   pathname.
  [EROFS]          The named file resides on a read-only file system.
  [EFAULT]        file points outside the process's allocated address space.
  [EIO]            An I/O error occurred while reading from or writing to the
                   file system.

SEE ALSO
  acct(5), sa(8)

WARNING
  No accounting is produced for programs running when a crash occurs. In
  particular non-terminating programs are never accounted for.
NAME
adjtime – correct the time to allow synchronization of the system clock

SYNOPSIS
#include <sys/time.h>
adjtime(delta, olddelta)
struct timeval *delta;
struct timeval *olddelta;

DESCRIPTION
adjtime makes small adjustments to the system time, as returned by gettimeofday(2), advancing or retarding it by the time specified by the timeval delta. 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. 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. If olddelta is non-zero, then the structure pointed to will contain, upon return, the number of microseconds still to be corrected 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 call adjtime(2) is restricted to the super-user.

RETURN VALUE
A return value of 0 indicates that the call succeeded. A return value of −1 indicates that an error occurred, and in this case an error code is stored in the global variable errno.

ERRORS
The following error codes may be set in errno:
[EFAULT] An argument points outside the process’s allocated address space.
[EPERM] The process’s effective user ID is not that of the super-user.

SEE ALSO
date(1), gettimeofday(2), timed(8), timedc(8),
TSP: The Time Synchronization Protocol for UNIX x4.3BSD, R. Gusella and S. Zatti
NAME
bind – bind a name to a socket

SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>
bind(s, name, namelen)
int s;
struct sockaddr *name;
int namelen;

DESCRIPTION
bind assigns a name to an unnamed socket. When a socket is created with socket(2) it exists
in a name space (address family) but has no name assigned. bind requests that name be
assigned to the socket.

NOTES
Binding a name in the UNIX domain creates a socket in the file system that must be deleted by
the caller when it is no longer needed (using unlink(2)).

The rules used in name binding vary between communication domains. Consult the manual
entries in section 4 for detailed information.

RETURN VALUE
If the bind is successful, a 0 value is returned. A return value of -1 indicates an error, which
is further specified in the global errno.

ERRORS
The bind call will fail if:
[EBADF] S is not a valid descriptor.
[ENOTSOCK] S is 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 has inade-
quate permission to access it.
[EFAULT] The name parameter is not in a valid part of the user address
space.

The following errors are specific to binding names in the UNIX domain.
[ENOTDIR] A component of the path prefix is not a directory.
[EINVAL] The pathname contains a character with the high-order bit set.
[ENAMEETOOLONG] A component of a pathname exceeded 255 characters, or an entire
path name exceeded 1023 characters.
[ENOENT] A prefix component of the path name does not exist.
[ELOOP] Too many symbolic links were encountered in translating the path-
name.
[EIO] An I/O error occurred while making the directory entry or allocating
the inode.
[EROFS] The name would reside on a read-only file system.
[EISDIR] A null pathname was specified.
SEE ALSO
  connect(2), listen(2), socket(2), getsockname(2)
NAME
  brk, sbrk – change data segment size

SYNOPSIS
  #include <sys/types.h>
  char *brk(addr)
  char *addr;
  char *sbrk(incr)
  int incr;

DESCRIPTION
  brk sets the system’s idea of 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). Locations
  greater than addr and below the stack pointer are not in the address space and will thus cause
  a memory violation if accessed.

  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 via execve the break is set at the highest location defined by
  the program and data storage areas. Ordinarily, therefore, only programs with growing data
  areas need to use sbrk.

  The getrlimit(2) system call may be used to determine the maximum permissible size of the
  data segment; it will not be possible to set the break beyond the rlim_max value returned from
  a call to getrlimit, e.g. “etext + rlp->rlim_max.” (see end(3) for the definition of etext).

RETURN VALUE
  Zero is returned if the brk could be set; -1 if the program requests more memory than the system
  limit. sbrk returns -1 if the break could not be set.

ERRORS
  sbrk will fail and no additional memory will be allocated if one of the following are true:
  [ENOMEM] The limit, as set by setrlimit(2), was exceeded.
  [ENOMEM] The maximum possible size of a data segment (compiled into the system) was exceeded.
  [ENOMEM] Insufficient space existed in the swap area to support the expansion.

SEE ALSO
  execve(2), getrlimit(2), malloc(3), end(3)

WARNING
  Setting the break may fail due to a temporary lack of swap space. It is not possible to distin-
  guish this from a failure caused by exceeding the maximum size of the data segment without
  consulting getrlimit.
NAME
cachectl – mark pages cacheable or uncachable

SYNOPSIS
#include <mips/cachectl.h>

cachectl(addr, nbytes, op)
char *addr;
int nbytes, op;

DESCRIPTION
The cachectl system call allows a process to make ranges of its address space cacheable or
uncacheable. Initially, a process's entire address space is cacheable.

op may be one of:
CACHEABLE     Make the indicated pages cacheable
UNCACHEABLE   Make the indicated pages uncacheable

The CACHEABLE and UNCACHEABLE op's affect the address range indicated by addr and
nbytes. addr must be page aligned and nbytes must be a multiple of the page size.

Changing a page from UNCACHEABLE state to CACHEABLE state will cause both the
instruction and data caches to be flushed if necessary to avoid stale cache information.

RETURN VALUE
cachectl returns 0 when no errors are detected. If errors are detected, cachectl returns -1 with
the error cause indicated in errno.

ERRORS
[EINVAL]     op parameter is not one of CACHEABLE or UNCACHEABLE.
[EINVAL]     addr is not page aligned, or nbytes is not multiple of pagesize.
[EFAULT]     Some or all of the address range addr to (addr+nbytes-1) is not access-
able.

SEE ALSO
getpagesize(2)
NAME
cacheflush – flush contents of instruction and/or data cache

SYNOPSIS
#include <mips/cachectl.h>

cacheflush(addr, nbytes, cache)
char *addr;
int nbytes, cache;

DESCRIPTION
Flushes contents of indicated cache(s) for user addresses in the range addr to (addr+nbytes-1).
cache may be one of:
 ICACHE   Flush only the instruction cache
 DCACHE   Flush only the data cache
 BCACHE   Flush both instruction and data caches

RETURN VALUE
cacheflush returns 0 when no errors are detected. If errors are detected, cacheflush returns -1
with the error cause indicated in errno.

ERRORS
 [EINVAL]   cache parameter is not one of ICACHE, DCACHE, or BCACHE.
 [EFAULT]   Some or all of the address range addr to (addr+nbytes-1) is not accessible.
NAME
    chdir – change current working directory

SYNOPSIS
    chdir(path)
    char *path;

DESCRIPTION
    path is the pathname of a directory. chdir causes this directory to become the current working
directory, the starting point for path names not beginning with “/”.

In order for a directory to become the current directory, a process must have execute (search)
access to the directory.

RETURN VALUE
    Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned
and errno is set to indicate the error.

ERRORS
    chdir will fail and the current working directory will be unchanged if one or more of the fol-
lowing are true:

    [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 255 characters, or an entire
                    path name exceeded 1023 characters.
    [ENOENT]        The named directory does not exist.
    [ELOOP]         Too many symbolic links were encountered in translating the path-
                    name.
    [EACCES]        Search permission is denied for any component of the path name.
    [EFAULT]       path points outside the process’s allocated address space.
    [EIO]           An I/O error occurred while reading from or writing to the file sys-
                    tem.

SEE ALSO
    chroot(2)
NAME
chmod – change mode of file

SYNOPSIS
chmod(path, mode)
char *path;
int mode;

fchmod(fd, mode)
int fd, mode;

DESCRIPTION
The file whose name is given by path or referenced by the descriptor fd has its mode changed
to mode. Modes are constructed by or’ing together some combination of the following,
declared in <sys/inode.h>:

- ISUID 04000  set user ID on execution
- ISGID 02000  set group ID on execution
- ISVTX 01000  ‘sticky bit’ (see below)
- IREAD 00400  read by owner
- IWRITE 00200  write by owner
- IEXEC 00100  execute (search on directory) by owner
  00070  read, write, execute (search) by group
  00067  read, write, execute (search) by others

If an executable file is set up for sharing (this is the default) then mode ISVTX (the ‘sticky
bit’) prevents the system from abandoning the swap-space image of the program-text portion
of the file when its last user terminates. Ability to set this bit on executable files is restricted
to the super-user.

If mode ISVTX (the ‘sticky bit’) is set on a directory, an unprivileged user may not delete or
rename files of other users in that directory. For more details of the properties of the sticky
bit, see sticky(8).

Only the owner of a file (or the super-user) may change the mode.

Writing or changing the owner of a file turns off the set-user-id and set-group-id bits unless the
user is the super-user. This makes the system somewhat more secure by protecting set-user-id
(set-group-id) files from remaining set-user-id (set-group-id) if they are modified, at the expense
of a degree of compatibility.

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned
and errno is set to indicate the error.

ERRORS
Chmod will fail and the file mode will be unchanged if:

- [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 255 characters, or an entire path name
                   exceeded 1023 characters.
- [ENOENT]     The named file 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 effective user ID does not match the owner of the file and the effective
user ID is not the super-user.

[EROFS] The named file resides on a read-only file system.

[EFAULT] Path points outside the process's allocated address space.

[EIO] An I/O error occurred while reading from or writing to the file system.

_Fchmod_ will fail if:

[EBADF] The descriptor is not valid.

[EINVAL] _Fd_ refers to a socket, not to a file.

[EROFS] The file resides on a read-only file system.

[EIO] An I/O error occurred while reading from or writing to the file system.

SEE ALSO

chmod(1), open(2), chown(2), stat(2), sticky(8)
NAME
    chown - change owner and group of a file

SYNOPSIS
    chown(path, owner, group)
    char *path;
    int owner, group;
    fchown(fd, owner, group)
    int fd, owner, group;

DESCRIPTION
    The file that is named by path or referenced by fd has its owner and group changed as specified. Only the super-user may change the owner of the file, because if users were able to give files away, they could defeat the file-space accounting procedures. The owner of the file may change the group to a group of which he is a member.

    On some systems, chown clears the set-user-id and set-group-id bits on the file to prevent accidental creation of set-user-id and set-group-id programs.

    fchown is particularly useful when used in conjunction with the file locking primitives (see flock(2)).

    One of the owner or group id's may be left unchanged by specifying it as -1.

    If the final component of path is a symbolic link, the ownership and group of the symbolic link is changed, not the ownership and group of the file or directory to which it points.

RETURN VALUE
    Zero is returned if the operation was successful; -1 is returned if an error occurs, with a more specific error code being placed in the global variable errno.

ERRORS
    chown will fail and the file will be unchanged if:
    [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 255 characters, or an entire path name exceeded 1023 characters.
    [ENOENT] The named file 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 effective user ID is not the super-user.
    [EROFS] The named file resides on a read-only file system.
    [EFAULT] path points outside the process's allocated address space.
    [EIO] An I/O error occurred while reading from or writing to the file system.

    fchown will fail if:
    [EBADF] fd does not refer to a valid descriptor.
    [EINVAL] fd refers to a socket, not a file.
    [EPERM] The effective user ID is not the super-user.
    [EROFS] The named file resides on a read-only file system.
[EIO] An I/O error occurred while reading from or writing to the file system.

SEE ALSO
chown(8), chgrp(1), chmod(2), flock(2)
NAME
close – delete a descriptor

SYNOPSIS
close(d)
int d;

DESCRIPTION
The close call deletes a descriptor from the per-process object reference table. If this is the last reference to the underlying object, then it will be deactivated. For example, on the last close of a file the current seek pointer associated with the file is lost; on the last close of a socket(2) associated naming information and queued data are discarded; on the last close of a file holding an advisory lock the lock is released (see further flock(2)).

A close of all of a process’s descriptors is automatic on exit, but since there is a limit on the number of active descriptors per process, close is necessary for programs that deal with many descriptors.

When a process forks (see fork(2)), all descriptors for the new child process reference the same objects as they did in the parent before the fork. If a new process is then to be run using execve(2), the process would normally inherit these descriptors. Most of the descriptors can be rearranged with dup2(2) or deleted with close before the execve is attempted, but if some of these descriptors will still be needed if the execve fails, it is necessary to arrange for them to be closed if the execve succeeds. For this reason, the call fcntl(d, F_SETFD, 1) is provided, which arranges that a descriptor will be closed after a successful execve; the call fcntl(d, F_SETFD, 0) restores the default, which is to not close the descriptor.

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the global integer variable errno is set to indicate the error.

ERRORS
close will fail if:
[EBADF] d is not an active descriptor.

SEE ALSO
accept(2), flock(2), open(2), pipe(2), socket(2), socketpair(2), execve(2), fcntl(2)
NAME
connect – initiate a connection on a socket

SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>
connect(s, name, namelen)
int s;
struct sockaddr *name;
int namelen;

DESCRIPTION
The parameter s is a socket. If it is of type SOCK_DGRAM, then this call specifies the peer
with which the socket is to be associated; this address is that to which datagrams are to be
sent, and the only address from which datagrams are to be received. If the socket is of type
SOCK_STREAM, then this call attempts to make a connection to another socket. The other
socket is specified by name, which is an address in the communications space of the socket.
Each communications space interprets the name parameter in its own way. Generally, stream
sockets may successfully connect only once; datagram sockets may use connect multiple times
to change their association. Datagram sockets may dissolve the association by connecting to
an invalid address, such as a null address.

RETURN VALUE
If the connection or binding succeeds, then 0 is returned. Otherwise a −1 is returned, and a
more specific error code is stored in errno.

ERRORS
The call fails if:
[EBADF] s is not a valid descriptor.
[ENOTSOCK] s is a descriptor for a file, not a socket.
[EADDRNOTAVAIL] The specified address is not available on this machine.
[EAFNOSUPPORT] Addresses in the specified address family cannot be used with this
socket.
[EISCONN] The socket is already connected.
[ETIMEDOUT] Connection establishment timed out without establishing a connection.
[ECONNREFUSED] The attempt to connect was forcefully rejected.
[ENETUNREACH] The network isn’t reachable from this host.
[EADDRINUSE] The address is already in use.
[EFAULT] The name parameter specifies an area outside the process address
space.
[EINPROGRESS] The socket is non-blocking and the connection cannot be com-
mpleted immediately. It is possible to select(2) for completion by
selecting the socket for writing.
[EALREADY] The socket is non-blocking and a previous connection attempt has
not yet been completed.

The following errors are specific to connecting names in the UNIX domain. These errors may
not apply in future versions of the UNIX IPC domain.
[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 255 characters, or an entire path name exceeded 1023 characters.
[ENOENT] The named socket does not exist.
[EACCES] Search permission is denied for a component of the path prefix.
[EACCES] Write access to the named socket is denied.
[ELOOP] Too many symbolic links were encountered in translating the pathname.

SEE ALSO accept(2), select(2), socket(2), getsockname(2)
NAME
creat – create a new file

SYNOPSIS
creat(name, mode)
char *name;

DESCRIPTION
This interface is made obsolete by open(2).
creat creates a new file or prepares to rewrite an existing file called name, given as the address of a null-terminated string. If the file did not exist, it is given mode mode, as modified by the process's mode mask (see umask(2)). Also see chmod(2) for the construction of the mode argument.

If the file did exist, its mode and owner remain unchanged but it is truncated to 0 length.
The file is also opened for writing, and its file descriptor is returned.

NOTES
The mode given is arbitrary; it need not allow writing. This feature has been used in the past by programs to construct a simple, exclusive locking mechanism. It is replaced by the O_EXCL open mode, or flock(2) facility.

RETURN VALUE
The value -1 is returned if an error occurs. Otherwise, the call returns a non-negative descriptor that only permits writing.

ERRORS
creat will fail and the file will not be created or truncated if one of the following occur:

[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 255 characters, or an entire path name exceeded 1023 characters.
[ENOENT] The named file does not exist.
[ELOOP] Too many symbolic links were encountered in translating the pathname.
[EACCES] Search permission is denied for a component of the path prefix.
[EACCES] The file does not exist and the directory in which it is to be created is not writable.
[EACCES] The file exists, but it is unwritable.
[EISDIR] The file is a directory.
[EMFILE] There are already too many files open.
[ENFILE] The system file table is full.
[ENOSPC] The directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory.
[ENOSPC] There are no free inodes on the file system on which the file is being created.
[EDQUOT] The directory in which the entry for the new file is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.
[EDQUOT] The user's quota of inodes on the file system on which the file is being created has been exhausted.

[EROFS] The named file resides on a read-only file system.

[ENXIO] The file is a character special or block special file, and the associated device does not exist.

[ETXTBSY] The file is a pure procedure (shared text) file that is being executed.

[EIO] An I/O error occurred while making the directory entry or allocating the inode.

[EFAULT] name points outside the process's allocated address space.

[EOPNOTSUPP] The file was a socket (not currently implemented).

SEE ALSO open(2), write(2), close(2), chmod(2), umask(2)
NAME

dup, dup2 — duplicate a descriptor

SYNOPSIS

newd = dup(oldd)
int newd, oldd;
dup2(oldd, newd)
int oldd, newd;

DESCRIPTION

dup duplicates an existing object descriptor. The argument oldd is a small non-negative
integer index in the per-process descriptor table. The value must be less than the size of the
table, which is returned by getdtablesize(2). The new descriptor returned by the call, newd, is
the lowest numbered descriptor that is not currently in use by the process.

The object referenced by the descriptor does not distinguish between references using oldd
and newd in any way. Thus if newd and oldd are duplicate references to an open file, read(2),
write(2) and lseek(2) calls all move a single pointer into the file, and append mode, non-
blocking I/O and asynchronous I/O options are shared between the references. If a separate
pointer into the file is desired, a different object reference to the file must be obtained by issuing
an additional open(2) call. The close-on-exec flag on the new file descriptor is unset.

In the second form of the call, the value of newd desired is specified. If this descriptor is
already in use, the descriptor is first deallocated as if a close(2) call had been done first.

RETURN VALUE

The value −1 is returned if an error occurs in either call. The external variable errno indicates
the cause of the error.

ERRORS

dup and dup2 fail if:

[EBADF]    oldd or newd is not a valid active descriptor
[EMFILE]   Too many descriptors are active.

SEE ALSO

accept(2), open(2), close(2), fcntl(2), pipe(2), socket(2), socketpair(2), getdtablesize(2)
NAME
execve – execute a file

SYNOPSIS
execve(name, argv, envp)
char *name, *argv[], *envp[];

DESCRIPTION
execve transforms the calling process into a new process. The new process is constructed
from an ordinary file called the new process file. This file is either an executable object file, or
a file of data for an interpreter. An executable object file consists of an identifying header,
followed by pages of data representing the initial program (text) and initialized data pages.
Additional pages may be specified by the header to be initialized with zero data. See a.out(5).

An interpreter file begins with a line of the form “#! interpreter”. When an interpreter file is
execve’d, the system execve’s the specified interpreter, giving it the name of the originally
exec’d file as an argument and shifting over the rest of the original arguments.

There can be no return from a successful execve because the calling core image is lost. This is
the mechanism whereby different process images become active.

The argument argv is a null-terminated array of character pointers to null-terminated character
strings. These strings constitute the argument list to be made available to the new process.
By convention, at least one argument must be present in this array, and the first element of
this array should be the name of the executed program (i.e., the last component of name).

The argument envp is also a null-terminated array of character pointers to null-terminated
strings. These strings pass information to the new process that is not directly an argument to
the command (see environ(7)).

Descriptors open in the calling process remain open in the new process, except for those for
which the close-on-exec flag is set (see close(2)). Descriptors that remain open are unaffected
by execve.

Ignored signals remain ignored across an execve, but signals that are caught are reset to their
default values. Blocked signals remain blocked regardless of changes to the signal action.
The signal stack is reset to be undefined (see sigvec(2) for more information).

Each process has real user and group IDs and an effective user and group IDs. The real ID
identifies the person using the system; the effective ID determines his access privileges. execve
changes the effective user and group ID to the owner of the executed file if the file has the
“set-user-ID” or “set-group-ID” modes. The real user ID is not affected.

The new process also inherits the following attributes from the calling process:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>process ID</td>
<td>getpid(2)</td>
</tr>
<tr>
<td>parent process ID</td>
<td>getppid(2)</td>
</tr>
<tr>
<td>process group ID</td>
<td>getpgid(2)</td>
</tr>
<tr>
<td>access groups</td>
<td>getgroups(2)</td>
</tr>
<tr>
<td>working directory</td>
<td>chdir(2)</td>
</tr>
<tr>
<td>root directory</td>
<td>chroot(2)</td>
</tr>
<tr>
<td>control terminal</td>
<td>tty(4)</td>
</tr>
<tr>
<td>resource usages</td>
<td>getrusage(2)</td>
</tr>
<tr>
<td>interval timers</td>
<td>getitimer(2)</td>
</tr>
<tr>
<td>resource limits</td>
<td>getrlimit(2)</td>
</tr>
<tr>
<td>file mode mask</td>
<td>umask(2)</td>
</tr>
<tr>
<td>signal mask</td>
<td>sigvec(2), sigmask(2)</td>
</tr>
</tbody>
</table>

MIPS Computer Systems, Inc. February 5, 1989
When the executed program begins, it is called as follows:

main(argc, argv, envp)
int argc;
char **argv, **envp;

where argc is the number of elements in argv (the "arg count") and argv is the array of character pointers to the arguments themselves.

envp is a pointer to an array of strings that constitute the environment of the process. A pointer to this array is also stored in the global variable "environ". Each string consists of a name, an "=" and a null-terminated value. The array of pointers is terminated by a null pointer. The shell sh(1) passes an environment entry for each global shell variable defined when the program is called. See environ(7) for some conventionally used names.

RETURN VALUE

If execve returns to the calling process an error has occurred; the return value will be -1 and the global variable errno will contain an error code.

ERRORS

execve will fail and return to the calling process if one or more of the following are true:

[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 255 characters, or an entire path name exceeded 1023 characters.
[ENOENT] The new process file does not exist.
[ELOOP] Too many symbolic links were encountered in translating the pathname.
[EACCES] Search permission is denied for a component of the path prefix.
[EACCES] The new process file is not an ordinary file.
[EACCES] The new process file mode denies execute permission.
[ENOEXEC] The new process file has the appropriate access permission, but has an invalid magic number in its header.
[ETXTBSY] The new process file is a pure procedure (shared text) file that is currently open for writing or reading by some process.
[ENOMEM] The new process requires more virtual memory than is allowed by the imposed maximum (getrlimit(2)).
[E2BIG] The number of bytes in the new process’s argument list is larger than the system-imposed limit. The limit in the system as released is 20480 bytes (NCARGS) in <sys/param.h>.
[EFAULT] The new process file is not as long as indicated by the size values in its header.
[EFAULT] path, argv, or envp point to an illegal address.
[EIO] An I/O error occurred while reading from the file system.

CAVEATS

If a program is setuid to a non-super-user, but is executed when the real uid is "root", then the program has some of the powers of a super-user as well.
SEE ALSO
exit(2), fork(2), execl(3), environ(7)
NAME
_exit – terminate a process

SYNOPSIS
_exit(status)
int status;

DESCRIPTION
_exit terminates a process with the following consequences:

All of the descriptors open in the calling process are closed. This may entail delays, for
example, waiting for output to drain; a process in this state may not be killed, as it is already
dying.

If the parent process of the calling process is executing a wait or is interested in the SIGCHLD
signal, then it is notified of the calling process’s termination and the low-order eight bits of
status are made available to it; see wait(2).

The parent process ID of all of the calling process’s existing child processes are also set to 1.
This means that the initialization process (see intro(2)) inherits each of these processes as
well. Any stopped children are restarted with a hangup signal (SIGHUP).

Most C programs call the library routine exit(3), which performs cleanup actions in the stan-
dard I/O library before calling _exit.

RETURN VALUE
This call never returns.

SEE ALSO
fork(2), sigvec(2), wait(2), exit(3)
NAME
fcntl — file control

SYNOPSIS
#include <fcntl.h>
res = fcntl(fd, cmd, arg)
int res;
int fd, cmd, arg;

DESCRIPTION
fcntl performs a variety of functions on open descriptors. The argument fd is an open descriptor to be operated on by cmd as follows:

F_DUPFD Return a new descriptor as follows:
Lowest numbered available descriptor greater than or equal to arg.
References the same object as the original descriptor.
New descriptor shares the same file pointer if the object was a file.
Same file access mode (read, write or read/write).
Same file status flags (i.e., both descriptors share the same file status flags).
The close-on-exec flag associated with the new descriptor is set to remain open across exec(2) system calls.

F_GETFD Get the close-on-exec flag associated with the descriptor fd. If the low-order bit is 0, the file will remain open across exec, otherwise the file will be closed upon execution of exec.

F_SETFD Set the close-on-exec flag associated with fd to the low order bit of arg (0 or 1 as above).

F_GETFL Get descriptor status flags, see /usr/include/fcntl.h for their definitions.

F_SETFL Set descriptor status flags, see /usr/include/fcntl.h for their definitions.

F_GETLK Get a description of the first lock which would block the lock specified in the flock structure pointed to by arg. The information retrieved overwrites the information in the flock structure. If no lock is found that would prevent this lock from being created, then the structure is passed back unchanged except for the lock type which will be set to F_UNLCK.

F_SETLK Set or clear an advisory record lock according to the flock structure pointed to by arg. F_SETLK is used to establish shared (F_RDLCK) and exclusive (F_WRLCK) locks, or to remove either type of lock (F_UNLCK). If the specified lock cannot be applied, fcntl will return with an error value of -1.

F_SETLKW This cmd is the same as F_SETLK except that if a shared or exclusive lock is blocked by other locks, the requesting process will sleep until the lock may be applied.

F_GETOWN Get the process ID or process group currently receiving SIGIO and SIGURG signals; process groups are returned as negative values.

F_SETOWN Set the process or process group to receive SIGIO and SIGURG signals; process groups are specified by supplying arg as negative, otherwise arg is interpreted as a process ID.

The SIGIO facilities are enabled by setting the FASYNC flag with F_SETFL.
Advisory locks allow cooperating processes to perform consistent operations on files, but do not guarantee exclusive access (i.e., processes may still access files without using advisory locks, possibly resulting in inconsistencies).

The record locking mechanism allows two types of locks: shared locks (F_RDLCK) and exclusive locks (F_WRLCK). More than one process may hold a shared lock for a particular segment of a file at any given time, but multiple exclusive, or both shared and exclusive, locks may not exist simultaneously on any segment.

In order to claim a shared lock, the descriptor must have been opened with read access. The descriptor on which an exclusive lock is being placed must have been opened with write access.

A shared lock may be upgraded to an exclusive lock, and vice versa, simply by specifying the appropriate lock type with a cmd of F_SETLKW or F_SETLK; the previous lock will be released and the new lock applied (possibly after other processes have gained and released the lock).

If the cmd is F_SETLK and the requested lock cannot be claimed immediately (e.g., another process holds an exclusive lock that partially or completely overlaps the current request) then the calling process will block until the lock may be acquired. Processes blocked awaiting a lock may be awakened by signals.

Care should be taken to avoid deadlock situations in applications in which multiple processes perform blocking locks on a set of common records.

The record that is to be locked or unlocked is described by the flock structure, which is defined in <fcntl.h> as follows:

```c
struct flock {
    short l_type;   /* F_RDLCK, F_WRLCK, or F_UNLCK */
    short l_whence; /* flag to choose starting offset */
    long l_start;   /* relative offset, in bytes */
    long l_len;     /* length, in bytes; 0 means lock to EOF */
    short l_pid;    /* returned with F_GETLK */
};
```

The flock structure describes the type (l_type), starting offset (l_whence), relative offset (l_start), and size (l_len) of the segment of the file to be affected. l_whence must be set to 0, 1, or 2 to indicate that the relative offset will be measured from the start of the file, current position, or end-of-file, respectively. The process id field (l_pid) is only used with the F_GETLK cmd to return the description of a lock held by another process.

Locks may start and extend beyond the current end-of-file, but may not be negative relative to the beginning of the file. A lock may be set to always extend to the end-of-file by setting l_len to zero (0). If such a lock also has l_whence and l_start set to zero (0), the entire file will be locked. Changing or unlocking a segment from the middle of a larger locked segment leaves two smaller segments 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 affect. All locks associated with a file for a given process are removed when the file is closed or the process terminates. Locks are not inherited by the child process in a fork(2) system call.

In order to maintain consistency in the network case, data must not be cached on client machines. For this reason, file buffering for an NFS file is turned off when the first lock is attempted on the file. Buffering will remain off as long as the file is open. Programs that do I/O buffering in the user address space, however, may have inconsistent results (the standard I/O package, for instance, is a common source of unexpected buffering).
The advisory record locking capabilities of *fcntl* are implemented throughout the network by the network lock daemon; see *lockd*(8C). If the file server crashes and is rebooted, the lock daemon will attempt to recover all locks that were associated with that server. If a lock cannot be reclaimed, the process that held the lock will be issued a SIGLOST signal.

**RETURN VALUE**

Upon successful completion, the value returned depends on *cmd* as follows:

- F_DUPFD: A new descriptor.
- F_GETFD: Value of flag (only the low-order bit is defined).
- F_GETFL: Value of flags.
- F_GETOWN: Value of descriptor owner.
- other: Value other than −1.

Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

**ERRORS**

*fcntl* will fail if one or more of the following are true:

- EBADF: *fd* is not a valid open descriptor.
- [EEMMEILE]: *cmd* is F_DUPFD and the maximum allowed number of descriptors are currently open.
- EINVAL: *cmd* is F_DUPFD and *arg* is negative or greater than the maximum allowed number (see *getdtablesize*(2)).
- EFAULT: *cmd* is F_GETLK, F_SETLK, or F_SETLKW and *arg* points to an invalid address.
- EINVAL: *cmd* is F_GETLK, F_SETLK, or F_SETLKW and the data *arg* points to is not valid.
- EBADF: *cmd* is F_SETLK or F_SETLKW and the process does not have the appropriate read or write permissions on the file.
- EAGAIN: *cmd* is F_SETLK, the lock type (*l_type*) is F_RDLCK (shared lock), and the segment of the file to be locked already has an exclusive lock held by another process. This error will also be returned if the lock type is F_WRLCK (exclusive lock) and another process already has the segment locked with either a shared or exclusive lock.

*cmd* is F_SETLKW and a signal interrupted the process while it was waiting for the lock to be granted.

- ENOLCK: *cmd* is F_SETLK or F_SETLKW and there are no more file lock entries available.

**SEE ALSO**


**BUGS**

File locks obtained through the *fcntl* mechanism do not interact in any way with those acquired via *flock*(2). They do, however, work correctly with the exclusive locks claimed by *lockf*(3).

F_GETLK returns F_UNLCK if the requesting process holds the specified lock. Thus, there is no way for a process to determine if it is still holding a specific lock after catching a SIGLOST signal.

In a network environment, the value of *l_pid* returned by F_GETLK is next to useless.
NAME
fixade – fix address exceptions (unaligned references)

SYNOPSIS
fixade(x)
int x;

DESCRIPTION
This system call enables or disables kernel fix up of misaligned memory references. The MIPS hardware traps load and store operations where the address is not a multiple of the number of bytes loaded or stored. Usually this trap indicates incorrect program operation and so by default the kernel converts this trap into a SIGBUS signal to the process, typically causing a core dump for debugging.

Older programs developed on systems with lax alignment constraints sometimes make occasional misaligned references in course of correct operation. The best way to port such programs to MIPS hardware is to correct the program by aligning the data. A SIGBUS handler exists to assist the programmer in locating unaligned references. See unaligned(3).

Some applications, however, must deal with unaligned data. The MIPS architecture provides special instructions, supported by builtin assembler macros, for loading and storing unaligned data. These applications can use these instructions where appropriate. Non-assembler programs can access these instructions via calls, also described in unaligned(3).

When it is inappropriate to modify the application to either align the data properly, or to use special access methods for unaligned data, this system call, fixade, can be used as a method of last resort. This system call directs the kernel to handle misaligned traps and emulate an unaligned reference. The program no longer receives a SIGBUS signal. This emulation is slow, and heavy use will significantly slow down program execution.

A non-zero argument enables and a zero argument disables the fix up.

If the program gets an address exception when making a reference outside its address space, it will still get a SIGBUS signal even if this is enabled.

SEE ALSO
unaligned(3)
NAME
flock – apply or remove an advisory lock on an open file

SYNOPSIS
#include <sys/file.h>
#define LOCK_SH  1 /* shared lock */
#define LOCK_EX  2 /* exclusive lock */
#define LOCK_NB  4 /* don’t block when locking */
#define LOCK_UN  8 /* unlock */
flock(fd, operation)
int fd, operation;

DESCRIPTION
flock applies or removes an advisory lock on the file associated with the file descriptor fd. A
lock is applied by specifying an operation parameter that is the inclusive or of LOCK_SH or
LOCK_EX and, possibly, LOCK_NB. To unlock an existing lock operation should be
LOCK_UN.

Advisory locks allow cooperating processes to perform consistent operations on files, but do
not guarantee consistency (i.e., 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 hap-
pen; instead the call will fail and the error EWOULDBLOCK will be returned.

NOTES
Locks are on files, not file descriptors. That is, file descriptors duplicated through dup(2) or
fork(2) 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.

RETURN VALUE
Zero is returned if the operation was successful; on an error a -1 is returned and an error
code is left in the global location errno.

ERRORS
The flock call fails if:
[EMWOULDBLOCK] The file is locked and the LOCK_NB option was specified.
[EBADF] The argument fd is an invalid descriptor.
[EINVAL] The argument fd refers to an object other than a file.

SEE ALSO
open(2), close(2), dup(2), execve(2), fork(2)
NAME
fork – create a new process

SYNOPSIS
pid = fork()
int pid;

DESCRIPTION
fork causes creation of a new process. The new process (child process) is an exact copy of the calling process except for the following:

The child process has a unique process ID.

The child process has a different parent process ID (i.e., the process ID of the parent process).

The child process has its own copy of the parent’s descriptors. These descriptors reference the same underlying objects, so that, for instance, file pointers in file objects are shared between the child and the parent, so that an lseek(2) on a descriptor in the child process can affect a subsequent read or write by the parent. This descriptor copying is also used by the shell to establish standard input and output for newly created processes as well as to set up pipes.

The child processes resource utilizations are set to 0; see setrlimit(2).

The child process does not receive real interval timer signals that were arranged by the parent; however, both virtual and profiling interval timer signals will continue to arrive.

RETURN VALUE
Upon successful completion, fork returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of −1 is returned to the parent process, no child process is created, and the global variable errno is set to indicate the error.

ERRORS
fork will fail and no child process will be created if one or more of the following are true:

[EAGAIN] The system-imposed limit on the total number of processes under execution would be exceeded. This limit is configuration-dependent.

[EAGAIN] The system-imposed limit MAXUPRC (<sys/param.h>) on the total number of processes under execution by a single user would be exceeded.

[ENOMEM] There is insufficient swap space for the new process.

SEE ALSO
execve(2), wait(2)
NAME
   fp_sigintr – generate a SIGFPE signal on floating-point interrupts

SYNOPSIS
   int fp_sigintr(x)
   int x;

DESCRIPTION
   The fp_sigintr system call causes every other floating-point interrupt to generate a SIGFPE signal. If the argument is 1 the next floating-point interrupt will cause a signal with the following one not causing a signal. If the argument is a 2 then the the next floating-point interrupt will not cause a signal with the following one causing a signal. If the argument is a 0 then the this feature is disabled and floating-point interrupts will not cause a signal.

   This is intended for use by fpi(3) to analyze the causes of floating-point interrupts.

ALSO SEE
   fpi(3)
   R2010 Floating Point Coprocessor Architecture
   R2360 Floating Point Board Product Description
NAME
fsync – synchronize a file’s in-core state with that on disk

SYNOPSIS
fsync(fd)
int fd;

DESCRIPTION
fsync causes all modified data and attributes of fd to be moved to a permanent storage device. This normally results in all in-core modified copies of buffers for the associated file to be written to a disk.

fsync should be used by programs that require a file to be in a known state, for example, in building a simple transaction facility.

RETURN VALUE
A 0 value is returned on success. A -1 value indicates an error.

ERRORS
The fsync fails if:
[EBADF] Fd is not a valid descriptor.
[EINVAL] fd refers to a socket, not to a file.
[EIO] An I/O error occurred while reading from or writing to the file system.

SEE ALSO
sync(2), sync(8), update(8)
NAME
getdirentries – gets directory entries in a filesystem independent format

SYNOPSIS
#include <sys/dir.h>

cc = getdirentries(fd, buf, nbytes, basep)
int cc, fd;
char *buf;
int nbytes;
long *basep;

DESCRIPTION
getdirentries attempts to put directory entries from the directory referenced by the file descriptor fd into the buffer pointed to by buf, in a filesystem independent format. Up to nbytes of data will be transferred. nbytes must be greater than or equal to the block size associated with the file, see stat(2). Sizes less than this may cause errors on certain filesystems.

The data in the buffer is a series of direct structures each containing the following entries:

  unsigned long d_fileno;
  unsigned short d_reclen;
  unsigned short d_namelen;
  char d_name[MAXNAMELEN + 1]; /* see below */

The d_fileno entry is a number which is unique for each distinct file in the filesystem. Files that are linked by hard links (see link(2)) have the same d_fileno. The d_reclen entry is the length, in bytes, of the directory record. The d_name entry contains a null terminated file name. The d_namelen entry specifies the length of the file name. Thus the actual size of d_name may vary from 2 to MAXNAMELEN + 1.

The structures are not necessarily tightly packed. The d_reclen entry may be used as an offset from the beginning of a direct structure to the next structure, if any.

Upon return, the actual number of bytes transferred is returned. The current position pointer associated with fd is set to point to the next block of entries. The pointer is not necessarily incremented by the number of bytes returned by getdirentries. If the value returned is zero, the end of the directory has been reached. The current position pointer may be set and retrieved by lseek(2). getdirentries writes the position of the block read into the location pointed to by basep. It is not safe to set the current position pointer to any value other than a value previously returned by lseek(2) or a value previously returned in the location pointed to by basep or zero.

RETURN VALUE
If successful, the number of bytes actually transferred is returned. Otherwise, a -1 is returned and the global variable errno is set to indicate the error.

ERRORS
getdirentries will fail if one or more of the following are true:

  EBAADF  fd is not a valid file descriptor open for reading.
  EFAULT  Either buf or basep point outside the allocated address space.
  EIO     An I/O error occurred while reading from or writing to the file system.
  EINTR   A read from a slow device was interrupted before any data arrived by the delivery of a signal.
SEE ALSO
open(2V), lseek(2)
NAME

getdomainname, setdomainname – get/set name of current domain

SYNOPSIS

getdomainname(name, namelen)
char *name;
int namelen;

setdomainname(name, namelen)
char *name;
int namelen;

DESCRIPTION

getdomainname returns the name of the domain for the current processor, as previously set by
setdomainname. The parameter namelen specifies the size of the name array. The returned
name is null-terminated unless insufficient space is provided.

setdomainname sets the domain of the host machine to be name, which has length namelen.
This call is restricted to the super-user and is normally used only when the system is
bootstrapped.

The purpose of domains is to enable two distinct networks that may have host names in com-
mon to merge. Each network would be distinguished by having a different domain name. At
the current time, only the yellow pages service makes use of domains.

RETURN VALUE

If the call succeeds a value of 0 is returned. If the call fails, then a value of −1 is returned and
an error code is placed in the global location errno.

ERRORS

The following errors may be returned by these calls:

EFAULT The name parameter gave an invalid address.
EPERM The caller was not the super-user. This error only applies to set-
domainname.

WARNINGS

Domain names are limited to 255 characters.
NAME
getdtablesiz e - get descriptor table size

SYNOPSIS
int nfds;
nfds = getdtablesiz e();

DESCRIPTION
Each process has a fixed size descriptor table, which is guaranteed to have at least 20 slots. The entries in the descriptor table are numbered with small integers starting at 0. The call getdtablesiz e returns the size of this table.

SEE ALSO
close(2), dup(2), open(2), select(2)
NAME
getgid, getegid – get group identity

SYNOPSIS
#include <sys/types.h>

gid = getgid()
gid_t gid;
egid = getegid()
gid_t egid;

DESCRIPTION
getgid returns the real group ID of the current process, getegid the effective group ID.

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, and it is for such processes that getgid is most useful.

SEE ALSO
getuid(2), setregid(2), setgid(3)
NAME
getgroups – get group access list

SYNOPSIS
#include <sys/param.h>

ngroups = getgroups(gidsetlen, gidset)
int ngroups, gidsetlen, *gidset;

DESCRIPTION
getgroups gets the current group access list of the user process and stores it in the array gidset. The parameter gidsetlen indicates the number of entries that may be placed in gidset. getgroups returns the actual number of groups returned in gidset. No more than NGROUPS, as defined in <sys/param.h>, will ever be returned.

RETURN VALUE
A successful call returns the number of groups in the group set. A value of -1 indicates that an error occurred, and the error code is stored in the global variable errno.

ERRORS
The possible errors for getgroup are:

[EINVAL] The argument gidsetlen is smaller than the number of groups in the group set.

[EFAULT] The argument gidset specifies an invalid address.

SEE ALSO
setgroups(2), initgroups(3X)

WARNING
The gidset array should be of type gid_t, but remains integer for compatibility with earlier systems.
NAME
gethostid, seethostid — get/set unique identifier of current host

SYNOPSIS

hostid = gethostid()
long hostid;

seethostid(hostid)
long hostid;

DESCRIPTION

seethostid establishes a 32-bit identifier for the current processor that is intended to be unique among all UNIX systems in existence. This is normally a DARPA Internet address for the local machine. This call is allowed only to the super-user and is normally performed at boot time.

gethostid returns the 32-bit identifier for the current processor.

SEE ALSO

hostid(1), gethostname(2)

ERRORS

32 bits for the identifier is too small.
NAME
gethostname, sethostname — get/set name of current host

SYNOPSIS

gethostname(name, namelen)
char *name;
int namelen;

sethostname(name, namelen)
char *name;
int namelen;

DESCRIPTION
gethostname returns the standard host name for the current processor, as previously set by
callsethostname. The parameter namelen specifies the size of the name array. The returned name
is null-terminated unless insufficient space is provided.

sethostname sets the name of the host machine to be name, which has length namelen. This
call is restricted to the super-user and is normally used only when the system is bootstrapped.

RETURN VALUE
If the call succeeds a value of 0 is returned. If the call fails, then a value of -1 is returned and
an error code is placed in the global location errno.

ERRORS
The following errors may be returned by these calls:

[EFAULT] The name or namelen parameter gave an invalid address.
[EPERM] The caller tried to set the hostname and was not the super-user.
[EINVAL] The size specified by I. namelen is longer than the maximum host name length.

SEE ALSO
gethostid(2)

BUGS
Host names are limited to MAXHOSTNAMELEN (from <sys/param.h>) characters, currently
64.
NAME
getitimer, setitimer — get/set value of interval timer

SYNOPSIS
#include <sys/time.h>
#define ITIMER_REAL 0  /* real time intervals */
#define ITIMER_VIRTUAL 1 /* virtual time intervals */
#define ITIMER_PROF 2  /* user and system virtual time */

getitimer(which, value)
int which;
struct itimerval *value;

setitimer(which, value, ovalue)
int which;
struct itimerval *value, *ovalue;

DESCRIPTION
The system provides each process with three interval timers, defined in <sys/time.h>. The
getitimer call returns the current value for the timer specified in which in the structure at value.
The setitimer call 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:

struct itimerval {
  struct timeval it_interval;  /* timer interval */
  struct timeval it_value;    /* current value */
};

If it_value is non-zero, it indicates the time to the next timer expiration. If it_interval is non-
zero, it specifies a value to be used in reloading it_value when the timer expires. Setting
it_value to 0 disables a timer. Setting it_interval to 0 causes a timer to be disabled after its
next expiration (assuming it_value is non-zero).

Time values smaller than the resolution of the system clock are rounded up to this resolution
(on the VAX, 10 milliseconds).

The ITIMER_REAL timer decrements in real time. A SIGALRM signal is delivered when this
timer expires.

The ITIMER_VIRTUAL timer decrements in process virtual time. It runs only when the pro-
cess is executing. A SIGVTALRM signal is delivered when it expires.

The ITIMER_PROF timer decrements both in process virtual time and when the system is run-
nong on behalf of the process. It is designed to be used by interpreters in statistically profiling
the execution of interpreted programs. Each time the ITIMER_PROF timer expires, the SIG-
PROF 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 <sys/time.h>. Timerclear sets a
time value to zero, timerisset tests if a time value is non-zero, and timercmp compares two time
values (beware that => and <= do not work with this macro).

RETURN VALUE
If the calls succeed, a value of 0 is returned. If an error occurs, the value -1 is returned, and
a more precise error code is placed in the global variable errno.
ERRORS
The possible errors are:

[EFAULT] The value parameter specified a bad address.
[EINVAL] A value parameter specified a time was too large to be handled.

SEE ALSO
sigvec(2), gettimeofday(2)
NAME
getpagesize – get system page size

SYNOPSIS
pagesize = getpagesize()
int pagesize;

DESCRIPTION
getpagesize returns the number of bytes in a page. Page granularity is the granularity of many
of the memory management calls.
The page size is a system page size and may not be the same as the underlying hardware page
size.

SEE ALSO
sbrk(2), pagesize(1)
NAME
getpeername – get name of connected peer

SYNOPSIS
getpeername(s, name, namelen)
int s;
struct sockaddr *name;
int *namelen;

DESCRIPTION
getpeername returns the name of the peer connected to socket s. The namelen parameter should be initialized to indicate the amount of space pointed to by name. On return it contains the actual size of the name returned (in bytes). The name is truncated if the buffer provided is too small.

DIAGNOSTICS
A 0 is returned if the call succeeds, –1 if it fails.

ERRORS
The call succeeds unless:
[EBADF] The argument s is not a valid descriptor.
[ENOTSOCK] The argument s is a file, not a socket.
[ENOTCONN] The socket is not connected.
[ENOBUFS] Insufficient resources were available in the system to perform the operation.
[EFAULT] The name parameter points to memory not in a valid part of the process address space.

SEE ALSO
accept(2), bind(2), socket(2), getsockname(2)
NAME
getpgrp – get process group

SYNOPSIS

```c
pgrp = getpgrp(pid)
int pgrp;
int pid;
```

DESCRIPTION

The process group of the specified process is returned by `getpgrp`. If `pid` is zero, then the call applies to the current process.

Process groups are used for distribution of signals, and by terminals to arbitrate requests for their input: processes that have the same process group as the terminal are foreground and may read, while others will block with a signal if they attempt to read.

This call is thus used by programs such as `csh(1)` to create process groups in implementing job control. The TIOCGRPGRP and TIOCSPGRP calls described in `tty(4)` are used to get/set the process group of the control terminal.

SEE ALSO

`setpgrp(2)`, `getuid(2)`, `tty(4)`
NAME
getpid, getppid - get process identification

SYNOPSIS
pid = getpid()
int pid;

ppid = getppid()
int ppid;

DESCRIPTION
getpid returns the process ID of the current process. Most often it is used to generate
uniquely-named temporary files.

getpid returns the process ID of the parent of the current process.

SEE ALSO
gethostid(2)
NAME
getpriority, setpriority — get/set program scheduling priority

SYNOPSIS
#include <sys/resource.h>

prio = getpriority(which, who)
int prio, which, who;

setpriority(which, who, prio)
int which, who, prio;

DESCRIPTION
The scheduling priority of the process, process group, or user, as indicated by which and who
is obtained with the getpriority call and set with the setpriority call. which is one of
PRIO_PROCESS, PRIO_PGRP, or PRIO_USER, and who is interpreted relative to which (a pro-
cess identifier for PRIO_PROCESS, process group identifier for PRIO_PGRP, and a user ID for
PRIO_USER). A zero value of who denotes the current process, process group, or user. prio
is a value in the range −20 to 20. The default priority is 0; lower priorities cause more favor-
able scheduling.

The getpriority call returns the highest priority (lowest numerical value) enjoyed by any of the
specified processes. The setpriority call sets the priorities of all of the specified processes to
the specified value. Only the super-user may lower priorities.

RETURN VALUE
Since getpriority can legitimately return the value −1, it is necessary to clear the external vari-
able errno prior to the call, then check it afterward to determine if a −1 is an error or a legiti-
mate value. The setpriority call returns 0 if there is no error, or −1 if there is.

ERRORS
getpriority and setpriority may return one of the following errors:
[ESRCH] No process was located using the which and who values specified.
[EINVAL] which was not one of PRIO_PROCESS, PRIO_PGRP, or
PRIO_USER.

In addition to the errors indicated above, setpriority may fail with one of the following errors
returned:
[EPERM] A process was located, but neither its effective nor real user ID
matched the effective user ID of the caller.
[EACCES] A non super-user attempted to lower a process priority.

SEE ALSO
nice(1), fork(2), renice(8)
NAME
getrlimit, setrlimit – control maximum system resource consumption

SYNOPSIS
#include <sys/time.h>
#include <sys/resource.h>
getrlimit(resource, rlp)
int resource;
struct rlimit *rlp;
setrlimit(resource, rlp)
int resource;
struct rlimit *rlp;

DESCRIPTION
Limits on the consumption of system resources by the current process and each process it creates may be obtained with the getrlimit call, and set with the setrlimit call.

The resource parameter is one of the following:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLIMIT_CPU</td>
<td>the maximum amount of cpu time (in seconds) to be used by each process.</td>
</tr>
<tr>
<td>RLIMITFSIZE</td>
<td>the largest size, in bytes, of any single file that may be created.</td>
</tr>
<tr>
<td>RLIMITDATA</td>
<td>the maximum size, in bytes, of the data segment for a process; this defines how far a program may extend its break with the sbrk(2) system call.</td>
</tr>
<tr>
<td>RLIMITSTACK</td>
<td>the maximum size, in bytes, of the stack segment for a process; this defines how far a program's stack segment may be extended. Stack extension is performed automatically by the system.</td>
</tr>
<tr>
<td>RLIMITCORE</td>
<td>the largest size, in bytes, of a core file that may be created.</td>
</tr>
<tr>
<td>RLIMITRSS</td>
<td>the maximum size, in bytes, to which a process's resident set size may grow. This imposes a limit on the amount of physical memory to be given to a process; if memory is tight, the system will prefer to take memory from processes that are exceeding their declared resident set size.</td>
</tr>
</tbody>
</table>

A resource limit is specified as a soft limit and a hard limit. When a soft limit is exceeded a process may receive a signal (for example, if the cpu time is exceeded), but it will be allowed to continue execution until it reaches the hard limit (or modifies its resource limit). The rlimit structure is used to specify the hard and soft limits on a resource,

```c
struct rlimit {
    int rlim_cur;    /* current (soft) limit */
    int rlim_max;    /* hard limit */
};
```

Only the super-user may raise the maximum limits. Other users may only alter rlim_cur within the range from 0 to rlim_max or (irreversibly) lower rlim_max.

An "infinite" value for a limit is defined as RLIM_INFINITY (0x7fffffffff).

Because this information is stored in the per-process information, this system call must be executed directly by the shell if it is to affect all future processes created by the shell; limit is thus a built-in command to csh(1).
The system refuses to extend the data or stack space when the limits would be exceeded in the normal way: a break call fails if the data space limit is reached. When the stack limit is reached, the process receives a segmentation fault (SIGSEGV); if this signal is not caught by a handler using the signal stack, this signal will kill the process.

A file I/O operation that would create a file that is too large will cause a signal SIGXFSZ to be generated; this normally terminates the process, but may be caught. When the soft cpu time limit is exceeded, a signal SIGXCPU is sent to the offending process.

RETURN VALUE
A 0 return value indicates that the call succeeded, changing or returning the resource limit. A return value of −1 indicates that an error occurred, and an error code is stored in the global location errno.

ERRORS
The possible errors are:

[EFAULT] The address specified for rlp is invalid.
[EPERM] The limit specified to setrlimit would have raised the maximum limit value, and the caller is not the super-user.

SEE ALSO
csh(1), quota(2), sigvec(2), sigstack(2)

WARNINGS
There should be limit and unlimit commands in sh(1) as well as in csh.
NAME

getrusage – get information about resource utilization

SYNOPSIS

#include <sys/time.h>
#include <sys/resource.h>

#define RUSAGE_SELF 0 /* calling process */
#define RUSAGE_CHILDREN -1 /* terminated child processes */

getrusage(who, rusage)
int who;
struct rusage *rusage;

mips_getrusage(who, rusage, rusage_size)
int who;
struct rusage *rusage;
int rusage_size;

DESCRIPTION

getrusage returns information describing the resources utilized by the current process, or all its
terminated child processes. mips_getrusage performs the same function as getrusage but takes
a third argument which is the size of the rusage structure. This interface will be used in the
future to return MIPS hardware specific resource use information as the rusage structure is
extended.

The who parameter is one of RUSAGE_SELF or RUSAGE_CHILDREN. The buffer to which
rusage points will be filled in with the following structure:

struct rusage {
    struct timeval ru_utime; /* user time used */
    struct timeval ru_stime; /* system time used */
    int ru_maxrss;
    int ru_ixrss; /* integral shared text memory size */
    int ru_idrss; /* integral unshared data size */
    int ru_isrss; /* integral unshared stack size */
    int ru_minflt; /* page reclaims */
    int ru_majflt; /* page faults */
    int ru_nswap; /* swaps */
    int ru_inblock; /* block input operations */
    int ru_oublock; /* block output operations */
    int ru_msgsnd; /* messages sent */
    int ru_msgrcv; /* messages received */
    int ru_nsignals; /* signals received */
    int ru_nvcsw; /* voluntary context switches */
    int ru_nivcsw; /* involuntary context switches */
};

The fields are interpreted as follows:

ru_utime the total amount of time spent executing in user mode.
ru_stime the total amount of time spent in the system executing on behalf of the
    process(es).
ru_maxrss the maximum resident set size utilized (in number of pages).
ru_ixrss an "integral" value indicating the amount of memory used by the text
    segment that was also shared among other processes. This value is
expressed in units of number of pages * seconds-of-execution and is calculated by summing the number of shared memory pages in use each time the internal system clock ticks and then averaging over 1 second intervals.

ru_idrss  
an integral value of the amount of unshared memory residing in the data segment of a process (expressed in units of number of pages * seconds-of-execution).

ru_isrss  
an integral value of the amount of unshared memory residing in the stack segment of a process (expressed in units of number of pages * seconds-of-execution).

ru_minflt  
the number of page faults serviced without any I/O activity; here I/O activity is avoided by "reclaiming" a page frame from the list of pages awaiting reallocation.

ru_majflt  
the number of page faults serviced that required I/O activity.

ru_nswap  
the number of times a process was "swapped" out of main memory.

ru_inblock  
the number of times the file system had to perform input.

ru_outblock  
the number of times the file system had to perform output.

ru_msgsnd  
the number of IPC messages sent.

ru_msgrcv  
the number of IPC messages received.

ru_nsignals  
the number of signals delivered.

ru_nvcsw  
the number of times a context switch resulted due to a process voluntarily giving up the processor before its time slice was completed (usually to await availability of a resource).

ru_nivcsw  
the number of times a context switch resulted due to a higher priority process becoming runnable or because the current process exceeded its time slice.

NOTES

The numbers ru_inblock and ru_outblock 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.

ERRORS

The possible errors for getrusage are:

[EINVAL]  
The who parameter is not a valid value.

[EFAULT]  
The address specified by the rusage parameter is not in a valid part of the process address space.

SEE ALSO

gmtimeofday(2), wait(2)

WARNING

There is no way to obtain information about a child process that has not yet terminated.
NAME
getsockopt - get socket name

SYNOPSIS
getsockopt(s, name, namelen)
int s;
struct sockaddr *name;
int *namelen;

DESCRIPTION
getsockopt returns the current name for the specified socket. The namelen parameter should be initialized to indicate the amount of space pointed to by name. On return it contains the actual size of the name returned (in bytes).

DIAGNOSTICS
A 0 is returned if the call succeeds, -1 if it fails.

ERRORS
The call succeeds unless:
[EBADF] The argument s is not a valid descriptor.
[ENOTSOCK] The argument s is a file, not a socket.
[ENOBUFS] Insufficient resources were available in the system to perform the operation.
[EFAULT] The name parameter points to memory not in a valid part of the process address space.

SEE ALSO
bind(2), socket(2)

WARNING
Names bound to sockets in the UNIX domain are inaccessible; getsockopt returns a zero length name.
NAME

getsockopt, setsockopt – get and set options on sockets

SYNOPSIS

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

getsockopt(s, level, optname, optval, optlen)
int s, level, optname;
char *optval;
int *optlen;

setsockopt(s, level, optname, optval, optlen)
int s, level, optname;
char *optval;
int optlen;

DESCRIPTION

getsockopt and setsockopt manipulate options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost “socket” level.

When manipulating socket options the level at which the option resides and the name of the option must be specified. To manipulate options at the “socket” level, level is specified as SOL_SOCKET. To manipulate options at any other level the protocol number of the appropriate protocol controlling the option is supplied. For example, to indicate that an option is to be interpreted by the TCP protocol, level should be set to the protocol number of TCP; see getprotoent(3N).

The parameters optval and optlen are used to access option values for setsockopt. For getsockopt they identify a buffer in which the value for the requested option(s) are to be returned. For getsockopt, optlen is a value-result parameter, initially containing the size of the buffer pointed to by optval, and modified on return to indicate the actual size of the value returned. If no option value is to be supplied or returned, optval may be supplied as 0.

optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file <sys/socket.h> contains definitions for “socket” level options, described below. Options at other protocol levels vary in format and name; consult the appropriate entries in section (4P).

Most socket-level options take an int parameter for optval. For setsockopt, the parameter should non-zero to enable a boolean option, or zero if the option is to be disabled. SO_LINGER uses a struct linger parameter, defined in <sys/socket.h>, which specifies the desired state of the option and the linger interval (see below).

The following options are recognized at the socket level. Except as noted, each may be examined with getsockopt and set with setsockopt.

- SO_DEBUG: toggle recording of debugging information
- SO_REUSEADDR: toggle local address reuse
- SO_KEEPALIVE: toggle keep connections alive
- SO_DONTROUTE: toggle routing bypass for outgoing messages
- SO_LINGER: linger on close if data
- SO_BROADCAST: toggle permission to transmit broadcast messages
- SO_OOBINLINE: toggle reception of out-of-band data in
- SO_SNDBUF: set buffer size for output
- SO_RCVBUF: set buffer size for input
- SO_TYPE: get the type of the socket
- SO_ERROR: get and clear error on the
SO_DEBUG enables debugging in the underlying protocol modules. SO_REUSEADDR indicates that the rules used in validating addresses supplied in a bind(2) call should allow reuse of local addresses. SO_KEEPALIVE enables the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified via a SIGPIPE signal. SO_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

SO_LINGER controls the action taken when unsent messages are queued on socket and a close(2) is performed. If the socket promises reliable delivery of data and SO_LINGER is set, the system will block the process on the close attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the setsockopt call when SO_LINGER is requested). If SO_LINGER is disabled and a close is issued, the system will process the close in a manner that allows the process to continue as quickly as possible.

The option SO_BROADCAST requests permission to send broadcast datagrams on the socket. Broadcast was a privileged operation in earlier versions of the system. With protocols that support out-of-band data, the SO_OOBINLINE option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with recv or read calls without the MSG_OOB flag. SO_SNDBUF and SO_RCVBUF are options to adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections, or may be decreased to limit the possible backlog of incoming data. The system places an absolute limit on these values. Finally, SO_TYPE and SO_ERROR are options used only with setsockopt. SO_TYPE returns the type of the socket, such as SOCK_STREAM; it is useful for servers that inherit sockets on startup. SO_ERROR returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

RETURN VALUE
A 0 is returned if the call succeeds, −1 if it fails.

ERRORS
The call succeeds unless:

[EBADF] The argument s is not a valid descriptor.
[ENOTSOC] The argument s is a file, not a socket.
[ENOPROTOOP] The option is unknown at the level indicated.
[EFAULT] The address pointed to by optval is not in a valid part of the process address space. For getsockopt, this error may also be returned if optlen is not in a valid part of the process address space.

SEE ALSO
ioctl(2), socket(2), getprotoent(3N)

WARNING
Several of the socket options should be handled at lower levels of the system.
NAME
getimeofday, settimeofday – get/set date and time

SYNOPSIS
#include <sys/time.h>
getimeofday(tp, tp)
struct timeval *tp;
struct timezone *tzp;
settimeofday(tp, tp)
struct timeval *tp;
struct timezone *tzp;

DESCRIPTION
The system’s notion of the current Greenwich time and the current time zone is obtained with
the getimeofday call, and set with the settimeofday call. 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 tzp is zero,
the time zone information will not be returned or set.

The structures pointed to by tp and tzp are defined in <sys/time.h> as:
struct timeval {
    long tv_sec;          /* seconds since Jan. 1, 1970 */
    long tv_usec;         /* and microseconds */
};

struct timezone {
    int tz_minuteswest;  /* of Greenwich */
    int tz_dsttime;      /* type of dst correction to apply */
};

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.

Only the super-user may set the time of day or time zone.

RETURN
A 0 return value indicates that the call succeeded. A -1 return value indicates an error
occurred, and in this case an error code is stored into the global variable errno.

ERRORS
The following error codes may be set in errno:
[EFAULT]   An argument address referenced invalid memory.
[EPERM]     A user other than the super-user attempted to set the time.

SEE ALSO
date(1), adjtime(2), ctime(3), timed(8)
NAME
getuid, geteuid - get user identity

SYNOPSIS
#include <sys/types.h>

uid = getuid()
uid_t uid;
euid = geteuid()
euid_t euid;

DESCRIPTION
getuid returns the real user ID of the current process, geteuid the effective user ID.
The real user ID identifies the person who is logged in. The effective user ID gives the process
additional permissions during execution of "set-user-ID" mode processes, which use getuid to
determine the real-user-id of the process that invoked them.

SEE ALSO
getgid(2), setreuid(2)
NAME
hwconf – get or set hardware configuration information

SYNOPSIS
#include <machine/hwconf.h>

hwconf(option, conf)
int option;
struct hw_config *conf;

DESCRIPTION
The hwconf system call allows a user process to get or set hardware configuration information.
The specific contents of the hardware configuration structure is dependent upon the particular
release of the kernel and the hardware configuration, but typical contents include MIPS chip
types and revision numbers, MIPS board types and revision numbers and serial numbers, and
non-volatile RAM, NVRAM, environment variables names and values.

option indicates whether the hardware configuration information should be retrieved or
modified.

option may be one of:
HWCONF_GET Return the hardware configuration information
HWCONF_SET Set the specified NVRAM environment variable to the value indicated in
conf. To use this option, call hwconf with the HWCONF_GET option,
modify the value for the desired NVRAM variable, and then call hwconf
with the HWCONF_SET option. Must be super-user.

RETURN VALUE
hdwconf returns the a -1 on failure with errno set to the specific error.

ERRORS
[EINVAL] option is not one of HWCONF_GET or HWCONF_SET.
[EFAULT] conf is not accessible.
[EACCES] Attempt to modify NVRAM environment variable when not super-user.

SEE ALSO
hwconf(8)
“System Programmer's Guide”

WARNING
MIPS memory board idprom information should be added.
NAME
intro – introduction to system calls and error numbers

SYNOPSIS
#include <sys/errno.h>

DESCRIPTION
This section describes all of the system calls. Most of these calls have one or more error
returns. An error condition is indicated by an otherwise impossible return value. This is
almost always -1; the individual descriptions specify the details. Note that a number of sys-
tem calls overload the meanings of these error numbers, and that the meanings must be inter-
preted according to the type and circumstances of the call.

As with normal arguments, all return codes and values from functions are of type integer
unless otherwise noted. An error number is also made available in the external variable errno,
which is not cleared on successful calls. Thus errno should be tested only after an error has
occurred.

The following is a complete list of the errors and their names as given in <sys/errno.h>.
Unused. Typically this error indicates an attempt to modify a file in some way forbidden
except to its owner or super-user. It is also returned for attempts by ordinary users to do
things allowed only to the super-user. This error occurs when a file name is specified and the
file should exist but doesn’t, or when one of the directories in a path name does not exist.
The process or process group whose number was given does not exist, or any such process is
already dead. An asynchronous signal (such as interrupt or quit) that the user has elected to
catch occurred during a system call. If execution is resumed after processing the signal and
the system call is not restarted, it will appear as if the interrupted system call returned this
error condition. Some physical I/O error occurred during a read or write. This error may in
some cases occur on a call following the one to which it actually applies. I/O on a special file
refers to a subdevice that does not exist, or beyond the limits of the device. It may also occur
when, for example, an illegal tape drive unit number is selected or a disk pack is not loaded
on a drive. An argument list longer than 20480 bytes (or the current limit, NCARGS in
<sys/param.h>) is presented to execve. A request is made to execute a file that, although it
has the appropriate permissions, does not start with a valid magic number, (see a.out(5)).
Either a file descriptor refers to no open file, or a read (resp. write) request is made to a file
that is open only for writing (resp. reading). wait and the process has no living or unwaited-
for children. In a fork, the system’s process table is full or the user is not allowed to create
any more processes. During an execve or break, a program asks for more core or swap space
than the system is able to supply, or a process size limit would be exceeded. A lack of swap
space is normally a temporary condition; however, a lack of core is not a temporary condi-
tion; the maximum size of the text, data, and stack segments is a system parameter. Soft lim-
its may be increased to their corresponding hard limits. An attempt was made to access a file
in a way forbidden by the protection system. The system encountered a hardware fault in
attempting to access the arguments of a system call. A plain file was mentioned where a
block device was required, e.g., in mount. An attempt to mount a device that was already
mounted or an attempt was made to dismount a device on which there is an active file (open
file, current directory, mounted-on file, or active text segment). A request was made to an
exclusive access device that was already in use. An existing file was mentioned in an inap-
propriate context, e.g., link. A hard link to a file on another device was attempted. An
attempt was made to apply an inappropriate system call to a device, e.g., to read a write-only
device, or the device is not configured by the system. A non-directory was specified where a
directory is required, for example, in a path name or as an argument to chdir. An attempt to
write on a directory. Some invalid argument: dismounting a non-mounted device, mentioning
an unknown signal in signal, or some other argument inappropriate for the call. Also set by
math functions, (see math(3)). The system’s table of open files is full, and temporarily no
more *opens* can be accepted. As released, the limit on the number of open files per process is 64. *getdtablesize*(2) will obtain the current limit. Customary configuration limit on most other UNIX systems is 20 per process. The file mentioned in an *ioctl* is not a terminal or one of the devices to which this call applies. An attempt to execute a pure-procedure program that is currently open for writing. Also an attempt to open for writing a pure-procedure program that is being executed. The size of a file exceeded the maximum (about 2^31 bytes). A *write* to an ordinary file, the creation of a directory or symbolic link, or the creation of a directory entry failed because no more disk blocks are available on the file system, or the allocation of an inode for a newly created file failed because no more inodes are available on the file system. An *lseek* was issued to a socket or pipe. This error may also be issued for other non-seeable devices. An attempt to modify a file or directory was made on a device mounted read-only. An attempt to make more than 32767 hard links to a file. A *write* on a pipe or socket for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is caught or ignored. The argument of a function in the math package (3M) is out of the domain of the function. The value of a function in the math package (3M) is unrepresentable within machine precision. An operation that would cause a process to block was attempted on an object in non-blocking mode (see *fcntl*(2)). An operation that takes a long time to complete (such as a *connect*(2)) was attempted on a non-blocking object (see *fcntl*(2)). An operation was attempted on a non-blocking object that already had an operation in progress. Self-explanatory. A required address was omitted from an operation on a socket. A message sent on a socket was larger than the internal message buffer or some other network limit. A protocol was specified that does not support the semantics of the socket type requested. For example, you cannot use the ARPA Internet UDP protocol with type SOCK_STREAM. A bad option or level was specified in a *getsockopt*(2) or *setsockopt*(2) call. The protocol has not been configured into the system or no implementation for it exists. The support for the socket type has not been configured into the system or no implementation for it exists. For example, trying to *accept* a connection on a datagram socket. The protocol family has not been configured into the system or no implementation for it exists. An address incompatible with the requested protocol was used. For example, you shouldn't necessarily expect to be able to use NS addresses with ARPA Internet protocols. Only one usage of each address is normally permitted. Normally results from an attempt to create a socket with an address not on this machine. A socket operation encountered a dead network. A socket operation was attempted to an unreachable network. The host you were connected to crashed and rebooted. A connection abort was caused internal to your host machine. A connection was forcibly closed by a peer. This normally results from a loss of the connection on the remote socket due to a timeout or a reboot. An operation on a socket or pipe was not performed because the system lacked sufficient buffer space or because a queue was full. A *connect* request was made on an already connected socket; or, a *sendto* or *sendmsg* request on a connected socket specified a destination when already connected. An request to send or receive data was disallowed because the socket is not connected and (when sending on a datagram socket) no address was supplied. A request to send data was disallowed because the socket had already been shut down with a previous *shutdown*(2) call. A *connect* or *send* request failed because the connected party did not properly respond after a period of time. (The timeout period is dependent on the communication protocol.) No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host. A path name lookup involved more than 8 symbolic links. A component of a path name exceeded 255 (MAXNAMELEN) characters, or an entire path name exceeded 1023 (MAXPATHLEN-1) characters. A socket operation failed because the destination host was down. A socket operation was attempted to an unreachable host. A directory with entries other than "." and ".." was supplied to a remove directory or rename call. A *write* to an ordinary file, the creation of a directory or symbolic link, or the creation of a directory entry failed because the
user's quota of disk blocks was exhausted, or the allocation of an inode for a newly created file failed because the user's quota of inodes was exhausted. A client referenced an open file, but the file has been deleted. An attempt was made to remotely mount a file system into a path which already has a remotely mounted component.

DEFINITIONS

**Process ID**
Each active process in the system is uniquely identified by a positive integer called a process ID. The range of this ID is from 0 to 30000.

**Parent process ID** A new process is created by a currently active process; (see *fork*(2)). The parent process ID of a process is the process ID of its creator.

**Process Group ID** Each active process is a member of a process group that is identified by a positive integer called the process group ID. This is the process ID of the group leader. This grouping permits the signaling of related processes (see *killpg*(2)) and the job control mechanisms of *csh*(1).

**Tty Group ID** Each active process can be a member of a terminal group that is identified by a positive integer called the tty group ID. This grouping is used to arbitrate between multiple jobs contending for the same terminal; (see *csh*(1) and *tty*(4)).

**Real User ID and Real Group** Each user on the system is identified by a positive integer termed the real user ID.

Each user is also a member of one or more groups. One of these groups is distinguished from others and used in implementing accounting facilities. The positive integer corresponding to this distinguished group is termed the real group ID.

All processes have a real user ID and real group ID. These are initialized from the equivalent attributes of the process that created it.

**Effective User Id, Effective Group Id,** Access to system resources is governed by three values: the effective user ID, the effective group ID, and the group access list.

The effective user ID and effective group ID are initially the process's real user ID and real group ID respectively. Either may be modified through execution of a set-user-ID or set-group-ID file (possibly by one of its ancestors) (see *execve*(2)).

The group access list is an additional set of group ID's used only in determining resource accessibility. Access checks are performed as described below in “File Access Permissions”.

**Super-user** A process is recognized as a super-user process and is granted special privileges if its effective user ID is 0.

**Special Processes** The processes with a process ID's of 0, 1, and 2 are special. Process 0 is the scheduler. Process 1 is the initialization process *init*, and is the ancestor of every other process in the system. It is used to control the process structure. Process 2 is the paging daemon.

**Descriptor** An integer assigned by the system when a file is referenced by *open*(2) or *dup*(2), or when a socket is created by *pipe*(2), *socket*(2) or *socketpair*(2), which uniquely identifies an
access path to that file or socket from a given process or any of its children.

**File Name** Names consisting of up to 255 (MAXNAMELEN) characters may be used to name an ordinary file, special file, or directory.

These characters may be selected from the set of all ASCII character excluding 0 (null) and the ASCII code for / (slash). (The parity bit, bit 8, must be 0.)

Note that it is generally unwise to use *, ?, [ or ] as part of file names because of the special meaning attached to these characters by the shell.

**Path Name** A path name is a null-terminated character string starting with an optional slash (/), followed by zero or more directory names separated by slashes, optionally followed by a file name. The total length of a path name must be less than 1024 (MAXPATHLEN) characters.

If a path name begins with a slash, the path search begins at the root directory. Otherwise, the search begins from the current working directory. A slash by itself names the root directory. A null pathname refers to the current directory.

**Directory** A directory is a special type of file that contains entries that are references to other files. Directory entries are called links. By convention, a directory contains at least two links, . and ..., referred to as dot and dot-dot respectively. Dot refers to the directory itself and dot-dot refers to its parent directory.

**Root Directory and Current Working Directory** Each process has associated with it a concept of a root directory and a current working directory for the purpose of resolving path name searches. A process's root directory need not be the root directory of the root file system.

**File Access Permissions** Every file in the file system has a set of access permissions. These permissions are used in determining whether a process may perform a requested operation on the file (such as opening a file for writing). Access permissions are established at the time a file is created. They may be changed at some later time through the chmod(2) call.

File access is broken down according to whether a file may be: read, written, or executed. Directory files use the execute permission to control if the directory may be searched.

File access permissions are interpreted by the system as they apply to three different classes of users: the owner of the file, those users in the file's group, anyone else. Every file has an independent set of access permissions for each of these classes. When an access check is made, the system decides if permission should be granted by checking the access information applicable to the caller.

Read, write, and execute/search permissions on a file are granted to a process if:

The process's effective user ID is that of the super-user.

The process's effective user ID matches the user ID of the owner of the file and the owner permissions allow the access.

The process's effective user ID does not match the user ID of the owner of the file, and either the process's effective group ID matches the group ID of the file, or the group ID of the file is in the process's group access list, and the group permissions allow the access.

Neither the effective user ID nor effective group ID and group access list of the process match the corresponding user ID and group ID of the file, but the permissions for "other users" allow access.
Otherwise, permission is denied.

**Sockets and Address Families** A socket is an endpoint for communication between processes. Each socket has queues for sending and receiving data.

Sockets are typed according to their communications properties. These properties include whether messages sent and received at a socket require the name of the partner, whether communication is reliable, the format used in naming message recipients, etc.

Each instance of the system supports some collection of socket types; consult `socket(2)` for more information about the types available and their properties.

Each instance of the system supports some number of sets of communications protocols. Each protocol set supports addresses of a certain format. An Address Family is the set of addresses for a specific group of protocols. Each socket has an address chosen from the address family in which the socket was created.

**SEE ALSO**
intro(3), perror(3)
NAME
iocl - control device

SYNOPSIS
#include <sys/ioctl.h>

ioctl(d, request, argp)
int d;
unsigned long request;
char *argp;

DESCRIPTION
ioctl performs a variety of functions on open descriptors. In particular, many operating
characteristics of character special files (e.g., terminals) may be controlled with ioctl requests.
The writeups of various devices in section 4 discuss how ioctl applies to them.

An ioctl request has encoded in it whether the argument is an "in" parameter or "out"
parameter, and the size of the argument argp in bytes. Macros and defines used in specifying
an ioctl request are located in the file <sys/ioctl.h>.

RETURN VALUE
If an error has occurred, a value of -1 is returned and errno is set to indicate the error.

ERRORS
ioctl will fail if one or more of the following are true:
[EBAFD] D is not a valid descriptor.
[ENOTTY] D is not associated with a character special device.
[EINVAL] The specified request does not apply to the kind of object that the
descriptor d references.

SEE ALSO
execve(2), fcntl(2), mt(4), tty(4), intro(4N), ad(4), arp(4), bk(4), de(4), dmc(4), ec(4), en(4),
ex(4), hy(4), ik(4), il(4), imp(4), inet(4F), ix(4), lo(4), mtio(4), np(4), pcl(4), ps(4), pty(4),
qe(4), rx(4), tb(4), un(4), uu(4), va(4), vp(4), vv(4)
NAME
kill - send signal to a process

SYNOPSIS
kill(pid, sig)
int pid, sig;

DESCRIPTION
kill sends the signal sig to a process, specified by the process number pid. sig may be one of
the signals specified in sigvec(2), or it may be 0, in which case error checking is performed but
no signal is actually sent. This can be used to check the validity of pid.

The sending and receiving processes must have the same effective user ID, otherwise this call
is restricted to the super-user. A single exception is the signal SIGCONT, which may always be
sent to any descendant of the current process.

If the process number is 0, the signal is sent to all processes in the sender's process group;
this is a variant of killpg(2).

If the process number is -1 and the user is the super-user, the signal is broadcast universally
except to system processes and the process sending the signal. If the process number is -1 and
the user is not the super-user, the signal is broadcast universally to all processes with the
same uid as the user except the process sending the signal. No error is returned if any process
could be signaled.

For compatibility with System V, if the process number is negative but not -1, the signal is
sent to all processes whose process group ID is equal to the absolute value of the process
number. This is a variant of killpg(2).

Processes may send signals to themselves.

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned
and errno is set to indicate the error.

ERRORS
kill will fail and no signal will be sent if any of the following occur:

[EINVAL] sig is not a valid signal number.
[ESRCH] No process can be found corresponding to that specified by pid.
[ESRCH] The process id was given as 0 but the sending process does not have a
process group.
[EPERM] The sending process is not the super-user and its effective user id does
not match the effective user-id of the receiving process. When signaling
a process group, this error was returned if any members of the group
could not be signaled.

SEE ALSO
getpid(2), getpgprp(2), killpg(2), sigvec(2)
NAME
killpg – send signal to a process group

SYNOPSIS

    killpg(pgrp, sig)
    int pgrp, sig;

DESCRIPTION

killpg sends the signal sig to the process group pgrp. See sigvec(2) for a list of signals.

The sending process and members of the process group must have the same effective user ID, or the sender must be the super-user. As a single special case the continue signal SIGCONT may be sent to any process that is a descendant of the current process.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the global variable errno is set to indicate the error.

ERRORS

killpg will fail and not signal will be sent if any of the following occur:

[EINVAL] Sig is not a valid signal number.
[ESRCH] No process can be found in the process group specified by pgrp.
[ESRCH] The process group was given as 0 but the sending process does not have a process group.
[EPERM] The sending process is not the super-user and one or more of the target processes has an effective user ID different from that of the sending process.

SEE ALSO

kill(2), getpgrp(2), sigvec(2)
NAME
kopt – get or set kernel options

SYNOPSIS
#include <mips/debug.h>

kopt(option, value, op)
char *option;
int value, op;

DESCRIPTION
The kopt system call allows a user process to get or set kernel options. The specific set of options is dependent upon the particular release of the kernel, but typical options control virtual memory system parameters, debugging options, and device driver options.

option points to a null-terminated character string naming a kernel option. The current set of kernel options is specified by the array kernargs in the kernel source file mips/kopt.c.

op may be one of:
KOPT_GET Return the specified option
KOPT_SET Set the specified option to value. Must be super-user.
KOPT_BIS Or the bits in value into the specified option. Must be super-user.
KOPT_BIC Clear the bits in value from the specified option. Must be super-user.

RETURN VALUE
kopt returns the previous value of the specified option on success, or -1 on failure. Since -1 is a legal value for many kernel options, errors must be disambiguated from successful returns of -1 by the value of errno.

ERRORS
[EINVAL] option name is too long.
[EINVAL] option is not known kernel option.
[EINVAL] op is not one of KOPT_GET, KOPT_SET, KOPT_BIS, or KOPT_BIC.
[EFAULT] option is not accessible.
[EACCES] Attempt to modify kernel option when not super-user.

SEE ALSO
kopt(8)
NAME

link – make a hard link to a file

SYNOPSIS

link(name1, name2)

char *name1, *name2;

DESCRIPTION

A hard link to name1 is created; the link has the name name2. name1 must exist.

With hard links, both name1 and name2 must be in the same file system. Unless the caller is the super-user, name1 must not be a directory. Both the old and the new link share equal access and rights to the underlying object.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

ERRORS

link will fail and no link will be created if one or more of the following are true:

[ENOTDIR]  A component of either path prefix is not a directory.

[EINVAL]  Either pathname contains a character with the high-order bit set.

[ENAMETOOLONG]  A component of either pathname exceeded 255 characters, or entire length of either path name exceeded 1023 characters.

[ENOENT]  A component of either path prefix does not exist.

[EACCES]  A component of either path prefix denies search permission.

[EACCES]  The requested link requires writing in a directory with a mode that denies write permission.

[ELOOP]  Too many symbolic links were encountered in translating one of the pathnames.

[ENOENT]  The file named by name1 does not exist.

[EINVAL]  The link named by name2 does exist.

[EPERM]  The file named by name1 is a directory and the effective user ID is not super-user.

[EXDEV]  The link named by name2 and the file named by name1 are on different file systems.

[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.

[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.

[EIO]  An I/O error occurred while reading from or writing to the file system to make the directory entry.

[EROFS]  The requested link requires writing in a directory on a read-only file system.

[EFAULT]  One of the pathnames specified is outside the process’s allocated address space.
SEE ALSO
  symlink(2), unlink(2)
NAME
listen – listen for connections on a socket

SYNOPSIS
listen(s, backlog)
int s, backlog;

DESCRIPTION
To accept connections, a socket is first created with socket(2), a willingness to accept incoming connections and a queue limit for incoming connections are specified with listen(2), and then the connections are accepted with accept(2). The listen call applies only to sockets of type SOCK_STREAM or SOCK_SEQPACKET.

The backlog parameter defines the maximum length the queue of pending connections may grow to. If a connection request arrives with the queue full the client may receive an error with an indication of ECONNREFUSED, or, if the underlying protocol supports retransmission, the request may be ignored so that retries may succeed.

RETURN VALUE
A 0 return value indicates success; -1 indicates an error.

ERRORS
The call fails if:
[EBADF] The argument s is not a valid descriptor.
[ENOTSOCK] The argument s is not a socket.
[EOPNOTSUPP] The socket is not of a type that supports the operation listen.

SEE ALSO
accept(2), connect(2), socket(2)

WARNING
The backlog is currently limited (silently) to 5.
NAME
lseek – move read/write pointer

SYNOPSIS
#include <sys/file.h>
#define L_SET 0 /* set the seek pointer */
#define L_INCR 1 /* increment the seek pointer */
#define L_XTND 2 /* extend the file size */

pos = lseek(d, offset, whence)
off_t pos;
int d;
off_t offset;
int whence;

DESCRIPTION
The descriptor d refers to a file or device open for reading and/or writing. lseek sets the file
pointer of d as follows:

If whence is L_SET, the pointer is set to offset bytes.
If whence is L_INCR, the pointer is set to its current location plus offset.
If whence is L_XTND, the pointer is set to the size of the file plus offset.

Upon successful completion, the resulting pointer location as measured in bytes from begin-
ing of the file is returned. Some devices are incapable of seeking. The value of the pointer
associated with such a device is undefined.

NOTES
Seeking far beyond the end of a file, then writing, creates a gap or “hole”, which occupies no
physical space and reads as zeros.

RETURN VALUE
Upon successful completion, the current file pointer value is returned. Otherwise, a value of
–1 is returned and errno is set to indicate the error.

ERRORS
lseek will fail and the file pointer will remain unchanged if:
[EBADF] fildes is not an open file descriptor.
[EINVAL] fildes is associated with a pipe or a socket.
[EINVAL] whence is not a proper value.

SEE ALSO
dup(2), open(2)

WARNING
This document's use of whence is incorrect English, but maintained for historical reasons.
NAME
mipsfpu – enabling and disabling the floating-point unit

SYNOPSIS
int
mipsfpu(x)
int x;

DESCRIPTION
This system call is used to enable and disable the floating-point unit. An non-zero argument
enables and a zero argument disables the floating-point unit. When disabled the system emu-
lates all instructions in software. This can only be executed by the super-user.

ERRORS
mipsfpu fails when the following occurs:
EPERM The caller is not the super-user.

WARNING
If you disable a floating-point unit which produces imprecise exceptions (the R2360) just as a
program using the floating-point unit is handling a signal which is trying to retrieve the
floating-point instruction causing the signal based on the floating-point unit’s implementation
revision register that program will fail to get the floating-point instruction that caused the sig-
nal. This is because the implementation revision register changed between the time the
instruction causing the signal was executed and the time signal handler handled it.
NAME
mknod – make a special file

SYNOPSIS
mknod(path, mode, dev)
char *path;
int mode, dev;

DESCRIPTION
mknod creates a new file whose name is path. The mode of the new file (including special file
bits) is initialized from mode. (The protection part of the mode is modified by the process's
mode mask (see umask(2))). The first block pointer of the i-node is initialized from dev and
is used to specify which device the special file refers to.

If mode indicates a block or character special file, dev is a configuration dependent
specification of a character or block I/O device. If mode does not indicate a block special or
character special device, dev is ignored.

mknod may be invoked only by the super-user.

RETURN VALUE
Upon successful completion a value of 0 is returned. Otherwise, a value of –1 is returned and
errno is set to indicate the error.

ERRORS
mknod will fail and the file mode will be unchanged if:

[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 255 characters, or an entire path
name exceeded 1023 characters.

[ENOENT] A component of the path prefix does not exist.

[EACCESS] Search permission is denied for a component of the path prefix.

[ELOOP] Too many symbolic links were encountered in translating the pathname.

[EPERM] The process's effective user ID is not super-user.

[EPERM] The pathname contains a character with the high-order bit set.

[EIO] An I/O error occurred while making the directory entry or allocating the
inode.

[ENOSPC] The directory in which the entry for the new node is being placed cannot be
extended because there is no space left on the file system con-
taining the directory.

[ENOSPC] There are no free inodes on the file system on which the node is being
created.

[EDQUOT] The directory in which the entry for the new node is being placed cannot
be extended because the user's quota of disk blocks on the file sys-
tem containing the directory has been exhausted.

[EDQUOT] The user's quota of inodes on the file system on which the node is being
created has been exhausted.

[EROFS] The named file resides on a read-only file system.

[EREPLACE] The named file exists.

[EFAULT] path points outside the process’s allocated address space.
SEE ALSO
    chmod(2), stat(2), umask(2)
NAME
mmap, munmap — map or unmap pages of memory

SYNOPSIS
#include <sys/mman.h>
#include <sys/types.h>
mmap(addr, len, prot, share, fd, off)
caddr_t addr;
int len, prot, share, fd;
off_t off;
munmap (addr, len)
caddr_t addr;
int len;

DESCRIPTION
mmap maps pages of memory from the memory device associated with the file fd into the
address space of the calling process, one page at a time. Pages are mapped from the memory
device, beginning at off, and into the caller's address space, beginning at addr, and continuing
for len bytes. fd is a file descriptor obtained by opening the device from which to map pages.
Only character-special devices are currently supported.

share specifies whether modifications made to mapped-in copies of pages are to be kept
"private" or are to be "shared" with other references. Currently, it must be set to
MAP_SHARED.

The parameter prot specifies the read/write accessibility of the mapped pages. The addr and
len parameters, and the sum of the current position in fd and off parameters, must be multi-
plies of pagesize (found using the getpagesize(2) call). malloc(2) returns a properly aligned
buffer if the request is for pagesize or larger bytes.

Currently, only 1 device may be mapped by a process. The file descriptor must be closed to
allow mapping of another device.

All pages are automatically unmapped when fd is closed. Specific pages can be unmapped
explicitly using munmap.

mmap can sometimes be used to install memory-mapped devices without writing a device
driver. However, this does not always work. In particular, devices that are mmap'ed into
user space and then accessed by user programs will see those accesses in user mode. If the
device contains registers that must be accessed in supervisor mode, mmap cannot be used to
drive it.

munmap unmaps previously mapped pages starting at addr and continuing for len bytes.
Unmapped pages refer, once again, to private pages within the caller's address space.
Unmapped pages are initialized to zero.

RETURN VALUE
Each call returns 0 on success, -1 on failure.

ERRORS
Both calls fail when:
EINVAL The argument address or length is not a multiple of the page size as
returned by getpagesize(2), or the length is negative.
EINVAL The entire range of pages specified in the call is not part of data space.
In addition mmap fails when:
EINVAL The specified fd does not refer to a character special device which
supports mapping (e.g. a frame buffer).

EINVAL
The specified *fd* is not open for reading and read access is requested, or not open for writing when write access is requested.

EINVAL
The sharing mode was not specified as MAP_SHARED.

EINVAL
Another file mapped by *mmap* is open.

**SEE ALSO**
getpagesize(2), munmap(2), close(2), malloc(2)
NAME
mdir – make a directory file

SYNOPSIS
mdir(path, mode)
char *path;
int mode;

DESCRIPTION
mdir creates a new directory file with name path. The mode of the new file is initialized
from mode. (The protection part of the mode is modified by the process’s mode mask; see
umask(2)).

The directory’s owner ID is set to the process’s effective user ID. The directory’s group ID is
set to that of the parent directory in which it is created.

The low-order 9 bits of mode are modified by the process’s file mode creation mask: all bits
set in the process’s file mode creation mask are cleared. See umask(2).

RETURN VALUE
A 0 return value indicates success. A −1 return value indicates an error, and an error code is
stored in errno.

ERRORS
mdir will fail and no directory will be created if:

[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 255 characters, or an entire path
 name exceeded 1023 characters.

[ENOENT] A component of the path prefix 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 path argument contains a byte with the high-order bit set.

[EROFS] The named file resides on a read-only file system.

[EEXIST] The named file exists.

[ENOSPC] The directory in which the entry for the new directory is being placed
cannot be extended because there is no space left on the file system contain-
ing the directory.

[ENOSPC] The new directory cannot be created because there is no space left
on the file system that will contain the directory.

[ENOSPC] There are no free inodes on the file system on which the directory is
being created.

[EDQUOT] The directory in which the entry for the new directory is being placed
cannot be extended because the user’s quota of disk blocks on the file
system containing the directory has been exhausted.

[EDQUOT] The new directory cannot be created because the user’s quota of disk
blocks on the file system that will contain the directory has been
exhausted.

[EDQUOT] The user’s quota of inodes on the file system on which the directory is
being created has been exhausted.
MKDIR (2-BSD)  RISC/os Programmer's Reference  MKDIR (2-BSD)

[EIO]  An I/O error occurred while making the directory entry or allocating the inode.

[EIO]  An I/O error occurred while reading from or writing to the file system.

[EFAULT]  path points outside the process's allocated address space.

SEE ALSO
   chmod(2), stat(2), umask(2)
NAME
mount - mount file system

SYNOPSIS
#include <sys/mount.h>

mount(type, dir, flags, data)
int type;
char *dir;
int flags;
caddr_t data;

DESCRIPTION
mount attaches a file system to a directory. After a successful return, references to directory
dir will refer to the root directory on the newly mounted file system. dir is a pointer to a null-
terminated string containing a path name. dir must exist already, and must be a directory. Its
old contents are inaccessible while the file system is mounted.

mount may be invoked only by the super-user.

The flags argument determines whether the file system can be written on, and if set-uid execution
is allowed. 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.

type indicates the type of the filesystem. It must be one of the types defined in mount.h. data
is a pointer to a structure which contains the type specific arguments to mount. Below is a list
of the filesystem types supported and the type specific arguments to each:

MOUNT_UFS
struct ufs_args {
    char *fspec;     /* Block special file to mount */
};

MOUNT_NFS
#include <nfs/nfs.h>
#include <netinet/in.h>
struct nfs_args {
    struct sockaddr_in *addr;  /* file server address */
    fhhandle_t *fh;            /* File handle to be mounted */
    int flags;                /* flags */
    int wsize;                /* write size in bytes */
    int rsize;                /* read size in bytes */
    int timeout;              /* initial timeout in .1 secs */
    int retrans;              /* times to retry send */
};

RETURN VALUE
mount returns 0 if the action occurred, and -1 if fspec is inaccessible or not an appropriate
file, if name does not exist, if fspec is already mounted, if dir is in use, or if there are already
too many file systems mounted.

ERRORS
mount fails when one of the following occurs:
EPERM The caller is not the super-user.
ENOTBLK fspec is not a block device.
ENXIO The major device number of fspec is out of range (this indicates no dev-
ice driver exists for the associated hardware).
EBUSY  
.dir is not a directory, or another process currently holds a reference to it.

EBUSY  
No space remains in the mount table.

EBUSY  
The super block for the file system had a bad magic number or an out of range block size.

EBUSY  
Not enough memory was available to read the cylinder group information for the file system.

EIO  
An I/O error occurred while reading the super block or cylinder group information.

ENOTDIR  
A component of the path prefix in .fspec or .dir is not a directory.

EINVAL  
The path name of .fspec or .dir contains a character with the high-order bit set.

ENAMETOOLONG  
The length of a component of the path name of .fspec or .dir exceeds 255 characters, or the length of the entire path name of .fspec or .dir exceeds 1023 characters.

ENOENT  
.fsspec or .dir does not exist.

ENOTDIR  
The file named by .dir is not a directory.

EACCES  
Search permission is denied for a component of the path prefix of .fspec or .dir.

EFAULT  
.fspec or .dir points outside the process's allocated address space.

ELOOP  
Too many symbolic links were encountered in translating the path name of .fspec or .dir.

EIO  
An I/O error occurred while reading from or writing to the file system.

SEE ALSO  
unmount(2), mount(8)

WARNING  
The error codes are in a state of disarray; too many errors appear to the caller as one value.
NAME
  nfssvc, async_daemon - NFS daemons

SYNOPSIS
  nfssvc(sock)
  int sock;
  async_daemon()

DESCRIPTION
  nfssvc starts an NFS daemon listening on socket sock. The socket must be AF_INET, and
  SOCK_DGRAM (protocol UDP/IP). The system call will return only if the process is killed.

  async_daemon implements the NFS daemon that handles asynchronous I/O for an NFS client.
  The system call never returns.

WARNING
  These two system calls allow kernel processes to have user context.

SEE ALSO
  mountd(8)
NAME
open – open a file for reading or writing, or create a new file

SYNOPSIS
#include <sys/file.h>
open(path, flags, mode)
char *path;
int flags, mode;

DESCRIPTION
open opens the file path for reading and/or writing, as specified by the flags argument and
returns a descriptor for that file. The flags argument may indicate the file is to be created if it
does not already exist (by specifying the O_CREAT flag), in which case the file is created with
mode mode as described in chmod(2) and modified by the process’ umask value (see
umask(2)).

path is the address of a string of ASCII characters representing a path name, terminated by a
null character. The flags specified are formed by or’ing the following values

O_RDONLY open for reading only O_WRONLY open for writing only
O_RDWR  open for reading and writing O_NONBLOCK do not block on open
O_APPEND append on each write O_CREAT create file if it does not
O_TRUNC truncate size to 0 O_EXCL error if create and file exists

Opening a file with O_APPEND set causes each write on the file to be appended to the end. If
O_TRUNC is specified and the file exists, the file is truncated to zero length. If O_EXCL is set
with O_CREAT, then if the file already exists, the open returns an error. This can be used to
implement a simple exclusive access locking mechanism. If O_EXCL is set and the last com-
ponent of the pathname is a symbolic link, the open will fail even if the symbolic link points
to a non-existent name. If the O_NONBLOCK
flag is specified and the open call would result in the process being blocked for some reason
(e.g. waiting for carrier on a dialup line), the open returns immediately. The first time the pro-
cess attempts to perform i/o on the open file it will block (not currently implemented).

Upon successful completion a non-negative integer termed a file descriptor is returned. The
file pointer used to mark the current position within the file is set to the beginning of the file.
The new descriptor is set to remain open across execve system calls; see close(2).
The system imposes a limit on the number of file descriptors open simultaneously by one pro-
cess. getdtablesize(2) returns the current system limit.

ERRORS
The named file is opened unless one or more of the following are true:

[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 255 characters, or an entire path
name exceeded 1023 characters.
[ENOENT] O_CREAT is not set and the named file does not exist.
[ENOENT] A component of the path name that must exist does not exist.
[EACCES] Search permission is denied for a component of the path prefix.
[EACCES] The required permissions (for reading and/or writing) are denied for the
named flag.
[EACCES] O_CREAT is specified, the file does not exist, and the directory in which
it is to be created does not permit writing.
[ELOOP]  Too many symbolic links were encountered in translating the pathname.

[EISDIR]  The named file is a directory, and the arguments specify it is to be opened for writting.

[EROFS]  The named file resides on a read-only file system, and the file is to be modified.

[EMFILE]  The system limit for open file descriptors per process has already been reached.

[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.

[ENOSPC]  O_CREAT is specified, the file does not exist, and the directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory.

[ENOSPC]  O_CREAT is specified, the file does not exist, and there are no free inodes on the file system on which the file is being created.

[EDQUOT]  O_CREAT is specified, the file does not exist, and the directory in which the entry for the new file is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.

[EDQUOT]  O_CREAT is specified, the file does not exist, and the user's quota of inodes on the file system on which the file is being created has been exhausted.

[EIO]  An I/O error occurred while making the directory entry or allocating the inode for O_CREAT.

[ETXTBSY]  The file is a pure procedure (shared text) file that is being executed and the open call requests write access.

[EFAULT]  *path* points outside the process's allocated address space.

[EEXIST]  O_CREAT and O Excl were specified and the file exists.

[EOPNOTSUPP]  An attempt was made to open a socket (not currently implemented).

**SEE ALSO**

`chmod(2)`, `close(2)`, `dup(2)`, `getdtablesizer(2)`, `lseek(2)`, `read(2)`, `write(2)`, `umask(2)`
NAME
pipe – create an interprocess communication channel

SYNOPSIS
pipe(fildes)
int fildes[2];

DESCRIPTION
The pipe system call creates an I/O mechanism called a pipe. The file descriptors returned can be used in read and write operations. When the pipe is written using the descriptor fildes[1] up to 4096 bytes of data are buffered before the writing process is suspended. A read using the descriptor fildes[0] will pick up the data.

It is assumed that after the pipe has been set up, two (or more) cooperating processes (created by subsequent fork calls) will pass data through the pipe with read and write calls.

The shell has a syntax to set up a linear array of processes connected by pipes.

Read calls on an empty pipe (no buffered data) with only one end (all write file descriptors closed) returns an end-of-file.

Pipes are really a special case of the socketpair(2) call and, in fact, are implemented as such in the system.

A signal is generated if a write on a pipe with only one end is attempted.

RETURN VALUE
The function value zero is returned if the pipe was created; −1 if an error occurred.

ERRORS
The pipe call will fail if:
[EMFILE] Too many descriptors are active.
[ENFILE] The system file table is full.
[EFAULT] The fildes buffer is in an invalid area of the process’s address space.

SEE ALSO
sh(1), read(2), write(2), fork(2), socketpair(2)

WARNING
Should more than 4096 bytes be necessary in any pipe among a loop of processes, deadlock will occur.
NAME
profil – execution time profile

SYNOPSIS
profil(buff, bufsiz, offset, scale)
char *buff;
int bufsiz, offset, scale;

DESCRIPTION
buff points to an area of core whose length (in bytes) is given by bufsiz. After this call, the
user's program counter (pc) is examined each clock tick (10 milliseconds); offset is subtracted
from it, and the result multiplied by scale. If the resulting number corresponds to a word
inside buff, that word is incremented.

The scale is interpreted as an unsigned, fixed-point fraction with 16 bits of fraction: 0x10000
gives a 1-1 mapping of pc's to words in buff; 0x8000 maps each pair of instruction words
together.

Profiling is turned off by giving a scale of 0 or 1. It is rendered ineffective by giving a bufsiz of
0. Profiling is turned off when an execve is executed, but remains on in child and parent both
after a fork. Profiling is turned off if an update in buff would cause a memory fault.

RETURN VALUE
A 0, indicating success, is always returned.

SEE ALSO
gprof(1), setitimer(2), monitor(3)
NAME
ptrace - process trace

SYNOPSIS

#include <signal.h>
#include <sys/ptrace.h>

ptrace(request, pid, addr, data)
int request, pid, addr, data;

DESCRIPTION

ptrace provides a means by which a process may control the execution of another process, and examine and change its core image. Its primary use is for the implementation of breakpoint debugging. There are four arguments whose interpretation depends on a request argument. Generally, pid is the process ID of the traced process. A process being traced behaves normally until it encounters some signal whether internally generated like "illegal instruction" or externally generated like "interrupt". See sigvec(2) for the list.

Upon encountering a signal the traced process enters a stopped state and its tracing process is notified via wait(2). If the traced process stops with a SIGTRAP the process may have been stopped for a number of reasons. Two status words addressable as registers in the traced process's uarea qualify SIGTRAPs: TRAPCAUSE, which contains the cause of the trap, and TRAPINFO, which contains extra information concerning the trap.

When the traced process is in the stopped state, its core image can be examined and modified using ptrace. If desired, another ptrace request can then cause the traced process either to terminate or to continue, possibly ignoring the signal.

The value of the request argument determines the precise action of the call:

0 This request is the only one that may be used by a child process; it may declare that it is to be traced by its parent. All other arguments are ignored. Peculiar results will ensue if the parent does not expect to trace the child.

1,2 The word in the traced process's address space at addr is returned. If I and D space are separated (e.g. historically on a pdp-11), request 1 indicates I space, 2 D space. addr must be 4-byte aligned. The traced process must be stopped. The input data is ignored.

3 The word of the system's per-process data area corresponding to addr is returned. addr is a constant defined in sys/ptrace.h. This space contains the registers and other information about the process; the constants correspond to fields in the user structure in the system.

4,5 The given data is written at the word in the process's address space corresponding to addr, which must be 4-byte aligned. The old value at the address is returned. If I and D space are separated, request 4 indicates I space, 5 D space. Attempts to write in pure procedure fail if another process is executing the same file.

6 The process's system data is written, as it is read with request 3. Only a few locations can be written in this way: the general registers, the floating point status and registers, and certain bits of the processor status word. The old value at the address is returned.

7 The data argument is taken as a signal number and the traced process's execution continues at location addr as if it had incurred that signal. Normally the signal number will be either 0 to indicate that the signal that caused the stop should be ignored, or that value fetched out of the process's image indicating which signal caused the stop. If addr is (int *)1 then execution continues from where it stopped.

8 The traced process terminates.

9 Execution continues as in request 7; however, as soon as possible after execution of at
least one instruction, execution stops again. The signal number from the stop is SIGTRAP. TRAPCAUSE will contain CAUSESINGLE. This is part of the mechanism for implementing breakpoints.

As indicated, these calls (except for request 0 and 20) can be used only when the subject process has stopped. The wait call is used to determine when a process stops; in such a case the "termination" status returned by wait has the value 0177 to indicate stoppage rather than genuine termination. If multiple processes are being traced, wait can be called multiple times and will return the status for the next stopped or terminated child or traced process.

To forestall possible fraud, ptrace inhibits the set-user-id and set-group-id facilities on subsequent execve(2) calls. If a traced process calls execve, it will stop before executing the first instruction of the new image showing signal SIGTRAP. In this case TRAPCAUSE will contain CAUSEEXEC and TRAPINFO will not contain anything interesting. If a traced process execs again, the same thing will happen.

If a traced process forks, both parent and child will be traced. Breakpoints from the parent will not be copied into the child. At the time of the fork, the child will be stopped with a SIGTRAP. The tracing process may then terminate the trace if desired. TRAPCAUSE will contain CAUSEFORK and TRAPINFO will contain the pid of its parent.

RETURN VALUE
A 0 value is returned if the call succeeds. If the call fails then a −1 is returned and the global variable errno is set to indicate the error.

ERRORS
[EINVAL] The request code is invalid.
[EINVAL] The specified process does not exist.
[EINVAL] The given signal number is invalid.
[EFAULT] The specified address is out of bounds.
[EPERM] The specified process cannot be traced.

SEE ALSO
wait(2), sigvec(2), adb(1)

BUGS
ptrace is unique and arcane; it should be replaced with a special file which can be opened and read and written. The control functions could then be implemented with ioctl(2) calls on this file. This would be simpler to understand and have much higher performance.

The request 0 call should be able to specify signals which are to be treated normally and not cause a stop. In this way, for example, programs with simulated floating point (which use "illegal instruction" signals at a very high rate) could be efficiently debugged.

The error indication, −1, is a legitimate function value; errno, see intro(2), can be used to disambiguate.

It should be possible to stop a process on occurrence of a system call; in this way a completely controlled environment could be provided.
NAME
  quota – manipulate disk quotas

SYNOPSIS
#include <sys/quota.h>
quota(cmd, uid, arg, addr)
int cmd, uid, arg;
caddr_t addr;

DESCRIPTION
N.B.: This call is not implemented in the current version of the system.

The quota call manipulates disk quotas for file systems which have had quotas enabled with setquota(2). The cmd parameter indicates a command to be applied to the user ID uid. arg is a command specific argument and addr is the address of an optional, command specific, data structure which is copied in or out of the system. The interpretation of arg and addr is given with each command below.

Q_SETDLIM  Set disc quota limits and current usage for the user with ID uid. arg is a major-minor device indicating a particular file system. addr is a pointer to a struct dqblk structure (defined in <sys/quota.h>). This call is restricted to the super-user.

Q_GETDLIM  Get disc quota limits and current usage for the user with ID uid. The remaining parameters are as for Q_SETDLIM.

Q_SETDUSE  Set disc usage limits for the user with ID uid. arg is a major-minor device indicating a particular file system. addr is a pointer to a struct dqusag structure (defined in <sys/quota.h>). This call is restricted to the super-user.

Q_SYNC     Update the on-disc copy of quota usages. The uid, arg, and addr parameters are ignored.

Q_SETUID   Change the calling process’s quota limits to those of the user with ID uid. The arg and addr parameters are ignored. This call is restricted to the super-user.

Q_SETWARN  Alter the disc usage warning limits for the user with ID uid. arg is a major-minor device indicating a particular file system. addr is a pointer to a struct dqwarn structure (defined in <sys/quota.h>). This call is restricted to the super-user.

Q_DOWARN   Warn the user with user ID uid about excessive disc usage. This call causes the system to check its current disc usage information and print a message on the terminal of the caller for each file system on which the user is over quota. If the arg parameter is specified as NODEV, all file systems which have disc quotas will be checked. Otherwise, arg indicates a specific major-minor device to be checked. This call is restricted to the super-user.

RETURN VALUE
A successful call returns 0 and, possibly, more information specific to the cmd performed; when an error occurs, the value −1 is returned and errno is set to indicate the reason.

ERRORS
A quota call will fail when one of the following occurs:

[EINVAL]  Cmd is invalid.
[ESRCH]    No disc quota is found for the indicated user.
[EPERM] The call is privileged and the caller was not the super-user.
[EINVAL] The arg parameter is being interpreted as a major-minor device and it indicates an unmounted file system.
[EFAULT] An invalid addr is supplied; the associated structure could not be copied in or out of the kernel.
[EUSERS] The quota table is full.

SEE ALSO
setquota(2), quotaon(8), quotacheck(8)

BUGS
There should be someway to integrate this call with the resource limit interface provided by setrlimit(2) and getrlimit(2).

The Australian spelling of disk is used throughout the quota facilities in honor of the implementors.
NAME
quotactl – manipulate disk quotas

SYNOPSIS
#include <ufs/quota.h>
quotactl(cmd, special, uid, addr)
int cmd;
char *special;
int uid;
caddr_t addr;

DESCRIPTION
The quotactl call manipulates disk quotas. The cmd parameter indicates a command to be
applied to the user ID uid. special is a pointer to a null-terminated string containing the path
name of the block special device for the file system being manipulated. The block special
device must be mounted. addr is the address of an optional, command specific, data structure
which is copied in or out of the system. The interpretation of addr is given with each
command below.

Q_QUOTAON Turn on quotas for a file system. addr is a pointer to a null terminated
string containing the path name of file containing the quotas for the file
system. The quota file must exist; it is normally created with the quota-
check(8) program. This call is restricted to the super-user.

Q_QUOTAOFF Turn off quotas for a file system. This call is restricted to the super-
user.

Q_GETQUOTA Get disk quota limits and current usage for user uid. addr is a pointer
to a struct dqblk structure (defined in <ufs/quota.h>). Only the super-
user may get the quotas of a user other than himself.

Q_SETQUOTA Set disk quota limits and current usage for user uid. addr is a pointer to
a struct dqblk structure (defined in <ufs/quota.h>). This call is re-
stricted to the super-user.

Q_SETQLIM Set disk quota limits for user uid. addr is a pointer to a struct dqblk
structure (defined in <ufs/quota.h>). This call is restricted to the super-user.

Q_SYNC Update the on-disk copy of quota usages. This call is restricted to the
super-user.

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned
and errno is set to indicate the error.

ERRORS
A quotactl call will fail when one of the following occurs:

EINVAL cmd is invalid.

EPERM The call is privileged and the caller was not the super-user.

EINVAL The special parameter is not a mounted file system or is a mounted file
system without quotas enabled.

ENOTBLK The special parameter is not a block device.

EFAULT An invalid addr is supplied; the associated structure could not be copied
in or out of the kernel.

EINVAL The addr parameter is being interpreted as the path of a quota file
which exists but is either not a regular file or is not on the file system pointed to by the special parameter.

E_USERS

The quota table is full.

SEE ALSO

quotaon(8), quotacheck(8)

BUGS

There should be some way to integrate this call with the resource limit interface provided by setrlimit(2) and getrlimit(2). Incompatible with Melbourne quotas.
NAME

read, readv -- read input

SYNOPSIS

cc = read(d, buf, nbytes)
int cc, d;
char *buf;
int nbytes;

#include <sys/types.h>
#include <sys/uio.h>

cc = readv(d, iov, iovcnt)
int cc, d;
struct iovec *iov;
int iovcnt;

DESCRIPTION

read attempts to read nbytes of data from the object referenced by the descriptor d into the buffer pointed to by buf. readv performs the same action, but scatters the input data into the iovecnt buffers specified by the members of the iov array: iov[0], iov[1], ..., iov[iovcnt - 1].

For readv, the iovec structure is defined as

struct iovec {
    caddr_t iov_base;
    int iov_len;
};

Each iovec entry specifies the base address and length of an area in memory where data should be placed. readv will always fill an area completely before proceeding to the next.

On objects capable of seeking, the read starts at a position given by the pointer associated with d (see lseek(2)). Upon return from read, the pointer is incremented by the number of bytes actually read.

Objects that are not capable of seeking always read from the current position. The value of the pointer associated with such an object is undefined.

Upon successful completion, read and readv return the number of bytes actually read and placed in the buffer. The system guarantees to read the number of bytes requested if the descriptor references a normal file that has that many bytes left before the end-of-file, but in no other case.

If the returned value is 0, then end-of-file has been reached.

RETURN VALUE

If successful, the number of bytes actually read is returned. Otherwise, a -1 is returned and the global variable errno is set to indicate the error.

ERRORS

read and readv will fail if one or more of the following are true:

EBADF
    D is not a valid file or socket descriptor open for reading.
EFAULT
    buf points outside the allocated address space.
EIO
    An I/O error occurred while reading from the file system.
EINVAL
    A read from a slow device was interrupted before any data arrived by the delivery of a signal.
EINVAL
    The pointer associated with d was negative.
[EWOULDBLOCK] The file was marked for non-blocking I/O, and no data were ready to be read.

In addition, `readv` may return one of the following errors:

[EINVAL] `iovcnt` was less than or equal to 0, or greater than 16.

[EINVAL] One of the `iov_len` values in the `iov` array was negative.

[EINVAL] The sum of the `iov_len` values in the `iov` array overflowed a 32-bit integer.

[EFAULT] Part of the `iov` points outside the process’s allocated address space.

SEE ALSO
dup(2), fcntl(2), open(2), pipe(2), select(2), socket(2), socketpair(2)
NAME
readlink – read value of a symbolic link

SYNOPSIS
cc = readlink(path, buf, bufsiz)
int cc;
char *path, *buf;
int bufsiz;

DESCRIPTION
readlink places the contents of the symbolic link name in the buffer buf, which has size bufsiz. The contents of the link are not null terminated when returned.

RETURN VALUE
The call returns the count of characters placed in the buffer if it succeeds, or a –1 if an error occurs, placing the error code in the global variable errno.

ERRORS
readlink will fail and the file mode will be unchanged if:
[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 255 characters, or an entire path name exceeded 1023 characters.
[ENOENT] The named file 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.
[EINVAL] The named file is not a symbolic link.
[EIO] An I/O error occurred while reading from the file system.
[EFAULT] buf extends outside the process’s allocated address space.

SEE ALSO
stat(2), lstat(2), symlink(2)
NAME
reboot – reboot system or halt processor

SYNOPSIS
#include <sys/reboot.h>
reboot(howto)
int howto;

DESCRIPTION
reboot reboots the system, and is invoked automatically in the event of unrecoverable system
failures. howto is a mask of options passed to the bootstrap program. The system call inter-
face permits only RB_HAL T or RB_AUTobo oT to be passed to the reboot program; the other
flags are used in scripts stored on the console storage media, or used in manual bootstrap pro-
cedures. When none of these options (e.g. RB_AUTobo oT ) is given, the system is rebooted
from file “vmunix” in the root file system of unit 0 of a disk chosen in a processor specific
way. An automatic consistency check of the disks is then normally performed.

The bits of howto are:

RB_HAL T the processor is simply halted; no reboot takes place. RB_HAL T should
be used with caution.

RB_ASKNAME Interpreted by the bootstrap program itself, causing it to inquire as to
what file should be booted. Normally, the system is booted from the
file “xx(0,0)vmunix” without asking.

RB_SINGLE Normally, the reboot procedure involves an automatic disk consistency
check and then multi-user operations. RB_SINGLE prevents the con-
sistency check, rather simply booting the system with a single-user shell
on the console. RB_SINGLE is interpreted by the init(8) program in the
newly booted system. This switch is not available from the system call
interface.

Only the super-user may reboot a machine.

RETURN VALUES
If successful, this call never returns. Otherwise, a −1 is returned and an error is returned in
the global variable errno.

ERRORS
EPERM The caller is not the super-user.

SEE ALSO
crash(8), halt(8), init(8), reboot(8)

BUGS
The notion of “console medium”, among other things, is specific to the VAX.
NAME
recv, recvfrom, recvmsg — receive a message from a socket

SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>

int cc = recv(s, buf, len, flags);

int cc = recvfrom(s, buf, len, flags, from, fromlen);

int cc = recvmsg(s, msg, flags);

DESCRIPTION
recv, recvfrom, and recvmsg are used to receive messages from a socket.
The recv call is normally used only on a connected socket (see connect(2)), while recvfrom andecvmsg may be used to receive data on a socket whether it is in a connected state or not.
If from is non-zero, the source address of the message is filled in. Fromlen is a value-result
parameter, initialized to the size of the buffer associated with from, and modified on return to
indicate the actual size of the address stored there. The length of the message is returned in
cc. If a message is too long to fit in the supplied buffer, excess bytes may be discarded
depending on the type of socket the message is received from (see socket(2)).
If no messages are available at the socket, the receive call waits for a message to arrive, unless
the socket is nonblocking (see ioctl(2)) in which case a cc of -1 is returned with the external
variable errno set to EWOULD BLOCK.
The select(2) call may be used to determine when more data arrives.
The flags argument to a recv call is formed by or’ing one or more of the values,

#define MSG_OOB 0x1 /* process out-of-band data */
#define MSG_PEEK 0x2 /* peek at incoming message */

The recvmsg call uses a msghdr structure to minimize the number of directly supplied parameters.
This structure has the following form, as defined in <sys/socket.h>:

struct msghdr {
    caddr_t msg_name;          /* optional address */
    int msg_name_len;          /* size of address */
    struct iovec *msg_iov;     /* scatter/gather array */
    int msg_iovlen;            /* # elements in msg_iov */
    caddr_t msg_accrights;     /* access rights sent/received */
    int msg_accrightslen;
};
Here msg_name and msg_namelen specify the destination address if the socket is unconnected; msg_name may be given as a null pointer if no names are desired or required. The msg_iov and msg_iovlen describe the scatter gather locations, as described in read(2). A buffer to receive any access rights sent along with the message is specified in msg_accrights, which has length msg_accrightslen. Access rights are currently limited to file descriptors, which each occupy the size of an int.

RETURN VALUE
These calls return the number of bytes received, or -1 if an error occurred.

ERRORS
The calls fail if:

[EBADF] The argument s is an invalid descriptor.

[ENOTSOCK] The argument s is not a socket.

[EWOULDBLOCK] The socket is marked non-blocking and the receive operation would block.

[EINTR] The receive was interrupted by delivery of a signal before any data was available for the receive.

[EFAULT] The data was specified to be received into a non-existent or protected part of the process address space.

SEE ALSO
fcntl(2), read(2), send(2), select(2), getsockopt(2), socket(2)
NAME
rename – change the name of a file

SYNOPSIS
rename(from, to)
char *from, *to;

DESCRIPTION
rename causes the link named from to be renamed as to. If to exists, then it is first removed. Both from and to must be of the same type (that is, both directories or both non-directories), and must reside on the same file system.
rename guarantees that an instance of to will always exist, even if the system should crash in the middle of the operation.
If the final component of from is a symbolic link, the symbolic link is renamed, not the file or directory to which it points.

CAVEAT
The system can deadlock if a loop in the file system graph is present. This loop takes the form of an entry in directory “a”, say “a/foo”, being a hard link to directory “b”, and an entry in directory “b”, say “b/bar”, being a hard link to directory “a”. When such a loop exists and two separate processes attempt to perform “rename a/foo b/bar” and “rename b/bar a/foo”, respectively, the system may deadlock attempting to lock both directories for modification. Hard links to directories should be replaced by symbolic links by the system administrator.

RETURN VALUE
A 0 value is returned if the operation succeeds, otherwise rename returns −1 and the global variable errno indicates the reason for the failure.

ERRORS
rename will fail and neither of the argument files will be affected if any of the following are true:

[EINVAL] Either pathname contains a character with the high-order bit set.
[ENAMETOOLONG] A component of either pathname exceeded 255 characters, or the entire length of either path name exceeded 1023 characters.
[ENOENT] A component of the from path does not exist, or a path prefix of to does not exist.
[EACCES] A component of either path prefix denies search permission.
[EAGAIN] The requested link requires writing in a directory with a mode that denies write permission.
[EPERM] The directory containing from is marked sticky, and neither the containing directory nor from are owned by the effective user ID.
[EPERM] The to file exists, the directory containing to is marked sticky, and neither the containing directory nor to are owned by the effective user ID.
[ELOOP] Too many symbolic links were encountered in translating either pathname.
[ENOTDIR] A component of either path prefix is not a directory.
[ENOTDIR] from is a directory, but to is not a directory.
[EISDIR] to is a directory, but from is not a directory.
[EXDEV] The link named by to and the file named by from are on different logical
devices (file systems). Note that this error code will not be returned if the implementation permits cross-device links.

[ENOSPC] The directory in which the entry for the new name 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 new name is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.

[EIO] An I/O error occurred while making or updating a directory entry.

[EROFS] The requested link requires writing in a directory on a read-only file system.

[EFAULT] path points outside the process's allocated address space.

[EINVAL] from is a parent directory of to, or an attempt is made to rename "." or "..",

[ENOTEMPTY] to is a directory and is not empty.

SEE ALSO open(2)
NAME
rmkdir – remove a directory file

SYNOPSIS
rmkdir(path)
char *path;

DESCRIPTION
rmkdir removes a directory file whose name is given by path. The directory must not have any entries other than "." and "..".

RETURN VALUE
A 0 is returned if the remove succeeds; otherwise a -1 is returned and an error code is stored in the global location errno.

ERRORS
The named file is removed unless one or more of the following are true:
[ENOTDIR] 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 255 characters, or an entire path name exceeded 1023 characters.
[ENOENT] The named directory does not exist.
[EINVAL] Too many symbolic links were encountered in translating the pathname.
[ENOTEMPTY] The named directory contains files other than "." and ".." in it.
[EACCES] Search permission is denied for a component of the path prefix.
[EACCES] Write permission is denied on the directory containing the path prefix.
[PERM] The directory containing the directory to be removed is marked sticky, and neither the containing directory nor the directory to be removed are owned by the effective user ID.
[EBUSY] The directory to be removed is the mount point for a mounted file system.
[EIO] An I/O error occurred while deleting the directory entry or deallocating the inode.
[EROFS] The directory entry to be removed resides on a read-only file system.
[EFAULT] path points outside the process's allocated address space.

SEE ALSO
mkdir(2), unlink(2)
NAME
    select – synchronous I/O multiplexing

SYNOPSIS
    #include <sys/types.h>
    #include <sys/time.h>

    nfoundsel ect(nfds, readfds, writefds, exceptfds, timeout)
    int nfounds, nfds;
    fd_set *readfds, *writefds, *exceptfds;
    struct timeval *timeout;
    FD_SET(fd, &fdset)
    FD_CLR(fd, &fdset)
    FD_ISSET(fd, &fdset)
    FD_ZERO(&fdset)
    int fd;
    fd_set fdset;

DESCRIPTION
    select examines the I/O descriptor sets whose addresses are passed in readfds, writefds, and exceptfds to see if some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. The first nfds descriptors are checked in each set; i.e. the descriptors from 0 through nfds-1 in the descriptor sets are examined. On return, select replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. The total number of ready descriptors in all the sets is returned in nfounds.

    The descriptor sets are stored as bit fields in arrays of integers. The following macros are provided for manipulating such descriptor sets: "FD_ZERO(&fdset)" initialises a descriptor set fdset to the null set. FD_SET(fd, &fdset) includes a particular descriptor fd in fdset. FD_CLR(fd, &fdset) removes fd from fdset. FD_ISSET(fd, &fdset) is nonzero if fd is a member of fdset, zero otherwise. The behavior of these macros is undefined if a descriptor value is less than zero or greater than or equal to FD_SETSIZE, which is normally at least equal to the maximum number of descriptors supported by the system.

    If timeout is a non-zero pointer, it specifies a maximum interval to wait for the selection to complete. If timeout is a zero pointer, the select blocks indefinitely. To affect a poll, the timeout argument should be non-zero, pointing to a zero-valued timeval structure.

    Any of readfds, writefds, and exceptfds may be given as zero pointers if no descriptors are of interest.

RETURN VALUE
    select returns the number of ready descriptors that are contained in the descriptor sets, or -1 if an error occurred. If the time limit expires then select returns 0. If select returns with an error, including one due to an interrupted call, the descriptor sets will be unmodified.

ERRORS
    An error return from select indicates:

    [EBADF] One of the descriptor sets specified an invalid descriptor.
    [EINVAL] A signal was delivered before the time limit expired and before any of the selected events occurred.
    [EINVAL] The specified time limit is invalid. One of its components is negative or too large.
SEE ALSO
accept(2), connect(2), read(2), write(2), recv(2), send(2), getdtablesize(2)

BUGS
Although the provision of getdtablesize(2) was intended to allow user programs to be written independent of the kernel limit on the number of open files, the dimension of a sufficiently large bit field for select remains a problem. The default size FD_SETSIZE (currently 256) is somewhat larger than the current kernel limit to the number of open files. However, in order to accommodate programs which might potentially use a larger number of open files with select, it is possible to increase this size within a program by providing a larger definition of FD_SETSIZE before the inclusion of <sys/types.h>.

select should probably return the time remaining from the original timeout, if any, by modifying the time value in place. This may be implemented in future versions of the system. Thus, it is unwise to assume that the timeout value will be unmodified by the select call.
NAME
send, sendto, sendmsg - send a message from a socket

SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>

cc = send(s, msg, len, flags)
int cc, s;
char *msg;
int len, flags;

cc = sendto(s, msg, len, flags, to, tolen)
int cc, s;
char *msg;
int len, flags;
struct sockaddr *to;
int tolen;

cc = sendmsg(s, msg, flags)
int cc, s;
struct msghdr msg[];
int flags;

DESCRIPTION
send, sendto, and sendmsg are used to transmit a message to another socket. send may be
used only when the socket is in a connected state, while sendto and sendmsg may be used at
any time.

The address of the target is given by to with tolen specifying its size. The length of the mes-
gage is given by len. If the message is too long to pass atomically through the underlying pro-
ocol, then the error EMSGSIZE is returned, and the message is not transmitted.

No indication of failure to deliver is implicit in a send. Return values of -1 indicate some
locally detected errors.

If no messages space is available at the socket to hold the message to be transmitted, then
send normally blocks, unless the socket has been placed in non-blocking I/O mode. The
select(2) call may be used to determine when it is possible to send more data.

The flags parameter may include one or more of the following:
#define MSG_OOB 0x1 /* process out-of-band data */
#define MSG_DONTROUTE 0x4 /* bypass routing, use direct interface */

The flag MSG_OOB is used to send "out-of-band" data on sockets that support this notion
(e.g. SOCK_STREAM); the underlying protocol must also support "out-of-band" data.
MSG_DONTROUTE is usually used only by diagnostic or routing programs.

See recv(2) for a description of the msghdr structure.

RETURN VALUE
The call returns the number of characters sent, or -1 if an error occurred.

ERRORS
[EBADF] An invalid descriptor was specified.
[ENOTSOCK] The argument s is not a socket.
[EFAULT] An invalid user space address was specified for a parameter.
[EMSGSIZE] The socket requires that message be sent atomically, and the size of the
message to be sent made this impossible.
[EWOULDBLOCK] The socket is marked non-blocking and the requested operation would block.

[ENOBUFS] The system was unable to allocate an internal buffer. The operation may succeed when buffers become available.

[ENOBUFS] The output queue for a network interface was full. This generally indicates that the interface has stopped sending, but may be caused by transient congestion.

SEE ALSO
fcntl(2), recv(2), select(2), getsockopt(2), socket(2), write(2)
NAME
setgroups – set group access list

SYNOPSIS
#include <sys/param.h>
setgroups(ngroups, gidset)
int ngroups, *gidset;

DESCRIPTION
setgroups sets the group access list of the current user process according to the array gidset.
The parameter ngroups indicates the number of entries in the array and must be no more than
NGROUPS, as defined in <sys/param.h>.
Only the super-user may set new groups.

RETURN VALUE
A 0 value is returned on success, −1 on error, with a error code stored in errno.

ERRORS
The setgroups call will fail if:
[EPERM] The caller is not the super-user.
[EFAULT] The address specified for gidset is outside the process address space.

SEE ALSO
setgroups(2), initgroups(3X)

BUGS
The gidset array should be of type gid_t, but remains integer for compatibility with earlier sys-
tems.
NAME
   setpgrp – set process group

SYNOPSIS
   setpgrp(pid, pgrp)
   int pid, pgrp;

DESCRIPTION
   setpgrp sets the process group of the specified process pid to the specified pgrp. If pid is zero,
   then the call applies to the current process.

   If the invoker is not the super-user, then the affected process must have the same effective
   user-id as the invoker or be a descendant of the invoking process.

RETURN VALUE
   setpgrp returns when the operation was successful. If the request failed, −1 is returned and the
   global variable errno indicates the reason.

ERRORS
   setpgrp will fail and the process group will not be altered if one of the following occur:
   [ESRCH] The requested process does not exist.
   [EPERM] The effective user ID of the requested process is different from that of the
   caller and the process is not a descendant of the calling process.

SEE ALSO
   getpgrp(2)
NAME
   setquota – enable/disable quotas on a file system

SYNOPSIS
   setquota(special, file)
   char *special, *file;

DESCRIPTION
   Disc quotas are enabled or disabled with the setquota call. special indicates a block special
   device on which a mounted file system exists. If file is nonzero, it specifies a file in that file
   system from which to take the quotas. If file is 0, then quotas are disabled on the file system.
   The quota file must exist; it is normally created with the quotacheck(8) program.
   Only the super-user may turn quotas on or off.

SEE ALSO
   quota(2), quotacheck(8), quotaon(8)

RETURN VALUE
   A 0 return value indicates a successful call. A value of −1 is returned when an error occurs
   and errno is set to indicate the reason for failure.

ERRORS
   setquota will fail when one of the following occurs:
   [EPERM]    The caller is not the super-user.
   [ENOENT]   special does not exist.
   [ENOTBLK]  special is not a block device.
   [ENXIO]    The major device number of special is out of range (this indicates no
device driver exists for the associated hardware).
   [EPERM]    The pathname contains a character with the high-order bit set.
   [ENOTDIR]  A component of the path prefix in file is not a directory.
   [EACCES]   file resides on a file system different from special.
   [EACCES]   file is not a plain file.
   [ENAMETOOLONG] The pathname was too long.
   [EFAULT]  special or file points outside the process's allocated address space.
   [EIO]      An I/O error occurred while reading from or writing to the file system.

BUGS
   The error codes are in a state of disarray; too many errors appear to the caller as one value.
NAME
setregid – set real and effective group ID

SYNOPSIS
setregid(rgid, egid)
int rgid, egid;

DESCRIPTION
The real and effective group ID’s of the current process are set to the arguments. Unprivileged users may change the real group ID to the effective group ID and vice-versa; only the super-user 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 VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

ERRORS
[EPERM] The current process is not the super-user and a change other than changing the effective group-id to the real group-id was specified.

SEE ALSO
getgid(2), setreuid(2), setgid(3)
NAME
setreuid - set real and effective user ID's

SYNOPSIS
setreuid(ruid, euid)
int ruid, euid;

DESCRIPTION
The real and effective user ID's of the current process are set according to the arguments. If
ruid or euid is -1, the current uid is filled in by the system. Unprivileged users may change
the real user ID to the effective user ID and vice-versa; only the super-user may make other
changes.

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned
and errno is set to indicate the error.

ERRORS
[EPERM] The current process is not the super-user and a change other than
changing the effective user-id to the real user-id was specified.

SEE ALSO
getuid(2), setregid(2), setuid(3)
NAME
shutdown – shut down part of a full-duplex connection

SYNOPSIS
shutdown(s, how)
int s, how;

DESCRIPTION
The shutdown call causes all or part of a full-duplex connection on the socket associated with s to be shut down. If how is 0, then further receives will be disallowed. If how is 1, then further sends will be disallowed. If how is 2, then further sends and receives will be disallowed.

DIAGNOSTICS
A 0 is returned if the call succeeds, −1 if it fails.

ERRORS
The call succeeds unless:
[EBADF] s is not a valid descriptor.
[ENOTSOCK] s is a file, not a socket.
[ENOTCONN] The specified socket is not connected.

SEE ALSO
connect(2), socket(2)
NAME
   sigblock - block signals

SYNOPSIS
   #include <signal.h>

   sigblock(mask);
   int mask;

   mask = sigmask(signum)

DESCRIPTION
   sigblock causes the signals specified in mask to be added to the set of signals currently being
   blocked from delivery. Signals are blocked if the corresponding bit in mask is a 1; the macro
   sigmask is provided to construct the mask for a given signum.

   It is not possible to block SIGKILL, SIGSTOP, or SIGCONT; this restriction is silently imposed
   by the system.

RETURN VALUE
   The previous set of masked signals is returned.

SEE ALSO
   kill(2), sigvec(2), sigsetmask(2)
NAME
sigpause — atomically release blocked signals and wait for interrupt

SYNOPSIS
sigpause(sigmask)
int sigmask;

DESCRIPTION
sigpause assigns sigmask to the set of masked signals and then waits for a signal to arrive; on
return the set of masked signals is restored. sigmask is usually 0 to indicate that no signals are
now to be blocked. sigpause always terminates by being interrupted, returning -1 with errno
set to EINTR.

In normal usage, a signal is blocked using sigblock(2), to begin a critical section, variables
modified on the occurrence of the signal are examined to determine that there is no work to
be done, and the process pauses awaiting work by using sigpause with the mask returned by
sigblock.

SEE ALSO
sigblock(2), sigvec(2)
NAME
sigreturn – return from signal

SYNOPSIS
#include <signal.h>
sigreturn(sep);
struct sigcontext *sep;

DESCRIPTION
sigreturn allows users to atomically unmask, switch stacks, and return from a signal context. The processes signal mask and stack status are restored from the context. The system call does not return; the users registers are restored from the context. Execution resumes at the specified program counter (sc_pc) in the signal context structure. This system call is used by the trampoline code, and longjmp(3) when returning from a signal to the previously executing program.

NOTES
This system call is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

RETURN VALUE
If successful, the system call does not return. Otherwise, a value of -1 is returned and errno is set to indicate the error.

ERRORS
sigreturn will fail and the process context will remain unchanged if the following occurs.

[EFAULT] scp points to memory that is not a valid part of the process address space.

SEE ALSO
sigvec(2), setjmp(3)
NAME
 sigsetmask, sigmask – set current signal mask

SYNOPSIS
 #include <signal.h>

 sigsetmask(mask);
 int mask;

 mask = sigmask(signum)

DESCRIPTION
 sigsetmask sets the current signal mask (those signals that are blocked from delivery). Signals are blocked if the corresponding bit in mask is a 1; the macro sigmask is provided to construct the mask for a given signum.

 The system quietly disallows SIGKILL, SIGSTOP, or SIGCONT to be blocked.

RETURN VALUE
 The previous set of masked signals is returned.

SEE ALSO
 kill(2), sigvec(2), sigblock(2), sigpause(2)
NAME
    sigstack — set and/or get signal stack context

SYNOPSIS
    #include <signal.h>
    struct sigstack {
        caddr_t ss_sp;
        int ss_onstack;
    };
    sigstack(ss, oss);
    struct sigstack *ss, *oss;

DESCRIPTION
    sigstack allows users to define an alternate stack on which signals are to be processed. If ss is
    non-zero, it specifies a signal stack on which to deliver signals and tells the system if the pro-
    cess is currently executing on that stack. When a signal's action indicates its handler should
    execute on the signal stack (specified with a sigvec(2) call), the system checks to see if the pro-
    cess is currently executing on that stack. If the process is not currently executing on the signal
    stack, the system arranges a switch to the signal stack for the duration of the signal handler's
    execution. If oss is non-zero, the current signal stack state is returned.

NOTES
    Signal stacks are not “grown” automatically, as is done for the normal stack. If the stack
    overflows unpredictable results may occur.

RETURN VALUE
    Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned
    and errno is set to indicate the error.

ERRORS
    sigstack will fail and the signal stack context will remain unchanged if one of the following
    occurs.
    [EFAULT] Either ss or oss points to memory that is not a valid part of the process
    address space.

SEE ALSO
    sigvec(2), setjmp(3)
NAME
sigvec – software signal facilities

SYNOPSIS
#include <signal.h>

struct sigvec {
    int (*sv_handler)();
    int sv_mask;
    int sv_flags;
};
sigvec(sig, vec, ovec)
int sig;
struct sigvec *vec, *ovec;

DESCRIPTION
The system defines a set of signals that may be delivered to a process. Signal delivery resembles the occurrence of a hardware interrupt: the signal is blocked from further occurrence, the current process context is saved, and a new one is built. A process may specify a handler to which a signal is delivered, or specify that a signal is to be blocked or ignored. A process may also specify that a default action is to be taken by the system when a signal occurs. Normally, signal handlers execute on the current stack of the process. This may be changed, on a per-handler basis, so that signals are taken on a special signal stack.

All signals have the same priority. Signal routines execute with the signal that caused their invocation blocked, but other signals may yet occur. A global signal mask defines the set of signals currently blocked from delivery to a process. The signal mask for a process is initialized from that of its parent (normally 0). It may be changed with a sigblock(2) or sigsetmask(2) call, or when a signal is delivered to the process.

When a signal condition arises for a process, the signal is added to a set of signals pending for the process. If the signal is not currently blocked by the process then it is delivered to the process. When a signal is delivered, the current state of the process is saved, a new signal mask is calculated (as described below), and the signal handler is invoked. The call to the handler is arranged so that if the signal handling routine returns normally the process will resume execution in the context from before the signal's delivery. If the process wishes to resume in a different context, then it must arrange to restore the previous context itself.

When a signal is delivered to a process a new signal mask is installed for the duration of the process' signal handler (or until a sigblock or sigsetmask call is made). This mask is formed by taking the current signal mask, adding the signal to be delivered, and or'ing in the signal mask associated with the handler to be invoked.

sigvec assigns a handler for a specific signal. If vec is non-zero, it specifies a handler routine and mask to be used when delivering the specified signal. Further, if the SV_ONSTACK bit is set in sv_flags, the system will deliver the signal to the process on a signal stack, specified with sigstack(2). If ovec is non-zero, the previous handling information for the signal is returned to the user.

The following is a list of all signals with names as in the include file <signal.h>:

SIGHUP     1     hangup
SIGINT     2     interrupt
SIGQUIT    3*    quit
SIGILL     4+    illegal instruction
SIGTRAP    5+    trace trap
SIGIOT     6+    IOT instruction
SIGEMT     7+    EMT instruction
SIGFPE  8*  floating point exception
SIGKILL  9  kill (cannot be caught, blocked, or
SIGBUS  10*  bus error
SIGSEGV  11*  segmentation violation
SIGSYS  12*  bad argument to system call
SIGPIPE  13  write on a pipe with no
SIGALRM  14  alarm clock
SIGTERM  15  software termination signal
SIGURG  16*  urgent condition present on socket
SIGSTOP  17†  stop (cannot be caught, blocked, or
SIGTSTP  18†  stop signal generated from keyboard
SIGCONT  19*  continue after stop (cannot be blocked)
SIGHLD  20*  child status has changed
SIGTTIN  21†  background read attempted from control terminal
SIGTTOU  22†  background write attempted to control terminal
SIGIO  23*  i/o is possible on a descriptor
SIGXCPU  24  cpu time limit exceeded (see setrlimit(2))
SIGXFSZ  25  file size limit exceeded (see setrlimit(2))
SIGVTALRM  26  virtual time alarm (see setitimer(2))
SIGPROF  27  profiling timer alarm (see setitimer(2))
SIGWINCH  28*  window size change
SIGUSR1  30  user defined signal 1
SIGUSR2  31  user defined signal 2

The starred signals in the list above cause a core image if not caught or ignored.

Once a signal handler is installed, it remains installed until another sigvec call is made, or an
execve(2) is performed. The default action for a signal may be reinstated by setting sv_handler
to SIG_DFL; this default is termination (with a core image for starred signals) except for sig-
sals marked with * or †. Signals marked with * are discarded if the action is SIG_DFL; sig-
sals marked with † cause the process to stop. If sv_handler is SIG_IGN the signal is sub-
sequently ignored, and pending instances of the signal are discarded.

If a caught signal occurs during certain system calls, the call is normally restarted. The call
can be forced to terminate prematurely with an EINTR error return by setting the
SV_INTERRUPT bit in sv_flags. The affected system calls are read(2) or write(2) on a slow de-
vice (such as a terminal; but not a file) and during a wait(2).

After a fork(2) or vfork(2) the child inherits all signals, the signal mask, the signal stack, and
the restart/interrupt flags.

dexecve(2) resets all caught signals to default action and resets all signals to be caught on the
user stack. Ignored signals remain ignored; the signal mask remains the same; signals that
interrupt system calls continue to do so.

NOTES
The mask specified in vec is not allowed to block SIGHUP, SIGSTOP, or SIGCONT. This is
done silently by the system.

The SV_INTERRUPT flag is not available in 4.2BSD, hence it should not be used if backward
compatibility is needed.

RETURN VALUE
A 0 value indicated that the call succeeded. A -1 return value indicates an error occurred
and errno set to indicated the reason.
ERRORS

sigvec will fail and no new signal handler will be installed if one of the following occurs:

[EFAULT] Either vec or ovec points to memory that is not a valid part of the process address space.

[EINVAL] sig is not a valid signal number.

[EINVAL] An attempt is made to ignore or supply a handler for SIGKILL or SIGSTOP.

[EINVAL] An attempt is made to ignore SIGCONT (by default SIGCONT is ignored).

SEE ALSO

kill(1), ptrace(2), kill(2), sigblock(2), sigsetmask(2), sigpause(2), sigstack(2), sigvec(2),
setjmp(3), siginterrupt(3), tty(4), sigterm(2), emulate_branch(3), fpio(3), cache_flush(2)

R2010 Floating Point Coprocessor Architecture Engineering Description

R2360 Floating Point Board Product Description

NOTES (MIPS)

The handler routine can be declared:

handler(sig, code, scp)
int sig, code;
struct sigcontext *scp;

Here sig is the signal number. MIPS hardware exceptions are mapped to specific signals as defined by the table below. code is a parameter that is either a constant as given below or zero. scp is a pointer to the sigcontext structure (defined in <signal.h>), that is the context at the time of the signal and is used to restore the context if the signal handler returns.

The following defines the mapping of MIPS hardware exceptions to signals and codes. All of these symbols are defined in either <signal.h> or <mips/cpu.h>:

<table>
<thead>
<tr>
<th>Hardware exception</th>
<th>Signal</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer overflow</td>
<td>SIGFPE</td>
<td>EXC_OV</td>
</tr>
<tr>
<td>Segmentation violation</td>
<td>SIGSEGV</td>
<td>SEXC_SEGV</td>
</tr>
<tr>
<td>Illegal Instruction</td>
<td>SIGILL</td>
<td>EXC_Il</td>
</tr>
<tr>
<td>Coprocessor Unusable</td>
<td>SIGILL</td>
<td>SEXC_CPU</td>
</tr>
<tr>
<td>Data Bus Error</td>
<td>SIGBUS</td>
<td>EXC_DBE</td>
</tr>
<tr>
<td>Instruction Bus Error</td>
<td>SIGBUS</td>
<td>EXC_BIE</td>
</tr>
<tr>
<td>Read Address Error</td>
<td>SIGBUS</td>
<td>EXC_RADE</td>
</tr>
<tr>
<td>Write Address Error</td>
<td>SIGBUS</td>
<td>EXC_WADE</td>
</tr>
<tr>
<td>User Breakpoint (used by debuggers)</td>
<td>SIGTRAP</td>
<td>BRK_USERBP</td>
</tr>
<tr>
<td>Kernel Breakpoint (used by prom)</td>
<td>SIGTRAP</td>
<td>BRK_KERNELBP</td>
</tr>
<tr>
<td>Taken Branch Delay Emulation</td>
<td>SIGTRAP</td>
<td>BRK_BD_TAKEN</td>
</tr>
<tr>
<td>Not Taken Branch Delay Emulation</td>
<td>SIGTRAP</td>
<td>BRK_BD_NOTTAKE</td>
</tr>
<tr>
<td>User Single Step (used by debuggers)</td>
<td>SIGTRAP</td>
<td>BRK_SSTEPBP</td>
</tr>
<tr>
<td>Overflow Check</td>
<td>SIGTRAP</td>
<td>BRK_OVERFLOW</td>
</tr>
<tr>
<td>Divide by Zero Check</td>
<td>SIGTRAP</td>
<td>BRK_DIVZERO</td>
</tr>
<tr>
<td>Range Error Check</td>
<td>SIGTRAP</td>
<td>BRK_RANGE</td>
</tr>
</tbody>
</table>

When a signal handler is reached, the program counter in the signal context structure (sc_pc) points at the instruction that caused the exception as modified by the branch delay bit in the cause register. The cause register at the time of the exception is also saved in the sigcontext structure (sc_cause). If the instruction that caused the exception is at a valid user address it can be retrieved with the following code sequence:
if(scp->sc_cause & CAUSE BD){
    branch_instruction = *(unsigned long *)(scp->sc_pc);
    exception_instruction = *(unsigned long *)(scp->sc_pc + 4);
}
else
    exception_instruction = *(unsigned long *)(scp->sc_pc);

Where CAUSE BD is defined in <mips/cpu.h>.

The signal handler may fix the cause of the exception and re-execute the instruction, emulate the instruction and then step over it or perform some non-local goto such as a longjump() or an exit().

If corrective action is performed in the signal handler and the instruction that caused the exception would then execute without a further exception, the signal handler simply returns and re-executes the instruction (even when the branch delay bit is set).

If execution is to continue after stepping over the instruction that caused the exception the program counter must be advanced. If the branch delay bit is set the program counter is set to the target of the branch else it is incremented by 4. This can be done with the following code sequence:

if(scp->sc_cause & CAUSE BD)
    emulate_branch(scp, branch_instruction);
else
    scp->sc_pc += 4;

emulate_branch() modifies the program counter value in the sigcontext structure to the target of the branch instruction. See emulate_branch(3) for more details.

For SIGFPE’s generated by floating-point instructions (code == 0) the floating-point control and status register at the time of the exception is also saved in the sigcontext structure (sc_fpc_csr). This register has the information on which exceptions have occurred. When a signal handler is entered the register contains the value at the time of the exception but with the exceptions bits cleared. On a return from the signal handler the exception bits in the floating-point control and status register are also cleared so that another SIGFPE will not occur (all other bits are restored from sc_fpc_csr).

If the floating-point unit is a R2360 (a floating-point board) and a SIGFPE is generated by the floating-point unit (code == 0) and program counter does not point at the instruction that caused the exception. In this case the instruction that caused the exception is in the floating-point instruction exception register. The floating-point instruction exception register at the time of the exception is also saved in the sigcontext structure (sc_fpc_eir). In this case the instruction that caused the exception can be retrieved with the following code sequence:

union fpc_irr fpc_irr;

    fpc_irr.fu_word = get_fpc_irr();
    if(sig == SIGFPE && code == 0 &&
        fpc_irr.fu_struct.implementation == IMPLEMENTATION_R2360)
        exception_instruction = scp->sc_fpc_eir;

The union fpc_irr, and the constant IMPLEMENTATION_R2360 are defined in <mips/fpu.h>. For the description of the routine get_fpc_irr() see fpc(3). All other floating-point implementations are handled in the normal manner with the instruction that caused the exception at the program counter as modified by the branch delay bit.
For SIGSEGV and SIGBUS errors the faulting virtual address is saved in `sc_badvaddr` in the signal context structure.

The SIGTRAP's caused by `break` instructions noted in the above table and all other yet to be defined `break` instructions fill the `code` parameter with the first argument to the `break` instruction (bits 25-16 of the instruction).
NAME
socket – create an endpoint for communication

SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>

s = socket(domain, type, protocol)
int s, domain, type, protocol;

DESCRIPTION
socket creates an endpoint for communication and returns a descriptor.

The domain parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file <sys/socket.h>. The currently understood formats are

- PF_UNIX (UNIX internal protocols),
- PF_INET (ARPA Internet protocols),
- PF_NS (Xerox Network Systems protocols), and
- PF_IMPLINK (IMP “host at IMP” link layer).

The socket has the indicated type, which specifies the semantics of communication. Currently defined types are:

- SOCK_STREAM
- SOCK_DGRAM
- SOCK_RAW
- SOCK_SEQPACKET
- SOCK_RDM

A SOCK_STREAM type provides sequenced, reliable, two-way connection based byte streams. An out-of-band data transmission mechanism may be supported. A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). A SOCK_SEQPACKET socket may provide a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer may be required to read an entire packet with each read system call. This facility is protocol specific, and presently implemented only for PF_NS. SOCK_RAW sockets provide access to internal network protocols and interfaces. The types SOCK_RAW, which is available only to the super-user, and SOCK_RDM, which is planned, but not yet implemented, are not described here.

The protocol specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the “communication domain” in which communication is to take place; see protocols(3N).

Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a connect(2) call. Once connected, data may be transferred using read(2) and write(2) calls or some variant of the send(2) and recv(2) calls. When a session has been completed a close(2) may be performed. Out-of-band data may also be transmitted as described in send(2) and received as described in recv(2).
The communications protocols used to implement a SOCK_STREAM insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with -1 returns and with ETIMEDOUT as the specific code in the global variable errno. The protocols optionally keep sockets “warm” by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for a extended period (e.g. 5 minutes). A SIGPIPE signal is raised if a on a broken stream; this causes naive processes, which do not handle the signal, to exit.

SOCK_SEQPACKET sockets employ the same system calls as SOCK_STREAM sockets. The only difference is that read(2) calls will return only the amount of data requested, and any remaining in the arriving packet will be discarded.

SOCK_DGRAM and SOCK_RAW sockets allow sending of datagrams to correspondents named in send(2) calls. Datagrams are generally received with recvfrom(2), which returns the next datagram with its return address.

An fcntl(2) call can be used to specify a process group to receive a SIGURG signal when the out-of-band data arrives. It may also enable non-blocking I/O and asynchronous notification of I/O events via SIGIO.

The operation of sockets is controlled by socket level options. These options are defined in the file <sys/socket.h>. setsockopt(2) and getsockopt(2) are used to set and get options, respectively.

RETURN VALUE
A -1 is returned if an error occurs, otherwise the return value is a descriptor referencing the socket.

ERRORS
The socket call fails if:

[EPROTONOSUPPORT] The protocol type or the specified protocol is not supported within this domain.

[EMFILE] The per-process descriptor table is full.

[ENFILE] The system file table is full.

[EACCES] Permission to create a socket of the specified type and/or protocol is denied.

[ENOBUFS] Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed.

SEE ALSO
accept(2), bind(2), connect(2), getsockname(2), getsockopt(2), ioctl(2), listen(2), read(2), recv(2), select(2), send(2), shutdown(2), socketpair(2), write(2)

"An Introductory 4.3BSD Interprocess Communication Tutorial," (reprinted in UNIX Programmer’s Supplementary Documents Volume 1, PS1:7) "An Advanced 4.3BSD Interprocess Communication Tutorial." (reprinted in UNIX Programmer’s Supplementary Documents Volume 1, PS1:8)
NAME
socketpair – create a pair of connected sockets

SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>
socketpair(d, type, protocol, sv)
int d, type, protocol;
int sv[2];

DESCRIPTION
The socketpair call creates an unnamed pair of connected sockets in the specified domain d, of the specified type, and using the optionally specified protocol. The descriptors used in referencing the new sockets are returned in sv[0] and sv[1]. The two sockets are indistinguishable.

DIAGNOSTICS
A 0 is returned if the call succeeds, -1 if it fails.

ERRORS
The call succeeds unless:
[EMFILE] Too many descriptors are in use by this process.
[EAFNOSUPPORT] The specified address family is not supported on this machine.
[EPROTONOSUPPORT] The specified protocol is not supported on this machine.
[EOPNOTSUPP] The specified protocol does not support creation of socket pairs.
[EFAULT] The address sv does not specify a valid part of the process address space.

SEE ALSO
read(2), write(2), pipe(2)

BUGS
This call is currently implemented only for the UNIX domain.
NAME

stat, lstat, fstat — get file status

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

stat(path, buf)
char *path;
struct stat *buf;

lstat(path, buf)
char *path;
struct stat *buf;

fstat(fd, buf)
int fd;
struct stat *buf;
```

DESCRIPTION

stat obtains information about the file path. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be reachable.

lstat is like stat except in the case where the named file is a symbolic link, in which case lstat returns information about the link, while stat returns information about the file the link references.

fstat obtains the same information about an open file referenced by the argument descriptor, such as would be obtained by an open call.

buf is a pointer to a stat structure into which information is placed concerning the file. The contents of the structure pointed to by buf

```
struct stat {
    dev_t st_dev;    /* device inode resides on */
    ino_t st_ino;    /* this inode's number */
    u_short st_mode; /* protection */
    short st_nlink;  /* number or hard links to the file */
    short st_uid;    /* user-id of owner */
    short st_gid;    /* group-id of owner */
    dev_t st_rdev;   /* the device type, for inode that is device */
    off_t st_size;   /* total size of file */
    time_t st_atime; /* file last access time */
    int st_spare1;
    time_t st_mtime; /* file last modify time */
    int st_spare2;
    time_t st_ctime; /* file last status change time */
    int st_spare3;
    long st_blksize; /* optimal blocksize for file system i/o ops */
    long st_blocks;  /* actual number of blocks allocated */
    long st_spare4[2];
};
```

st_atime

Time when file data was last read or modified. Changed by the following system calls: mknod(2), utimes(2), read(2), and write(2). For reasons of efficiency, st_atime is not set when a directory is searched, although this would be more logical.
st_mtime  Time when data was last modified. It is not set by changes of owner, group, link count, or mode. Changed by the following system calls: mkod(2), utimes(2), write(2).

st_ctime  Time when file status was last changed. It is set both by writing and changing the i-node. Changed by the following system calls: chmod(2) chown(2), link(2), mkod(2), rename(2), unlink(2), utimes(2), write(2).

The status information word st_mode has bits:

```c
#define S_IFMT 0170000 /* type of file */
#define S_IFDIR 0040000 /* directory */
#define S_IFCHR 0020000 /* character special */
#define S_IFBLK 0060000 /* block special */
#define S_IFREG 0100000 /* regular */
#define S_IFLNK 0200000 /* symbolic link */
#define S_IFSOCK 0140000 /* socket */
#define S_ISUID 0004000 /* set user id on execution */
#define S_ISGID 0002000 /* set group id on execution */
#define S_ISVTX 0001000 /* save swapped text even after use */
#define S_IREAD 0000400 /* read permission, owner */
#define S_IWRITE 0000200 /* write permission, owner */
#define S_IEXEC 0000100 /* execute/search permission, owner */
```

The mode bits 0000070 and 0000007 encode group and others permissions (see chmod(2)).

RETURN VALUE
Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

ERRORS
stat and lstat will fail if one or more of the following are true:

- [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 255 characters, or an entire path name exceeded 1023 characters.
- [ENOENT] The named file 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.
- [EFAULT] buf or name points to an invalid address.
- [EIO] An I/O error occurred while reading from or writing to the file system.

fstat will fail if one or both of the following are true:

- [EBADF] fildes is not a valid open file descriptor.
- [EFAULT] buf points to an invalid address.
- [EIO] An I/O error occurred while reading from or writing to the file system.

CAVEAT
The fields in the stat structure currently marked st_spare1, st_spare2, and st_spare3 are present in preparation for inode time stamps expanding to 64 bits. This, however, can break certain programs that depend on the time stamps being contiguous (in calls to utimes(2)).
SEE ALSO
   chmod(2), chown(2), utimes(2)

BUGS
   Applying fstat to a socket (and thus to a pipe) returns a zero'd buffer, except for the blocksize field, and a unique device and inode number.
NAME

statfs – get file system statistics

SYNOPSIS

#include <sys/vfs.h>

statfs(  path,  buf)
char *path;
struct statfs *buf;

fstatfs(fd,  buf)
int fd;
struct statfs *buf;

DESCRIPTION

statfs returns information about a mounted file system. path is the path name of any file
within the mounted filesystem. buf is a pointer to a statfs structure defined as follows:

typedef struct {
    long   val[2];
} fsid_t;

struct statfs {
    long   f_type;    /* type of info, zero for now */
    long   f_bsize;  /* fundamental file system block size */
    long   f_blocks; /* total blocks in file system */
    long   f_bfree;  /* free blocks */
    long   f_bavail; /* free blocks available to non-superuser */
    long   f_files;  /* total file nodes in file system */
    long   f_ffree;  /* free file nodes in fs */
    fsid_t f_fsid;   /* file system id */
    long   f_spare[7]; /* spare for later */
};

Fields that are undefined for a particular file system are set to -1. fstatfs returns the same
information about an open file referenced by descriptor fd.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, -1 is returned and the global
variable errno is set to indicate the error.

ERRORS

statfs fails if one or more of the following are true:

ENOTDIR A component of the path prefix of path is not a directory.
EINVAL path contains a character with the high-order bit set.
ENOMEMETOLONG The length of a component of path exceeds 255 characters, or the length
of path exceeds 1023 characters.
ENOENT The file referred to by path does not exist.
EACCES Search permission is denied for a component of the path prefix of path.
ELOOP Too many symbolic links were encountered in translating path.
EFAULT buf or path points to an invalid address.
EIO An I/O error occurred while reading from or writing to the file system.
fstats fails if one or both of the following are true:

EBADF   \textit{fd} is not a valid open file descriptor.
EFAULT  \textit{buf} points to an invalid address.
EIO      An I/O error occurred while reading from or writing to the file system.
NAME
swapon – add a swap device for interleaved paging/swapping

SYNOPSIS
swapon(special)
char *special;

DESCRIPTION
swapon makes the block device special available to the system for allocation for paging and swapping. The names of potentially available devices are known to the system and defined at system configuration time. The size of the swap area on special is calculated at the time the device is first made available for swapping.

RETURN VALUE
If an error has occurred, a value of -1 is returned and errno is set to indicate the error.

ERRORS
swapon succeeds unless:

[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 255 characters, or an entire path name exceeded 1023 characters.
[ENOENT] The named 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 is not the super-user.
[ENOTBLK] special is not a block device.
[EBUSY] The device specified by special has already been made available for swapping.
[EINVAL] The device configured by special was not configured into the system as a swap device.
[ENXIO] The major device number of special 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] special points outside the process’s allocated address space.

SEE ALSO
swapon(8), config(8)

BUGS
There is no way to stop swapping on a disk so that the pack may be dismounted.
This call will be upgraded in future versions of the system.
NAME
   symlink – make symbolic link to a file

SYNOPSIS
   symlink(name1, name2)
   char *name1, *name2;

DESCRIPTION
   A symbolic link name2 is created to name1 (name2 is the name of the file created, name1 is
   the string used in creating the symbolic link). Either name may be an arbitrary path name; the
   files need not be on the same file system.

RETURN VALUE
   Upon successful completion, a zero value is returned. If an error occurs, the error code is
   stored in errno and a −1 value is returned.

ERRORS
   The symbolic link is made unless on or more of the following are true:
   [ENOTDIR]   A component of the name2 prefix is not a directory.
   [EINVAL]   Either name1 or name2 contains a character with the high-order bit set.
   [ENAMETOOLONG] A component of either pathname exceeded 255 characters, or the entire
                    length of either path name exceeded 1023 characters.
   [ENOENT]   The named file does not exist.
   [EACCES]   A component of the name2 path prefix denies search permission.
   [ELOOP]    Too many symbolic links were encountered in translating the pathname.
   [EEXIST]   name2 already exists.
   [EIO]      An I/O error occurred while making the directory entry for name2, or
               allocating the inode for name2, or writing out the link contents of
               name2.
   [EROFS]    The file name2 would reside on a read-only file system.
   [ENOSPC]   The directory in which the entry for the new symbolic link is being
               placed cannot be extended because there is no space left on the file sys-
               tem containing the directory.
   [ENOSPC]   The new symbolic link cannot be created because there is no
               space left on the file system that will contain the symbolic link.
   [ENOSPC]   There are no free inodes on the file system on which the symbolic link is
               being created.
   [EDQUOT]   The directory in which the entry for the new 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.
   [EDQUOT]   The new symbolic link cannot be created because the user’s quota of
               disk blocks on the file system that will contain the symbolic link has
               been exhausted.
   [EDQUOT]   The user’s quota of inodes on the file system on which the symbolic link
               is being created has been exhausted.
   [EIO]      An I/O error occurred while making the directory entry or allocating the
               inode.
   [EFAULT]  name1 or name2 points outside the process’s allocated address space.
SEE ALSO
link(2), ln(1), unlink(2)
NAME
  sync – update super-block

SYNOPSIS
  sync()

DESCRIPTION
  sync causes all information in core memory that should be on disk to be written out. This
  includes modified super blocks, modified i-nodes, and delayed block I/O.

  sync should be used by programs that examine a file system, for example fsck, df, etc. sync is
  mandatory before a boot.

SEE ALSO
  fsync(2), sync(8), update(8)

BUGS
  The writing, although scheduled, is not necessarily complete upon return from sync.
NAME
syscall – indirect system call

SYNOPSIS
#include <syscall.h>
system(number, arg, ...) (VAX-11)

DESCRIPTION
syscall performs the system call whose assembly language interface has the specified number, register arguments r0 and r1 and further arguments arg. Symbolic constants for system calls can be found in the header file <syscall.h>.
The r0 value of the system call is returned.

DIAGNOSTICS
When the C-bit is set, syscall returns −1 and sets the external variable errno (see intro(2)).

BUGS
There is no way to simulate system calls such as pipe(2), which return values in register r1.
NAME
truncate, ftruncate — truncate a file to a specified length

SYNOPSIS
truncate(path, length)
    char *path;
    off_t length;

    ftruncate(fd, length)
    int fd;
    off_t length;

DESCRIPTION
truncate causes the file named by path or referenced by fd to be truncated to at most length bytes in size. If the file previously was larger than this size, the extra data is lost. With ftruncate, the file must be open for writing.

RETURN VALUES
A value of 0 is returned if the call succeeds. If the call fails a -1 is returned, and the global variable errno specifies the error.

ERRORS
truncate succeeds unless: r.TP 20 [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 255 characters, or an entire pathname exceeded 1023 characters.
[ENOENT] The named file does not exist.
[EACCES] Search permission is denied for a component of the path prefix.
[EINVAL] The named file is not writable by the user.
[ELOOP] Too many symbolic links were encountered in translating the pathname.
[EISDIR] The named file is a directory.
[EROFS] The named file resides on a read-only file system.
[ETXTBSY] The file is a pure procedure (shared text) file that is being executed.
[EIO] An I/O error occurred updating the inode.
[EFAULT] Path points outside the process's allocated address space.

ftruncate succeeds unless:
[EBADF] The fd is not a valid descriptor.
[EINVAL] The fd references a socket, not a file.
[EINVAL] The fd is not open for writing.

SEE ALSO
open(2)

BUGS
These calls should be generalized to allow ranges of bytes in a file to be discarded.
NAME

umask – set file creation mode mask

SYNOPSIS

oumask = umask(numask)
int oumask, numask;

DESCRIPTION

umask sets the process's file mode creation mask to numask and returns the previous value of
the mask. The low-order 9 bits of numask are used whenever a file is created, clearing
the previous value of the mask.

The low-order 9 bits of numask are used whenever a file is created, clearing
the default access to his files.

The value is initially 022 (write access for owner only). The mask is inherited by child
processes.

RETURN VALUE

The previous value of the file mode mask is returned by the call.

SEE ALSO

chmod(2), mknod(2), open(2)
NAME
uname - get general system information

SYNOPSIS
#include <sys/utsname.h>

int uname(un)
struct utsname *un;

DESCRIPTION
uname stores information identifying the current operating system and machine into the structure pointed to by the argument.
The utsname structure is defined in the include file <sys/utsname.h>. It consists of 13 fields, 7 of which are defined and the rest of which are reserved for future use. The currently defined fields (with available values) are:
sysname The network identification name (same as the hostname).
nodename The network identification name (same as the hostname and the above sysname field).
release The operating system release name.
version The MIPS system version number.
machine The hardware type.
m_type (MIPS-specific) The MIPS hardware type.
base_rel (MIPS-specific) The base release for the system.
The valid values for these fields are defined in the utsname.h include file.

RETURN VALUE
If successful, uname will return a non-negative value; otherwise, it will return -1 and errno will indicate the error.

SEE ALSO
hwconf(2), gethostname(2).
NAME
unlink – remove directory entry

SYNOPSIS
unlink(path)
char *path;

DESCRIPTION
unlink removes the entry for the file path from its directory. If this entry was the last link to
the file, and no process has the file open, then all resources associated with the file are
reclaimed. If, however, the file was open in any process, the actual resource reclamation is
delayed until it is closed, even though the directory entry has disappeared.

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned
and errno is set to indicate the error.

ERRORS
The unlink succeeds unless:

[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 255 characters, or an entire path
name exceeded 1023 characters.
[ENOENT] The named file does not exist.
[EACCES] Search permission is denied for a component of the path prefix.
[EACCES] Write permission is denied on the directory containing the link to be
removed.
[ELOOP] Too many symbolic links were encountered in translating the pathname.
[EPERM] The named file is a directory and the effective user ID of the process is
not the super-user.
[EPERM] The directory containing the file is marked sticky, and neither the contain-
ing directory nor the file to be removed are owned by the effective
user ID.
[EBUSY] The entry to be unlinked is the mount point for a mounted file system.
[EIO] An I/O error occurred while deleting the directory entry or deallocating
the inode.
[EROFS] The named file resides on a read-only file system.
[EFAULT] path points outside the process’s allocated address space.

SEE ALSO
close(2), link(2), rmdir(2)
NAME
unmount – remove a file system

SYNOPSIS
unmount(name)
char *name;

DESCRIPTION
unmount announces to the system that the directory name is no longer to refer to the root of a mounted file system. The directory name reverts to its ordinary interpretation.

RETURN VALUE
unmount returns 0 if the action occurred; -1 if the directory is inaccessible or does not have a mounted file system, or if there are active files in the mounted file system.

ERRORS
unmount may fail with one of the following errors:
EPERM The caller is not the super-user.
ENOTDIR A component of the path prefix of name is not a directory.
EINVAL name is not the root of a mounted file system.
EBUSY A process is holding a reference to a file located on the file system.
EINVAL The path name contains a character with the high-order bit set.
ENAMETOOLONG The length of a component of the path name exceeds 255 characters, or the length of the entire path name exceeds 1023 characters.
ENOENT name does not exist.
EACCES Search permission is denied for a component of the path prefix.
EFAULT name points outside the process’s allocated address space.
ELOOP Too many symbolic links were encountered in translating the path name.
EIO An I/O error occurred while reading from or writing to the file system.

SEE ALSO
mount(2), mount(8), umount(8)

BUGS
The error codes are in a state of disarray; too many errors appear to the caller as one value.
NAME
utimes – set file times

SYNOPSIS
#include <sys/time.h>

utimes(file, tvp)
char *file;
struct timeval tvp[2];

DESCRIPTION
The utimes call uses the "accessed" and "updated" times in that order from the tvp vector to set the corresponding recorded times for file.

The caller must be the owner of the file or the super-user. The "inode-changed" time of the file is set to the current time.

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

ERRORS
utime will fail if one or more of the following are true:

[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 255 characters, or an entire path name exceeded 1023 characters.
[ENOENT] The named file does not exist.
[ELOOP] Too many symbolic links were encountered in translating the pathname.
[EPERM] The process is not super-user and not the owner of the file.
[EACCES] Search permission is denied for a component of the path prefix.
[EROFS] The file system containing the file is mounted read-only.
[EFAULT] file or tvp points outside the process’s allocated address space.
[EIO] An I/O error occurred while reading or writing the affected inode.

SEE ALSO
stat(2)
NAME

vfork - spawn new process in a virtual memory efficient way

SYNOPSIS

pid = vfork()
int pid;

DESCRIPTION

vfork can be used to create new processes without fully copying the address space of the old
process, which is horrendously inefficient in a paged environment. It is useful when the pur-
pose of fork(2) would have been to create a new system context for an execve. vfork differs
from fork in that the child borrows the parent's memory and thread of control until a call to
execve(2) or an exit (either by a call to exit(2) or abnormally.) The parent process is suspended
while the child is using its resources.

vfork returns 0 in the child's context and (later) the pid of the child in the parent's context.

vfork can normally be used just like fork. It does not work, however, to return while running
in the chlds context from the procedure that called vfork since the eventual return from vfork
would then return to a no longer existent stack frame. Be careful, also, to call _exit rather
than exit if you can't execve, since exit will flush and close standard I/O channels, and thereby
mess up the parent processes standard I/O data structures. (Even with fork it is wrong to call
exit since buffered data would then be flushed twice.)

SEE ALSO

fork(2), execve(2), sigvec(2), wait(2),

DIAGNOSTICS

Same as for fork.

BUGS

This system call will be eliminated when proper system sharing mechanisms are implemented.
Users should not depend on the memory sharing semantics of vfork as it will, in that case, be
made synonymous to fork.

To avoid a possible deadlock situation, processes that are children in the middle of a vfork
are never sent SIGTTOU or SIGTTIN signals; rather, output or iocits are allowed and input
attempts result in an end-of-file indication.
NAME
vhangup - virtually "hangup" the current control terminal

SYNOPSIS
vhangup()

DESCRIPTION
vhangup is used by the initialization process init(8) (among others) to arrange that users are
given "clean" terminals at login, by revoking access of the previous users' processes to the
terminal. To effect this, vhangup searches the system tables for references to the control ter-
|minal of the invoking process, revoking access permissions on each instance of the terminal
|that it finds. Further attempts to access the terminal by the affected processes will yield i/o
|errors (EBADF). Finally, a hangup signal (SIGHUP) is sent to the process group of the control
|terminal.

SEE ALSO
init (8)

BUGS
Access to the control terminal via /dev/tty is still possible.
This call should be replaced by an automatic mechanism that takes place on process exit.
NAME
wait, wait3 -- wait for process to terminate

SYNOPSIS

#include <sys/wait.h>

pid = wait(status)
int pid;
union wait *status;
pid = wait(0)
int pid;

#include <sys/time.h>
#include <sys/resource.h>

pid = wait3(status, options, rusage)
int pid;
union wait *status;
int options;
struct rusage *rusage;

pid = mips_wait3(status, options, rusage, rusage_size)
int pid;
union wait *status;
int options;
struct rusage *rusage;
int rusage_size;

DESCRIPTION
wait causes its caller to delay until a signal is received or one of its child processes terminates. If any child has died since the last wait, return is immediate, returning the process id and exit status of one of the terminated children. If there are no children, return is immediate with the value -1 returned.

On return from a successful wait call, status is nonzero, and the high byte of status contains the low byte of the argument to exit supplied by the child process; the low byte of status contains the termination status of the process. A more precise definition of the status word is given in <sys/wait.h>.

wait3 provides an alternate interface for programs that must not block when collecting the status of child processes. Mips_wait3 performs the same function as wait3 but takes a fourth argument which is the size of the rusage structure. This interface will be used in the future to return MIPS hardware specific resource use information as the rusage structure is extended.

The status parameter is defined as above. The options parameter is used to indicate the call should not block if there are no processes that wish to report status (WNOHANG), and/or that children of the current process that are stopped due to a SIGTTIN, SIGTTOU, SIGTSTP, or SIGSTOP signal should also have their status reported (WUNTRACED). If rusage is non-zero, a summary of the resources used by the terminated process and all its children is returned (this information is currently not available for stopped processes).

When the WNOHANG option is specified and no processes wish to report status, wait3 returns a pid of 0. The WNOHANG and WUNTRACED options may be combined by or'ing the two values.

NOTES
See sigvec(2) for a list of termination statuses (signals); 0 status indicates normal termination. A special status (0177) is returned for a stopped process that has not terminated and can be
restarted; see *ptrace*(2). If the 0200 bit of the termination status is set, a core image of the process was produced by the system.

If the parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

*wait* and *wait3* are automatically restarted when a process receives a signal while awaiting termination of a child process.

**RETURN VALUE**

If *wait* returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

*wait3* returns −1 if there are no children not previously waited for; 0 is returned if WNOHANG is specified and there are no stopped or exited children.

**ERRORS**

*wait* will fail and return immediately if one or more of the following are true:

- [ECHILD]: The calling process has no existing unwaited-for child processes.
- [EFAULT]: The *status* or *rusage* arguments point to an illegal address.

**SEE ALSO**

exit(2)
NAME
write, writev - write output

SYNOPSIS
cc = write(d, buf, nbytes)
int cc, d;
char *buf;
int nbytes;
#include <sys/types.h>
#include <sys/uio.h>
cc = writev(d, iov, iovcnt)
int cc, d;
struct iovec *iov;
int iovcnt;

DESCRIPTION
write attempts to write nbytes of data to the object referenced by the descriptor d from the
buffer pointed to by buf. writev performs the same action, but gathers the output data from the
iovcnt buffers specified by the members of the iov array: iov[0], iov[1], ..., iov[iovcnt - 1].

For writev, the iovec structure is defined as

struct iovec {
    caddr_t iov_base;
    int iov_len;
};

Each iovec entry specifies the base address and length of an area in memory from which data
should be written. writev will always write a complete area before proceeding to the next.

On objects capable of seeking, the write starts at a position given by the pointer associated
with d, see lseek(2). Upon return from write, the pointer is incremented by the number of
bytes actually written.

Objects that are not capable of seeking always write from the current position. The value of
the pointer associated with such an object is undefined.

If the real user is not the super-user, then write clears the set-user-id bit on a file. This
prevents penetration of system security by a user who "captures" a writable set-user-id file
owned by the super-user.

When using non-blocking I/O on objects such as sockets that are subject to flow control, write
and writev may write fewer bytes than requested; the return value must be noted, and the
remainder of the operation should be retried when possible.

RETURN VALUE
Upon successful completion the number of bytes actually written is returned. Otherwise a -1
is returned and the global variable errno is set to indicate the error.

ERRORS
write and writev will fail and the file pointer will remain unchanged if one or more of the fol-
lowing are true:

[EBADF]  D is not a valid descriptor open for writing.
[EPipe]  An attempt is made to write to a pipe that is not open for reading by
any process.
[EPipe]  An attempt is made to write to a socket of type SOCK_STREAM that is
        not connected to a peer socket.
[EBIG] An attempt was made to write a file that exceeds the process's file size limit or the maximum file size.

[EFAULT] Part of iov or data to be written to the file points outside the process's allocated address space.

[EINVAL] The pointer associated with d was negative.

[ENOSPC] There is no free space remaining on the file system containing the file.

[EDQUOT] The user's quota of disk blocks on the file system containing the file has been exhausted.

[EIO] An I/O error occurred while reading from or writing to the file system.

[EWOULDBLOCK] The file was marked for non-blocking I/O, and no data could be written immediately.

In addition, writev may return one of the following errors:

[EINVAL] iovcnt was less than or equal to 0, or greater than 16.

[EINVAL] One of the iov_len values in the iov array was negative.

[EINVAL] The sum of the iov_len values in the iov array overflowed a 32-bit integer.

SEE ALSO

cntl(2), lseek(2), open(2), pipe(2), select(2)
NAME
abort – generate a fault

DESCRIPTION
*abort* executes an instruction which is illegal in user mode. This causes a signal that normally terminates the process with a core dump, which may be used for debugging.

SEE ALSO
adb(1), sigvec(2), exit(2)

DIAGNOSTICS
Usually “Illegal instruction – core dumped” from the shell.

ERRORS
The abort() function does not flush standard I/O buffers. Use *fflush* (3S).
NAME
  abort – terminate Fortran program

SYNOPSIS
  call abort()

DESCRIPTION
  abort terminates the program that calls it, closing all open files truncated to the current position of the file pointer. The abort usually results in a core dump.

DIAGNOSTICS
  When invoked, abort prints "Fortran abort routine called" on the standard error output. The shell prints the message "abort - core dumped" if a core dump results.

SEE ALSO
  abort(3C)
NAME
    abs – integer absolute value

SYNOPSIS
    abs(i)
    int i;

DESCRIPTION
    abs returns the absolute value of its integer operand.

SEE ALSO
    floor(3M) for fabs

ERRORS
    Applying the abs function to the most negative integer generates a result which is the most
    negative integer. That is,
    abs(0x80000000)
    returns 0x80000000 as a result.
NAME
access – determine accessibility of a file

SYNOPSIS
integer function access (name, mode)
characters(s) name, mode

DESCRIPTION
access checks the given file, name, for accessibility with respect to the caller according to
mode. mode may include in any order and in any combination one or more of:

r       test for read permission
w       test for write permission
x       test for execute permission
(blank)  test for existence

An error code is returned if either argument is illegal, or if the file cannot be accessed in all of
the specified modes. 0 is returned if the specified access would be successful.

FILES
/usr/lib/libU77.a

SEE ALSO
access(2), perror(3F)

ERRORS
Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.
NAME
alarm – schedule signal after specified time

SYNOPSIS
alarm(seconds)
unsigned seconds;

DESCRIPTION
This interface is made obsolete by setitimer(2).

>.alarm causes signal SIGALRM, see sigvec(2), to be sent to the invoking process in a
number of seconds given by the argument. Unless caught or ignored, the signal terminates the
process.

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is 0,
any alarm request is canceled. Because of scheduling delays, resumption of execution of when
the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is
2147483647 seconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO
sigpause(2), sigvec(2), signal(3C), sleep(3), ualarm(3), usleep(3)
NAME
alarm — execute a subroutine after a specified time

SYNOPSIS
    integer function alarm (time, proc)
    integer time
    external proc

DESCRIPTION
    This routine arranges for subroutine proc to be called after time seconds. If time is "0", the
    alarm is turned off and no routine will be called. The returned value will be the time remaining
    on the last alarm.

FILES
    /usr/lib/libU77.a

SEE ALSO
    alarm(3C), sleep(3F), signal(3F)

BUGS
    Alarm and sleep interact. If sleep is called after alarm, the alarm process will never be called.
    SIGALRM will occur at the lesser of the remaining alarm time or the sleep time.
NAME
asinh, acosh, atanh – inverse hyperbolic functions

SYNOPSIS
#include <math.h>
double asinh(x)
double x;
double acosh(x)
double x;
double atanh(x)
double x;

DESCRIPTION
These functions compute the designated inverse hyperbolic functions for real arguments.

ERROR (due to Roundoff etc.)
These functions inherit much of their error from log1p described in exp(3M).

DIAGNOSTICS
Acosh returns the default quiet NaN if the argument is less than 1.
Atanh returns the default quiet NaN if the argument has absolute value bigger than or equal to 1.

SEE ALSO
math(3M), exp(3M)

AUTHOR
W. Kahan, Kwok–Choi Ng
NAME
assert – program verification

SYNOPSIS
#include <assert.h>
assert(expression)

DESCRIPTION
assert is a macro that indicates expression is expected to be true at this point in the program. It causes an exit(2) with a diagnostic comment on the standard output when expression is false (0). Compiling with the cc(1) option -DNDEBUG effectively deletes assert from the program.

DIAGNOSTICS
‘Assertion failed: file f line n.’ f is the source file and n the source line number of the assert statement.
NAME
　atof, atoi, atol – convert ASCII to numbers

SYNOPSIS
   double atof(nptr)
   char *nptr;
   atoi(nptr)
   char *nptr;
   long atol(nptr)
   char *nptr;

DESCRIPTION
   These functions convert a string pointed to by nptr to floating, integer, and long integer
   representation respectively. The first unrecognized character ends the string.

   atof recognizes an optional string of spaces, then an optional sign, then a string of digits
   optionally containing a decimal point, then an optional ‘e’ or ‘E’ followed by an optionally
   signed integer.

   atoi and atol recognize an optional string of spaces, then an optional sign, then a string of
   digits.

SEE ALSO
   scanf(3S)

ERRORS
   There are no provisions for overflow.
NAME
bcopy, bcmp, bzero, ffs – bit and byte string operations

SYNOPSIS
bcopy(src, dst, length)
char *src, *dst;
int length;
bcmp(b1, b2, length)
char *b1, *b2;
int length;
bzero(b, length)
char *b;
int length;
ffs(i)
int i;

DESCRIPTION
The functions bcopy, bcmp, and bzero operate on variable length strings of bytes. They do not check for null bytes as the routines in string(3) do.
bcopy copies length bytes from string src to the string dst.
bcmp compares byte string b1 against byte string b2, returning zero if they are identical, non-zero otherwise. Both strings are assumed to be length bytes long.
bzero places length 0 bytes in the string b1.
ffs find the first bit set in the argument passed it and returns the index of that bit. Bits are numbered starting at 1. A return value of 0 indicates the value passed is zero.

ERRORS
The bcopy routine take parameters backwards from strcpy.
NAME
   htonl, htons, ntohl, ntohs – convert values between host and network byte order

SYNOPSIS
   #include <sys/types.h>
   #include <netinet/in.h>
   netlong = htonl(hostlong);
   u_long netlong, hostlong;
   netshort = htons(hostshort);
   u_short netshort, hostshort;
   hostlong = ntohl(netlong);
   u_long hostlong, netlong;
   hostshort = ntohs(netshort);
   u_short hostshort, netshort;

DESCRIPTION
   These routines convert 16 and 32 bit quantities between network byte order host byte order.
   On machines such as the SUN these routines are defined as null macros in the include file
   <netinet/in.h>.
   These routines are most often used in conjunction with Internet addresses and ports as
   returned by gethostbyname(3N) and getservent(3N).

SEE ALSO
   gethostbyname(3N), getservent(3N)

ERRORS
   The VAX handles bytes backwards from most everyone else in the world. This is not
   expected to be fixed in the near future.
NAME
  chdir – change default directory

SYNOPSIS
  integer function chdir (dirname)
  character*(s)  dirname

DESCRIPTION
  The default directory for creating and locating files will be changed to dirname. Zero is
  returned if successful; an error code otherwise.

FILES
  /usr/lib/libU77.a

SEE ALSO
  chdir(2), cd(1), perror(3F)

BUGS
  Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.
  Use of this function may cause inquire by unit to fail.
NAME
   chmod – change mode of a file

SYNOPSIS
   integer function chmod (name, mode)
   character(*) name, mode

DESCRIPTION
   This function changes the filesystem mode of file name. Mode can be any specification recognized by chmod(1). Name must be a single pathname.
   The normal returned value is 0. Any other value will be a system error number.

FILES
   /usr/lib/libU77.a
   /bin/chmod              exec'ed to change the mode.

SEE ALSO
   chmod(1)

BUGS
   Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.
NAME

crypt, setkey, encrypt – DES encryption

SYNOPSIS

```
char *crypt(key, salt)
char *key, *salt;
setkey(key)
char *key;
encrypt(block, edflag)
char *block;
cc ... -lcrypt
```

DESCRIPTION

NOTE: By default, setkey is not available, and encrypt ignores the value of edflag (it is always treated as 0). Standard versions of these routines are available in the crypt library (/usr/lib/libcrypt.a), which is available in the USA version of UNIX-BSD.

crypt is the password encryption routine. It is based on the NBS (National Bureau of Standards) Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to crypt is normally a user's typed password. The second is a 2-character string chosen from the set [a-zA-Z0-9]. The salt string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The other entries provide (rather primitive) access to the actual DES algorithm. The argument of setkey is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine.

The argument to the encrypt entry is likewise a character array of length 64 containing 0's and 1's. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by setkey. If edflag is 0, the argument is encrypted; if non-zero, it is decrypted.

SEE ALSO

passwd(1), passwd(5), login(1), getpass(3)

ERRORS

The return value points to static data whose content is overwritten by each call.
NAME
cmtime, localtime, gmtime, asctime, timezone, tzset — convert date and time to ASCII

SYNOPSIS
void tzset();
char *ctime(clock);
time_t *clock;
#include <time.h>
char *asctime(tm);
struct tm *tm;
struct tm *localtime(clock);
time_t *clock;
struct tm *gmtime(clock);
time_t *clock;
char *timezone(zone, dst)

DESCRIPTION
tzset uses the value of the environment variable TZ to set up the time conversion information used by localtime.

If TZ does not appear in the environment, the TZDEFAULT file (as defined in tzfile.h) is used by localtime. If this file fails for any reason, the GMT offset as provided by the kernel is used. In this case, DST is ignored, resulting in the time being incorrect by some amount if DST is currently in effect. If this fails for any reason, GMT is used.

If TZ appears in the environment but is value is a null string, Greenwich Mean Time is used; if TZ appears and begins with a slash, it is used as the absolute pathname of the tzfile(5)-format file from which to read the time conversion information; if TZ appears and begins with a character other than a slash, it’s used as a pathname relative to the system time conversion information directory, defined as TZDIR in the include file tzfile.h. If this file fails for any reason, GMT is used.

Programs that always wish to use local wall clock time should explicitly remove the environmental variable TZ with unsetenv (3).

cmtime converts a longer integer, pointed to by clock, such as returned by time(3c) into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

Sun Sep 16 01:03:52 1973

localtime and gmtime return pointers to structures containing the broken-down time. localtime corrects for the time zone and possible daylight savings time; gmtime converts directly to GMT, which is the time UNIT uses. asctime converts a broken-down time to ASCII and returns a pointer to a 26-character string.

The structure declaration from the include file is:

struct tm {
    int tm_sec;  /* 0-59 seconds */
    int tm_min;  /* 0-59 minutes */
    int tm_hour; /* 0-23 hour */
    int tm_mday; /* 1-31 day of month */
    int tm_mon;  /* 0-11 month */
    int tm_year; /* 0- year - 1900 */
    int tm_wday; /* 0-6 day of week (Sunday = 0) */
int tm_yday;  /* 0-365 day of year */
int tm_isdst; /* flag: daylight savings time in effect */
char *tm_zone;  /* abbreviation of timezone name */
long tm_gmtoff; /* offset from GMT in seconds */
};

tm_isdst is non-zero if a time zone adjustment such as Daylight Savings time is in effect.

tm_gmtoff is the offset (in seconds) of the time represented from GMT, with positive values indicating East of Greenwich.

timezone remains for compatibility reasons only; it's impossible to reliably map timezone's arguments zone, a "minutes west of GMT " value and dst, a "daylight saving time in effect" flag) to a time zone abbreviation.

If the environmental string TZNAME exists, timezone returns its value, unless it consists of two comma separated strings, in which case the second string is returned if dst is non-zero, else the first string. If TZNAME doesn't exist, zone is checked for equality with a built-in table of values, in which case timezone returns the time zone or daylight time zone abbreviation associated with that value. If the requested zone does not appear in the table, the difference from GMT is returned; e.g., in Afghanistan, timezone((60+4+30), 0) is appropriate because it is 4:30 ahead of GMT, and the return string GMT+430 is returned. Programs that in the past used the timezone function should return the zone name as set by localtime to assure correctness.

FILES
/etc/zoneinfo time zone information directory
/etc/zoneinfo/localtime local time zone file

SEE ALSO
gettimeofday(2), getenv(3), time(3c), tzfile(5), environ(7)

NOTE
The return values point to static data whose content is overwritten by each call. The tm_zone field of a returned struct tm points to a static array of characters, which will also be overwritten at the next call (and by calls to tzset).
NAME

isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii, toupper,tolower, toascii – character classification macros

SYNOPSIS

#include <ctype.h>
isalpha(c)
...

DESCRIPTION

These macros classify ASCII coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. isascii and toascii are defined on all integer values; the rest are defined only where isascii is true and on the single non-ASCII value EOF (see stdio(3S)).

isalpha c is a letter
isupper c is an upper case letter
islower c is a lower case letter
isdigit c is a digit
isxdigit c is a hex digit
isalnum c is an alphanumeric character
isspace c is a space, tab, carriage return, newline, vertical tab, or formfeed
ispunct c is a punctuation character (neither control nor alphanumeric)
isprint c is a printing character, code 040(8) (space) through 0176 (tilde)
isgraph c is a printing character, similar to isprint except false for space.
iscntrl c is a delete character (0177) or ordinary control character (less than 040).
isascii c is an ASCII character, code less than 0200
tolower c is converted to lower case. Return value is undefined if not isupper(c).
toupper c is converted to upper case. Return value is undefined if not islower(c).
toascii c is converted to be a valid ascii character.

SEE ALSO

ascii(7)
NAME
curses - screen functions with "optimal" cursor motion

SYNOPSIS
cc [ flags ] files -leurses -ltermcap [ libraries ]

DESCRIPTION
These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen, and the user sets up an image of a new one. Then the refresh() tells the routines to make the current screen look like the new one. In order to initialize the routines, the routine initscr() must be called before any of the other routines that deal with windows and screens are used. The routine endwin() should be called before exiting.

SEE ALSO
Screen Updating and Cursor Movement Optimization: A Library Package, Ken Arnold, ioctl(2), getenv(3), tty(4), termcap(5)

AUTHOR
Ken Arnold

FUNCTIONS
addch(ch)  add a character to stdscr
addstr(str) add a string to stdscr
box(win,vert,hor) draw a box around a window
cbreak() set cbreak mode
clear() clear stdscr
clearok(scr,boolf) set clear flag for scr
clrbot() clear to bottom on stdscr
clreol() clear to end of line on stdscr
delch() delete a character
deleteln() delete a line
delwin(win) delete win
echo() set echo mode
eendwin() end window modes
erase() erase stdscr
flusok(win,boolf) set flush-on-refresh flag for win
getch() get a char through stdscr
getcap(name) get terminal capability name
getstr(str) get a string through stdscr
getmode() get tty modes
getxwin(y,x) get (y,x) co-ordinates
inch() get char at current (y,x) co-ordinates
initscr() initialize screens
insch(c) insert a char
insertln() insert a line
leaveok(win,boolf) set leave flag for win
longname(termbuf,name) get long name from termbuf
move(y,x) move to (y,x) on stdscr
mvcur(lstxy,.lstx,newy,newx) actually move cursor
newwin(lines,cols,begins_y,begins_x) create a new window
nIl() set newline mapping
nocbreak() unset cbreak mode
noecho() unset echo mode
norl() unset newline mapping
noraw() unset raw mode
overlay(win1,win2) overlay win1 on win2
overwrite(win1,win2)
printf(fmt, arg1, arg2, ...)
raw()
refresh()
resetty()
savetty()
scanw(fmt, arg1, arg2, ...)
scroll(win)
scrollOk(win, boolf)
setterm(name)
standend()
standout()
subwin(win, lines, cols, begin_y, begin_x)
touchline(win, y, sx, ex)
touchoverlap(win1, win2)
touchwin(win)
unctrl(ch)
waddch(win, ch)
waddstr(win, str)
wclear(win)
wclrtobot(win)
wclroteol(win)
wdelch(win, c)
wdeteleln(win)
erase(win)
wgetch(win)
wgetstr(win, str)
winch(win)
winsch(win, c)
winsertln(win)
wmove(win, y, x)
wprintw(win, fmt, arg1, arg2, ...)
wrefresh(win)
wscanf(fmt, arg1, arg2, ...)
wstandend(win)
wstandout(win)

overwrite win1 on top of win2
printf on stdscr
set raw mode
make current screen look like stdscr
reset tty flags to stored value
stored current tty flags
scanf through stdscr
scroll win one line
set scroll flag
set term variables for name
end standout mode
start standout mode
create a subwindow
mark line y sx through sy as changed
mark overlap of win1 on win2 as changed
"change" all of win
printable version of ch
add char to win
add string to win
clear win
clear to bottom of win
clear to end of line on win
delete char from win
delete line from win
erase win
get a char through win
get a string through win
get char at current (y,x) in win
insert char into win
insert line into win
set current (y,x) co-ordinates on win
printf on win
make screen look like win
scanf through win
end standout mode on win
start standout mode on win

ERRORS
NAME
disassembler – disassemble a MIPS instruction and print the results

SYNOPSIS
int disassembler (iadr, regstyle, get_symname, get_regvalue, get_bytes, print_header)
unsigned iadr;
int regstyle;
char *(get_symname)();
int *(get_regvalue)();
long *(get_bytes)();
void *(print_header)();

DESCRIPTION
Disassembler disassembles and prints a MIPS machine instruction on stdout.

iadr is the instruction address to be disassembled. Regstyle specifies how registers are named
in the disassembly; if the value is 0, compiler names are used; otherwise, hardware names are
used.

The next four arguments are function pointers, most of which give the caller some flexibility in
the appearance of the disassembly. The only function that MUST be provided is get_bytes. All
other functions are optional. Get_bytes is called with no arguments and returns the next
byte(s) to disassemble.

Get_symname is passed an address, which is the target of a jal instruction. If NULL is returned
or if get_symname is NULL, the disassembler prints the address; otherwise, the string name is
printed as returned from get_symname. If get_regvalue is not NULL, it is passed a register
number and returns the current contents of the specified register. Disassembler prints this
information along with the instruction disassembly. If print_header is not NULL, it is passed
the instruction address iadr and the current instruction to be disassembled, which is the return
value from get_bytes. Print_header can use these parameters to print any desired information
before the actual instruction disassembly is printed.

If get_bytes is NULL, the disassembler returns -1 and errno is set to EINVAL; otherwise, the
number of bytes that were disassembled is returned. If the disassembled word is a jump or
branch instruction, the instruction in the delay slot is also disassembled.

The program must be loaded with the object file access routine library libmld.a.

SEE ALSO
ldfcn(4).
NAME
dbminit, fetch, store, delete, firstkey, nextkey – data base subroutines

SYNOPSIS
#include <dbm.h>
typedef struct {
    char *dptr;
    int dsize;
} datum;
dbminit(file)
    char *file;
datum fetch(key)
    datum key;
store(key, content)
    datum key, content;
delete(key)
    datum key;
datum firstkey()
    datum key;
datum nextkey(key)
    datum key;
dbmclose()

DESCRIPTION
Note: the dbm library has been superceded by ndbm(3), and is now implemented using
ndbm. These functions maintain key/content pairs in a data base. The functions will handle
very large (a billion blocks) databases and will access a keyed item in one or two file system
accesses. The functions are obtained with the loader option -ldbm.

keys and contents are described by the datum typedef. A datum specifies a string of dsize
bytes pointed to by dptr. Arbitrary binary data, as well as normal ASCII strings, are allowed.
The data base 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 dbminit. At the time of this call, the
files file.dir and file.pag 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 fetch and data is placed under a key by
store. A key (and its associated contents) is deleted by delete. A linear pass through all keys
in a database may be made, in an (apparently) random order, by use of firstkey and nextkey.
Firstkey will return the first key in the database. With any key nextkey will return the next key
in the database. This code will traverse the data base:

    for (key = firstkey(); key.dptr != NULL; key = nextkey(key))

The routine dbmclose closes the current database.

DIAGNOSTICS
All functions that return an int indicate errors with negative values. A zero return indicates
ok. Routines that return a datum indicate errors with a null (0) dptr.

SEE ALSO
ndbm(3)
ERRORS

The `.pag' file will contain holes so that its apparent size is about four times its actual content. Older UNIX systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp, cat, tp, tar, ar) without filling in the holes.

dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 1024 bytes). Moreover all key/content pairs that hash together must fit on a single block. store will return an error in the event that a disk block fills with inseparable data.

delete does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by firstkey and nextkey depends on a hashing function, not on anything interesting.
NAME
opendir, readdir, telldir, seekdir, rewinddir, closedir – directory operations

SYNOPSIS
#include <sys/types.h>
#include <sys/dir.h>

DIR *opendir(filename)
char *filename;

struct direct *readdir(dirp)
DIR *dirp;

long telldir(dirp)
DIR *dirp;

seekdir(dirp, loc)
DIR *dirp;
long loc;

rewinddir(dirp)
DIR *dirp;

closedir(dirp)
DIR *dirp;

DESCRIPTION
opendir opens the directory named by filename and associates a directory stream with it. opendir returns a pointer to be used to identify the directory stream in subsequent operations. The pointer NULL is returned if filename cannot be accessed, or if it cannot malloc(3) enough memory to hold the whole thing.

readdir returns a pointer to the next directory entry. It returns NULL upon reaching the end of the directory or detecting an invalid seekdir operation.

telldir returns the current location associated with the named directory stream.

seekdir sets the position of the next readdir operation on the directory stream. The new position reverts to the one associated with the directory stream when the telldir operation was performed. Values returned by telldir are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the telldir value may be invalidated due to undetected directory compaction. It is safe to use a previous telldir value immediately after a call to opendir and before any calls to readdir.

rewinddir resets the position of the named directory stream to the beginning of the directory.

closedir closes the named directory stream and frees the structure associated with the DIR pointer.

Sample code which searches a directory for entry "name" is:

len = strlen(name);
dirp = opendir(".");
for (dp = readdir(dirp); dp != NULL; dp = readdir(dirp))
   if (dp->d_namlen == len && !strcmp(dp->d_name, name)) {
      closedir(dirp);
      return FOUND;
   }
closedir(dirp);
return NOT_FOUND;
SEE ALSO
open(2), close(2), read(2), lseek(2), dir(5)
NAME
ecvt, fcvt, gcvt – output conversion

SYNOPSIS

    char *ecvt(value, ndigit, decept, sign)
    double value;
    int ndigit, *decept, *sign;
    char *fcvt(value, ndigit, decept, sign)
    double value;
    int ndigit, *decept, *sign;
    char *gcvt(value, ndigit, buf)
    double value;
    char *buf;

DESCRIPTION
ecvt converts the value to a null-terminated string of ndigit ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through decept (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by sign is non-zero, otherwise it is zero. The low-order digit is rounded.

fcvt is identical to ecvt, except that the correct digit has been rounded for Fortran F-format output of the number of digits specified by ndigits.

gcvt converts the value to a null-terminated ASCII string in buf and returns a pointer to buf. It attempts to produce ndigit significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

SEE ALSO
printf(3)

ERRORS
The return values point to static data whose content is overwritten by each call.
NAME
emulate_branch – MIPS branch emulation

SYNOPSIS
#include <signal.h>

emulate_branch(sc, branch_instruction)
struct sigcontext *sc;
unsigned long branch_instruction;
execute_branch(branch_instruction)
unsigned long branch_instruction;

DESCRIPTION
emulate_branch is passed a signal context structure and a branch instruction. It emulates the
branch based on the register values in the signal context structure. It modifies the value of the
program counter in the signal context structure (sc.pc) to the target of the branch instruction.
The program counter must initially be pointing at the branch and the register values must be
those at the time of the branch. If the branch is not taken the program counter is advanced
to point to the instruction after the delay slot (sc.pc += 8).

In the case the branch instruction is a branch on coprocessor 2 or 3 instruction emulate_branch
calls execute_branch to execute the branch in data space to determine if it is taken or not.
can’t emulate or execute the branch currently.

RETURN VALUE
emulate_branch returns a 0 if the branch was emulated successfully. An non-zero value indi-
cates the value passed as a branch instruction was not a branch instruction.
execute_branch returns non-zero on taken branches and zero on non-taken branches.

ALSO SEE
sigvec(2), cache_flush(3) signal(2), sigset(2)

ERRORS
Since execute_branch in only intended to be used by emulate_branch it does not check it’s
parameter to see if in fact it is a branch instruction. It is really a stop gap in case a coproces-
sor is added without the kernel fully supporting it (which is unlikely).
NAME
emulate_branch – MIPS branch emulation

SYNOPSIS
#include <signal.h>
emulate_branch(sep, branch_instruction)
struct sigcontext *sep;
unsigned long branch_instruction;

DESCRIPTION
Emulate_branch is passed a signal context structure and a branch instruction. It emulates the
branch based on the register values in the signal context structure. It modifies the value of the
program counter in the signal context structure (sc_pc) to the target of the branch instruction.
The program counter must initially be pointing at the branch and the register values must be
those at the time of the branch. If the branch is not taken the program counter is advanced
to point to the instruction after the delay slot (sc_pc += 8).

In the case the branch instruction is a branch on coprocessor 2 or 3 instruction emulate_branch
can’t emulate or execute the branch currently.

RETURN VALUE
Emulate_branch returns a 0 if the branch was emulated successfully. An non-zero value indi-
cates the value passed as a branch instruction was not a branch instruction.

ALSO SEE
signal(2), sigset(2)
NAME
  end, etext, edata – last locations in program
  eprol, _ftext, _fdata, _fbss – first locations in program
  _procedure_table, _procedure_table_size, _procedure_string_table – runtime procedure table

SYNOPSIS
  #include <syms.h>
  extern _END;
  extern _ETEXT;
  extern _EDATA;
  extern eprol;
  extern _FTEXT;
  extern _FDATA;
  extern _FBSS;
  extern _PROCEDURE_TABLE;
  extern _PROCEDURE_TABLE_SIZE;
  extern _PROCEDURE_STRING_TABLE;

DESCRIPTION
  These names refer neither to routines nor to locations with interesting contents except for
  _PROCEDURE_TABLE and _PROCEDURE_STRING_TABLE. Except for eprol these are all
  names of loader defined symbols. The address of _ETEXT is the first address above the pro-
  gram text, _EDATA is above the initialized data region, _END is above the uninitialized data
  region, and eprol is the first instruction of the user’s program that follows the runtime startup
  routine.

  When execution begins, the program break coincides with _END, but it is reset by the routines
  brk(2), malloc(3), standard input/output (stdio(3)), the profile (-p) option of cc(1), etc. The
  current value of the program break is reliably returned by ‘sbrk(0)’, see brk(2).

  The loader defined symbols _PROCEDURE_TABLE, _PROCEDURE_TABLE_SIZE and
  _PROCEDURE_STRING_TABLE refer to the data structures of the runtime procedure table.
  Since these are loader defined symbols the data structures are build by ld(1) only if they are
  referenced. See the include file <sym.h> for the definition of the runtime procedure table
  and see the include file <exception.h> for its uses.

SEE ALSO
  brk(2), malloc(3)
NAME
   end, etext, edata – last locations in program
   eprol, _ftext, _fdata, _fbss – first locations in program
   _procedure_table, _procedure_table_size, _procedure_string_table – runtime procedure table

SYNOPSIS
   #include <syms.h>
   extern _END;
   extern _ETEXT;
   extern _EDATA;
   extern eprol;
   extern _FTEXT;
   extern _FDATA;
   extern _FBSS;
   extern _PROCEDURE_TABLE;
   extern _PROCEDURE_TABLE_SIZE;
   extern _PROCEDURE_STRING_TABLE;

DESCRIPTION
These names refer neither to routines nor to locations with interesting contents except for
_PROCEDURE_TABLE and _PROCEDURE_STRING_TABLE. Except for eprol these are all
names of loader defined symbols. The address of _ETEXT is the first address above the pro-
gram text, _EDATA is above the initialized data region, _END is above the uninitialized data
region, and eprol is the first instruction of the user’s program that follows the runtime startup
routine.

When execution begins, the program break coincides with _END, but it is reset by the routines
brk(2), malloc(3), standard input/output (stdio(3)), the profile (-p) option of cc(1), etc. The
current value of the program break is reliably returned by ‘sbrk(0)’, see brk(2).

The loader defined symbols _PROCEDURE_TABLE, _PROCEDURE_TABLE_SIZE and
_PROCEDURE_STRING_TABLE refer to the data structures of the runtime procedure table.
Since these are loader defined symbols the data structures are build by ld(1) only if they are
referenced. See the include file <sym.h> for the definition of the runtime procedure table
and see the include file <exception.h> for its uses.

SEE ALSO
   brk(2), malloc(3)
NAME
ethers, ether_ntoa, ether_aton, ether_ntohost, ether_hostton, ether_line - Ethernet address mapping operations

SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/ether.h>

char *
ether_ntoa(e)
   struct ether_addr *e;

struct ether_addr *
ether_aton(s)
   char *s;

ether_ntohost(hostname, e)
   char *hostname;
   struct ether_addr *e;

ether_hostton(hostname, e)
   char *hostname;
   struct ether_addr *e;

der_line(l, e, hostname)
   char *l;
   struct ether_addr *e;
   char *hostname;

DESCRIPTION
ether_ntoa, ether_aton, ether_ntohost, ether_hostton, ether_line

These routines are useful for mapping 48 bit Ethernet numbers to their ASCII representations or their corresponding host names, and vice versa.

The function ether_ntoa converts a 48 bit Ethernet number pointed to by e to its standard ASCII+1 representation; it returns a pointer to the ASCII string. The representation is of the form: "x:x:x:x:x:x" where x is a hexadecimal number between 0 and ff. The function ether_aton converts an ASCII string in the standard representation back to a 48 bit Ethernet number; the function returns NULL if the string cannot be scanned successfully.

The function ether_ntohost maps an Ethernet number (pointed to by e) to its associated hostname. The string pointed to by hostname must be long enough to hold the hostname and a null character. The function returns zero upon success and non-zero upon failure. Inversely, the function ether_hostton maps a hostname string to its corresponding Ethernet number; the function modifies the Ethernet number pointed to by e. The function also returns zero upon success and non-zero upon failure.

The function ether_line scans a line (pointed to by l) and sets the hostname and the Ethernet number (pointed to by e). The string pointed to by hostname must be long enough to hold the hostname and a null character. The function returns zero upon success and non-zero upon failure. The format of the scanned line is described by ethers(5).

FILES
/etc/ethers (or the yellowpages' maps ethers.byaddr and ethers.byname)
SEE ALSO
ethers(5)
NAME

erf, erfc – error functions

SYNOPSIS

#include <math.h>

double erf(x)

double x;

double erfc(x)

double x;

DESCRIPTION

Erf (x) returns the error function of x; where erf (x) := (2/√π) ∫₀^x exp(-t²) dt.

Erfc (x) returns 1.0–erf (x).

The entry for erfc is provided because of the extreme loss of relative accuracy if erf (x) is
called for large x and the result subtracted from 1. (e.g. for x = 10, 12 places are lost).

SEE ALSO

math(3M)
NAME
etime, dtime – return elapsed execution time

SYNOPSIS
function etime (tarray)
  real tarray(2)

function dtime (tarray)
  real tarray(2)

DESCRIPTION
These two routines return elapsed runtime in seconds for the calling process. Dtime returns the elapsed time since the last call to dtime, or the start of execution on the first call.
The argument array returns user time in the first element and system time in the second element. The function value is the sum of user and system time.
The resolution of all timing is 1/HZ sec. where HZ is currently 60.

FILES
/usr/lib/libU77.a

SEE ALSO
times(2)
NAME
examples – library of sample programs

SYNOPSIS
examples

DESCRIPTION
examples is a library containing sample programs to illustrate Ada language use and
demonstrate the capabilities of the language, including those provided by the packages in the
standard, verdirlib, and publiclib libraries.

Note: programs in the examples are neither supported nor warranted by MIPS.
The directory contains the program files listed below.

arguments.a uses package COMMAND_LINE from verdirlib to print program
arguments and environment variables.
date uses package CALENDAR from standard to print current date and
time.

hanoi.a, termbody.a, termspec.a
demonstrates solution to "Towers of Hanoi" problem.

hello a typical first program, which uses package TEXT_IO from standard to
print the message "hello, world".

mortgage.a uses package MATH from verdirlib to calculate mortgage payments.

queens.a provides a solution of the "8 Queens" chess problem gerneralized for any
board with sides of 4-12 squares.

random.a uses packages CALENDAR from standard to create pseudo-random
numbers.

slideshow.a uses the package CURSES in publiclib and illustrates background tasks.

sort_file sorts lines in a file within specifies columns.

sort_integer.a uses packages ORDERING form verdirlib to sort input of IO integer in
ascending and descending order.

ucp, uctrans.a uses package CALENDAR from standard to maintain a calendar file;
these illustrate the translation of a program from Pascal to Ada. ucp is
in Pascal, and uctrans.a is a close translation of UC.PAS to Ada.

FILES
/usr/vads5/examples/*

SEE ALSO
publiclib, standard, verdirlib
NAME
execl, execv, execlp, execvp, exec, execve, exec, environ – execute a file

SYNOPSIS
execl(name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn;
execl(name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn, *envp[];
execl(name, arg0, arg1, ..., argn, 0, envp)
char *name, *arg0, *arg1, ..., *argn, *envp[];
exect(name, argv, envp)
char *name, *argv[], *envp[];
extern char **environ;

DESCRIPTION
These routines provide various interfaces to the execve system call. Refer to execve(2) for a
description of their properties; only brief descriptions are provided here.
exec in all its forms overlays the calling process with the named file, then transfers to the entry
point of the core image of the file. There can be no return from a successful exec; the calling
core image is lost.
The name argument is a pointer to the name of the file to be executed. The pointers arg[0],
arg[1] ... address null-terminated strings. Conventionally arg[0] is the name of the file.
Two interfaces are available. execl is useful when a known file with known arguments is being
called; the arguments to execl are the character strings constituting the file and the arguments;
the first argument is conventionally the same as the file name (or its last component). A 0
argument must end the argument list.
The execv version is useful when the number of arguments is unknown in advance; the argu-
ments to execv are the name of the file to be executed and a vector of strings containing the
arguments. The last argument string must be followed by a 0 pointer.
The execl version is used when the executed file is to be manipulated with ptrace(2). The pro-
gram is forced to single step a single instruction giving the parent an opportunity to manipulate
its state. On the VAX-11 this is done by setting the trace bit in the process status longword.
When a C program is executed, it is called as follows:
main(argc, argv, envp)
int argc;
char **argv, **envp;

where argc is the argument count and argv is an array of character pointers to the arguments
themselves. As indicated, argc is conventionally at least one and the first member of the array
points to a string containing the name of the file.
argv is directly usable in another execv because argv[argc] is 0.

envp is a pointer to an array of strings that constitute the environment of the process. Each
string consists of a name, an "=" and a null-terminated value. The array of pointers is ter-
minalized by a null pointer. The shell sh(1) passes an environment entry for each global shell
variable defined when the program is called. See environ(7) for some conventionally used
names. The C run-time start-off routine places a copy of envp in the global cell environ, which
is used by execv and execl to pass the environment to any subprograms executed by the current
program.
execp and execvp are called with the same arguments as execl and execv, but duplicate the shell’s actions in searching for an executable file in a list of directories. The directory list is obtained from the environment.

FILES
/bin/sh shell, invoked if command file found by execp or execvp

SEE ALSO
execve(2), fork(2), environ(7), csh(1)

DIAGNOSTICS
If the file cannot be found, if it is not executable, if it does not start with a valid magic number (see a.out(5)), if maximum memory is exceeded, or if the arguments require too much space, a return constitutes the diagnostic; the return value is -1. Even for the super-user, at least one of the execute-permission bits must be set for a file to be executed.

ERRORS
If execvp is called to execute a file that turns out to be a shell command file, and if it is impossible to execute the shell, the values of argv[0] and argv[-1] will be modified before return.
NAME
exit — terminate a process after flushing any pending output

SYNOPSIS

exit(status)
int status;

DESCRIPTION
exit terminates a process after calling the Standard I/O library function _cleanup to flush any buffered output. exit never returns.

SEE ALSO
exit(2), intro(3)
NAME
exp, expm1, log, log10, log1p, pow – exponential, logarithm, power

SYNOPSIS
#include <math.h>

double exp(x)
double x;

float fexp(float x)
float x;

double expm1(x)
double x;

float fexpm1(float x)
float x;

double log(x)
double x;

float flog(float x)
float x;

double log10(x)
double x;

float flog10(float x)
float x;

double log1p(x)
double x;

float flog1p(float x)
float x;

double pow(x,y)
double x,y;

DESCRIPTION
Exp and fexp return the exponential function of x for double and float data types respectively.

Expm1 and fexpm1 return exp(x)−1 accurately even for tiny x for double and float data types respectively.

Log and flog return the natural logarithm of x for double and float data types respectively.

Log10 and flog10 return the logarithm of x to base 10 for double and float data types respectively.

Log1p and flog1p return log(1+x) accurately even for tiny x for double and float data types respectively.

Pow(x,y) returns xy.

ERROR (due to Roundoff etc.)
exp(x), log(x), expm1(x) and log1p(x) are accurate to within an ulp, and log10(x) to within about 2 ulps; an ulp is one Unit in the Last Place. The error in pow(x,y) is below about 2 ulps when its magnitude is moderate, but increases as pow(x,y) approaches the over/underflow thresholds until almost as many bits could be lost as are occupied by the floating-point format’s exponent field; 11 bits for IEEE 754 Double. No such drastic loss has been exposed by testing; the worst errors observed have been below 300 ulps for IEEE 754 Double. Moderate values of pow are accurate enough that pow(integer,integer) is exact until it is bigger
than \(2^{\ast}53\) for IEEE 754 Double.

**DIAGNOSTICS**

- \(\text{exp}\) returns \(\infty\) when the correct value would overflow, or the smallest non-zero value when the correct value would underflow.
- \(\text{Log}\) and \(\text{log10}\) returns the default quiet \(\text{NaN}\) when \(x\) is less than zero indicating the invalid operation. \(\text{Log}\) and \(\text{log10}\) returns \(-\infty\) when \(x\) is zero.
- \(\text{Pow}\) returns \(\infty\) when \(x\) is 0 and \(y\) is non-positive. \(\text{Pow}\) returns \(\text{NaN}\) when \(x\) is negative and \(y\) is not an integer indicating the invalid operation. When the correct value for \(\text{pow}\) would overflow or underflow, \(\text{pow}\) returns \(\pm\infty\) or 0 respectively.

**NOTES**

\(\text{Pow}(x,0)\) returns \(x^{\ast}0 = 1\) for all \(x\) including \(x = 0, \infty\), and \(\text{NaN}\). Previous implementations of \(\text{pow}\) may have defined \(x^{\ast}0\) to be undefined in some or all of these cases. Here are reasons for returning \(x^{\ast}0 = 1\) always:

1. Any program that already tests whether \(x\) is zero (or infinite or \(\text{NaN}\)) before computing \(x^{\ast}0\) cannot care whether \(0^{\ast}0 = 1\) or not. Any program that depends upon \(0^{\ast}0\) to be invalid is dubious anyway since that expression's meaning and, if invalid, its consequences vary from one computer system to another.

2. Some Algebra texts (e.g. Sigler's) define \(x^{\ast}0 = 1\) for all \(x\), including \(x = 0\). This is compatible with the convention that accepts \(a[0]\) as the value of polynomial

\[ p(x) = a[0]x^{\ast}0 + a[1]x^{\ast}1 + a[2]x^{\ast}2 + \ldots + a[n]x^{\ast}n \]

at \(x = 0\) rather than reject \(a[0]x^{\ast}0\) as invalid.

3. Analysts will accept \(0^{\ast}0 = 1\) despite that \(x^{\ast}y\) can approach anything or nothing as \(x\) and \(y\) approach 0 independently. The reason for setting \(0^{\ast}0 = 1\) anyway is this:

   - If \(x(z)\) and \(y(z)\) are any functions analytic (expandable in power series) in \(z\) around \(z = 0\), and if there \(x(0) = y(0) = 0\), then \(x(z)^{\ast}y(z) \to 1\) as \(z \to 0\).

4. If \(0^{\ast}0 = 1\), then \(\infty^{\ast}0 = 1/0^{\ast}0 = 1\) too; and then \(\text{NaN}^{\ast}0 = 1\) too because \(x^{\ast}0 = 1\) for all finite and infinite \(x\), i.e., independently of \(x\).

**SEE ALSO**

- math(3M)

**AUTHOR**

Kwok-Choi Ng, W. Kahan
NAME
close, flush – close or flush a stream

SYNOPSIS
#include <stdio.h>
fclose(stream)
FILE *stream;
flush(stream)
FILE *stream;

DESCRIPTION
fclose causes any buffers for the named stream to be emptied, and the file to be closed.
Buffers allocated by the standard input/output system are freed.
fclose is performed automatically upon calling exit(3).
flush causes any buffered data for the named output stream to be written to that file. The stream remains open.

SEE ALSO
close(2), fopen(3S), setbuf(3S)

DIAGNOSTICS
These routines return EOF if stream is not associated with an output file, or if buffered data cannot be transferred to that file.
NAME
fdate – return date and time in an ASCII string

SYNOPSIS
subroutine fdate (string)
character(*) string

character(*) function fdate()

DESCRIPTION
 Fdate returns the current date and time as a 24 character string in the format described under ctime(3). Neither 'newline' nor NULL will be included.
 Fdate can be called either as a function or as a subroutine. If called as a function, the calling routine must define its type and length. For example:

    character*24   fdate
    external      fdate

    write(*,*) fdate()

FILES
/usr/lib/libU77.a

SEE ALSO
cftime(3), time(3F), itime(3F), idate(3F), ltime(3F)
NAME
ferror, feof, clearerr, fileno - stream status inquiries

SYNOPSIS
#include <stdio.h>
feof(stream)
FILE *stream;
ferror(stream)
FILE *stream
clearerr(stream)
FILE *stream
fileno(stream)
FILE *stream;

DESCRIPTION
feof returns non-zero when end of file is read on the named input stream, otherwise zero. Unless cleared by clearerr, the end-of-file indication lasts until the stream is closed.
ferror returns non-zero when an error has occurred reading or writing the named stream, otherwise zero. Unless cleared by clearerr, the error indication lasts until the stream is closed.
clearerr resets the error and end-of-file indicators on the named stream.
fileno returns the integer file descriptor associated with the stream, see open(2).
Currently all of these functions are implemented as macros; they cannot be redeclared.

SEE ALSO
fopen(3S), open(2)
NAME
fabs, floor, ceil, rint – absolute value, floor, ceiling, and round-to-nearest functions

SYNOPSIS
#include <math.h>
double floor(x)
double x;
float ffloor(float x)
float x;
double ceil(x)
double x;
float fceil(float x)
float x;
double trunc(x)
double x;
float ftrunc(float x)
float x;
double fabs(x)
double x;
double rint(x)
double x;
double fmod (x, y)
double x, y;

DESCRIPTION
Floor and ffloor returns the largest integer no greater than x for double and float data types respectively.

Ceil and fceil returns the smallest integer no less than x for double and float data types respectively.

Trunc and ftrunc returns the integer (represented as a floating-point number) of x with the fractional bits truncated for double and float data types respectively.

Fabs returns the absolute value |x|.

Rint returns the integer (represented as a double precision number) nearest x in the direction of the prevailing rounding mode.

Fmod returns the floating-point remainder of the division of x by y: zero if y is zero or if x/y would overflow; otherwise the number f with the same sign as x, such that x = iy + f for some integer i, and |f| < |y|.

NOTES
In the default rounding mode, to nearest, rint(x) is the integer nearest x with the additional stipulation that if |rint(x)−x|=1/2 then rint(x) is even. Other rounding modes can make rint act like floor, or like ceil, or round towards zero.

Another way to obtain an integer near x is to declare (in C)
    double x;    int k;    k = x;
The MIPS C compilers rounds x towards 0 to get the integer k. Also note that, if x is larger than k can accommodate, the value of k and the presence or absence of an integer overflow are hard to predict.
The routine fabs is in libc.a rather than libm.a.

SEE ALSO
abs(3), ieee(3M), math(3M)
NAME
  flush – flush output to a logical unit

SYNOPSIS
  subroutine flush (lunit)

DESCRIPTION
  Flush causes the contents of the buffer for logical unit lunit to be flushed to the associated file. This is most useful for logical units 0 and 6 when they are both associated with the control terminal.

FILES
  /usr/lib/libI77.a

SEE ALSO
  fclose(3S)
NAME
fork - create a copy of this process

SYNOPSIS
integer function fork()

DESCRIPTION
Fork creates a copy of the calling process. The only distinction between the 2 processes is that the value returned to one of them (referred to as the 'parent' process) will be the process id of the copy. The copy is usually referred to as the 'child' process. The value returned to the 'child' process will be zero.

All logical units open for writing are flushed before the fork to avoid duplication of the contents of I/O buffers in the external file(s).

If the returned value is negative, it indicates an error and will be the negation of the system error code. See perror(3F).

A corresponding exec routine has not been provided because there is no satisfactory way to retain open logical units across the exec. However, the usual function of fork/exec can be performed using system(3F).

FILES
/usr/lib/libU77.a

SEE ALSO
fork(2), wait(3F), kill(3F), system(3F), perror(3F)
NAME
fopen, freopen, fdopen — open a stream

SYNOPSIS
#include <stdio.h>
FILE *fopen(filename, type)
char *filename, *type;
FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;
FILE *fdopen(fildes, type)
char *type;

DESCRIPTION
fopen opens the file named by filename and associates a stream with it. fopen returns a pointer to be used to identify the stream in subsequent operations.

type is a character string having one of the following values:
"r" open for reading
"w" create for writing
"a" append: open for writing at end of file, or create for writing

In addition, each type may be followed by a "+" to have the file opened for reading and writing. "r+" positions the stream at the beginning of the file, "w+" creates or truncates it, and "a+" positions it at the end. Both reads and writes may be used on read/write streams, with the limitation that an fseek, rewind, or reading an end-of-file must be used between a read and a write or vice-versa.

freopen substitutes the named file in place of the open stream. It returns the original value of stream. The original stream is closed.

freopen is typically used to attach the reopened constant names, stdin, stdout, stderr, to specified files.

fdopen associates a stream with a file descriptor obtained from open, dup, creat, or pipe(2). The type of the stream must agree with the mode of the open file.

SEE ALSO
open(2), fclose(3)

DIAGNOSTICS
fopen and freopen return the pointer NULL if filename cannot be accessed, if too many files are already open, or if other resources needed cannot be allocated.

ERRORS
fdopen is not portable to systems other than UNIX.

The read/write types do not exist on all systems. Those systems without read/write modes will probably treat the type as if the "+" was not present. These are unreliable in any event.

In order to support the same number of open files as does the system, fopen must allocate additional memory for data structures using calloc after 20 files have been opened. This confuses some programs which use their own memory allocators. An undocumented routine, f_prealloc, may be called to force immediate allocation of all internal memory except for buffers.
NAME
fp_class – classes of IEEE floating-point values

SYNOPSIS
#include <fp_class.h>
int fp_class_d(double x);
int fp_class_f(float x);

DESCRIPTION
These routines are used to determine the class of IEEE floating-point values. They return one of the constants in the file <fp_class.h> and never cause an exception even for signaling NaN's. These routines are to implement the recommended function class(x) in the appendix of the IEEE 754-1985 standard for binary floating-point arithmetic.

The constants in <fp_class.h> refer to the following classes of values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_SNAN</td>
<td>Signaling NaN (Not-a-Number)</td>
</tr>
<tr>
<td>FP_QNAN</td>
<td>Quiet NaN (Not-a-Number)</td>
</tr>
<tr>
<td>FP_POS_INF</td>
<td>+∞ (positive infinity)</td>
</tr>
<tr>
<td>FP_NEG_INF</td>
<td>−∞ (negative infinity)</td>
</tr>
<tr>
<td>FP_POS_NORM</td>
<td>positive normalized non-zero</td>
</tr>
<tr>
<td>FP_NEG_NORM</td>
<td>negative normalized non-zero</td>
</tr>
<tr>
<td>FP_POS_DENORM</td>
<td>positive denormalized</td>
</tr>
<tr>
<td>FP_NEG_DENORM</td>
<td>negative denormalized</td>
</tr>
<tr>
<td>FP_POS_ZERO</td>
<td>+0.0 (positive zero)</td>
</tr>
<tr>
<td>FP_NEG_ZERO</td>
<td>−0.0 (negative zero)</td>
</tr>
</tbody>
</table>

ALSO SEE
ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic
NAME
fp_class – classes of IEEE floating-point values

SYNOPSIS
#include <fp_class.h>

int fp_class_d(double x);
int fp_class_f(float x);

DESCRIPTION
These routines are used to determine the class of IEEE floating-point values. They return one of the constants in the file <fp_class.h> and never cause an exception even for signaling NaN’s. These routines are to implement the recommended function class(x) in the appendix of the IEEE 754-1985 standard for binary floating-point arithmetic.

The constants in <fp_class.h> refer to the following classes of values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_SNaN</td>
<td>Signaling NaN (Not-a-Number)</td>
</tr>
<tr>
<td>FP_QNaN</td>
<td>Quiet NaN (Not-a-Number)</td>
</tr>
<tr>
<td>FP_POS_INF</td>
<td>+∞ (positive infinity)</td>
</tr>
<tr>
<td>FP_NEG_INF</td>
<td>-∞ (negative infinity)</td>
</tr>
<tr>
<td>FP_POS_NORM</td>
<td>positive normalized non-zero</td>
</tr>
<tr>
<td>FP_NEG_NORM</td>
<td>negative normalized non-zero</td>
</tr>
<tr>
<td>FP_POS_DENORM</td>
<td>positive denormalized</td>
</tr>
<tr>
<td>FP_NEG_DENORM</td>
<td>negative denormalized</td>
</tr>
<tr>
<td>FP_POS_ZERO</td>
<td>+0.0 (positive zero)</td>
</tr>
<tr>
<td>FP_NEG_ZERO</td>
<td>-0.0 (negative zero)</td>
</tr>
</tbody>
</table>

ALSO SEE
ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic
NAME
fpc – floating-point control registers

SYNOPSIS
#include <sys/fpu.h>
int get_fpc_csr()
int set_fpc_csr(csr)
int csr;
int get_fpc_ier()
int get_fpc_ier()
void set_fpc_led(value)
int value;
int swapRM(x)
int x;
int swapINX(x)
int x;

DESCRIPTION
These routines are to get and set the floating-point control registers of MIPS floating-point
units. All of these routines take and or return their values as 32 bit integers.
The file <sys/fpu.h> contains unions for each of the control registers. Each union contains a
structure that breaks out the bit fields into the logical parts for each control register. This file
also contains constants for fields of the control registers.
All implementations of MIPS floating-point have a control and status register and a implementation revision register. The control and status register is returned by get_fpc_csr. The routine
set_fpc_csr sets the control and status register and returns the old value. The implementation revision register is read-only and is returned by the routine get_fpc_ier.
The R2360 floating-point units (floating-point boards) have two additional control registers. The
exception instruction register is a read-only register and is returned by the routine
get_fpc_ier. The other floating-point control register on the R2360 is the leds register. The low
8 bits corresponds to the leds where a one is off and a zero is on. The leds register is a write-
only register and is set with the routine set_fpc_leds.
The routine swapRN sets only the rounding mode and returns the old rounding mode. The
routine swapINX sets only the sticky inexact bit and returns the old one. The bits in the arguments and return values to swapRN and swapINX are right justified.

ALSO SEE
R2010 Floating Point Coprocessor Architecture
R2360 Floating Point Board Product Description
NAME
fpc - floating-point control registers

SYNOPSIS
#include <mips/fpu.h>
#include <sys/fpu.h>

int get_fpc_csr()
int set_fpc_csr(csr)
int csr;
int get_fpc_irr()
int get_fpc_eir()
void set_fpc_led(value)
int value;
int swapRM(x)
int x;
int swapINX(x)
int x;

DESCRIPTION
These routines are to get and set the floating-point control registers of MIPS floating-point units. All of these routines take and or return their values as 32 bit integers.

The file <mips/fpu.h> <sys/fpu.h> contains unions for each of the control registers. Each union contains a structure that breaks out the bit fields into the logical parts for each control register. This file also contains constants for fields of the control registers.

All implementations of MIPS floating-point have a control and status register and a implementation revision register. The control and status register is returned by get_fpc_csr. The routine set_fpc_csr sets the control and status register and returns the old value. The implementation revision register is read-only and is returned by the routine get_fpc_irr.

The R2360 floating-point units (floating-point boards) have two additional control registers. The exception instruction register is a read-only register and is returned by the routine get_fpc_eir. The other floating-point control register on the R2360 is the leds register. The low 8 bits corresponds to the leds where a one is off and a zero is on. The leds register is a write-only register and is set with the routine set_fpc_leds.

The routine swapRN sets only the rounding mode and returns the old rounding mode. The routine swapINX sets only the sticky inexact bit and returns the old one. The bits in the arguments and return values to swapRN and swapINX are right justified.

ALSO SEE
R2010 Floating Point Coprocessor Architecture
R2360 Floating Point Board Product Description
NAME
   fpi – floating-point interrupt analysis

SYNOPSIS
   #include <fpi.h>
   void fpi()
   void print_fpicounts()
   int fpi_counts[];
   char *fpi_list[];

DESCRIPTION
   MIPS floating-point units generate floating-point interrupts for some classes of operations that occur with low frequency. In these cases the system software then emulates the operation in software. As a program takes floating-point interrupts its performance degrades since the operations are emulated in software. The routines and counters described here are used to analyze the causes of floating-point interrupts.

   The routine fpi makes a sysmips(2) [MIPS_fpsigint] system call to causes floating-point interrupts to generate a SIGFPE. It also sets up a special signal handler for SIGFPE’s. On a floating-point interrupt that signal handler determines the precise cause of the interrupt and increments the appropriate counter in fpi_counts[].

   The routine print_fpicounts prints out the value of the counters and their description on stderr as in the following example:
   
   source signaling NaN = 0
   source quiet NaN = 10
   source denormalized value = 23
   move of zero = 83
   negate of zero = 84
   implemented only in software = 5
   invalid operation = 96
   divide by zero = 3837
   destination overflow = 398
   destination underflow = 489

   The constants in the file <fpi.h> along the counters, fpi_counts[], and the descriptive strings, fpi_list[], can also be used to format messages.

LIMITATIONS
   fpi can’t be used with programs that normally generate SIGFPE’s.

ALSO SEE
   R2010 Floating Point Coprocessor Architecture
   R2360 Floating Point Board Product Description
   sysmips(2) [MIPS_fpsigint].
NAME
   fpi – floating-point interrupt analysis

SYNOPSIS
   #include <fpi.h>
   void fpi()
   void printfpi_counts()
   int fpi_counts[];
   char *fpi_list[];

DESCRIPTION
   MIPS floating-point units generate floating-point interrupts for some classes of operations that occur with low frequency. In these cases the system software then emulates the operation in software. As a program takes floating-point interrupts its performance degrades since the operations are emulated in software. The routines and counters described here are used to analyze the causes of floating-point interrupts.

   The routine fpi makes a fp_sigintr(2) sysmips(2) [MIPS_FPSIGINT] system call to causes floating-point interrupts to generate a SIGFPE. It also sets up a special signal handler for SIGFPE’s. On a floating-point interrupt that signal handler determines the precise cause of the interrupt and increments the appropriate counter in fpi_counts[].

   The routine printfpi_counts prints out the value of the counters and their description on stderr as in the following example:
      
      source signaling NaN = 0
      source quiet NaN = 10
      source denormalized value = 23
      move of zero = 83
      negate of zero = 84
      implemented only in software = 5
      invalid operation = 96
      divide by zero = 3837
      destination overflow = 398
      destination underflow = 489
      
   The constants in the file <fpi.h> along the counters, fpi_counts[], and the descriptive strings, fpi_list[], can also be used to format messages.

LIMITATIONS
   fpi can’t be used with programs that normally generate SIGFPE’s.

ALSO SEE
   R2010 Floating Point Coprocessor Architecture
   R2360 Floating Point Board Product Description
   fp_sigintr(2). sysmips(2) [MIPS_FPSIGINTR].

NAME
fread, fwrite – buffered binary input/output

SYNOPSIS
#include <stdio.h>

fread(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

fwrite(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

DESCRIPTION
fread reads, into a block beginning at ptr, nitems of data of the type of *ptr from the named input stream. It returns the number of items actually read.

If stream is stdin and the standard output is line buffered, then any partial output line will be flushed before any call to read(2) to satisfy the fread.

fwrite appends at most nitems of data of the type of *ptr beginning at ptr to the named output stream. It returns the number of items actually written.

SEE ALSO
read(2), write(2), fopen(3S), getc(3S), putc(3S), gets(3S), puts(3S), printf(3S), scanf(3S)

DIAGNOSTICS
fread and fwrite return 0 upon end of file or error.
NAME
frexp, ldexp, modf – split into mantissa and exponent

SYNOPSIS
double frexp(value, eptr)
double value;
int *eptr;
double ldexp(value, exp)
double value;
double modf(value, iptr)
double value, *iptr;

DESCRIPTION
frexp returns the mantissa of a double value as a double quantity, x, of magnitude less than 1 and stores an integer n such that value = x * 2^n indirectly through eptr.

ldexp returns the quantity value * 2^{exp}.

modf returns the positive fractional part of value and stores the integer part indirectly through iptr.
NAME
  fseek, ftell - reposition a file on a logical unit

SYNOPSIS
  integer function fseek (lunit, offset, from)
  integer offset, from

  integer function ftell (lunit)

DESCRIPTION
  lunit must refer to an open logical unit. offset is an offset in bytes relative to the position
  specified by from. Valid values for from are:
     0 meaning 'beginning of the file'
     1 meaning 'the current position'
     2 meaning 'the end of the file'

The value returned by fseek will be 0 if successful, a system error code otherwise. (See
perror(3F))

Ftell returns the current position of the file associated with the specified logical unit. The value
is an offset, in bytes, from the beginning of the file. If the value returned is negative, it indi-
cates an error and will be the negation of the system error code. (See perror(3F))

FILES
  /usr/lib/libU77.a

SEE ALSO
  fseek(3S), perror(3F)
NAME
fseek, ftell, rewind – reposition a stream

SYNOPSIS
#include <stdio.h>
fseek(stream, offset, ptrname)
FILE *stream;
long offset;
long ftell(stream)
FILE *stream;
rewind(stream)

DESCRIPTION
fseek sets the position of the next input or output operation on the stream. The new position
is at the signed distance offset bytes from the beginning, the current position, or the end of the
file, according as ptrname has the value 0, 1, or 2.

fseek undoes any effects of ungetc(3S).

ftell returns the current value of the offset relative to the beginning of the file associated with
the named stream. It is measured in bytes on UNIX; on some other systems it is a magic
cookie, and the only foolproof way to obtain an offset for fseek.

rewind(stream) is functionally equivalent to fseek(stream, 0L, 0), but it does not return a use-
ful return value.

SEE ALSO
lseek(2), fopen(3S)

DIAGNOSTICS
fseek returns -1 for improper seeks, otherwise zero.
NAME
getarg, iargc – return command line arguments

SYNOPSIS
subroutine getarg (k, arg)
character*(s) arg

function iargc ()

DESCRIPTION
A call to getarg will return the kth command line argument in character string arg. The 0th
argument is the command name.

iargc returns the index of the last command line argument.

FILES
/usr/lib/libU77.a

SEE ALSO
getenv(3F), execve(2)
NAME
getc, fgetc – get a character from a logical unit

SYNOPSIS

integer function getc (char)
character char

integer function fgetc (lunit, char)
character char

DESCRIPTION

These routines return the next character from a file associated with a fortran logical unit, bypassing normal fortran I/O. Getc reads from logical unit 5, normally connected to the control terminal input.

The value of each function is a system status code. Zero indicates no error occurred on the read; -1 indicates end of file was detected. A positive value will be either a UNIX system error code or an f77 I/O error code. See perror(3F).

FILES
/usr/lib/libU77.a

SEE ALSO

getc(3S), intro(2), perror(3F)
NAME
getc, getchar, fgetc, getw – get character or word from stream

SYNOPSIS
#include <stdio.h>
int getc(stream)
FILE *stream;
int getchar()
int fgetc(stream)
FILE *stream;
int getw(stream)
FILE *stream;

DESCRIPTION
getc returns the next character from the named input stream.
getchar() is identical to getc(stdin).

fgetc behaves like getc, but is a genuine function, not a macro; it may be used to save object
text.

getw returns the next int (a 32-bit integer on a VAX-11) from the named input stream. It
returns the constant EOF upon end of file or error, but since that is a good integer value, feof
and ferror(3S) should be used to check the success of getw. getw assumes no special align-
ment in the file.

SEE ALSO
clearrerr(3S), fopen(3S), putc(3S), puts(3S), scanf(3S), fread(3S), ungetc(3S)

DIAGNOSTICS
These functions return the integer constant EOF at end of file, upon read error, or if an
attempt is made to read a file not opened by fopen. The end-of-file condition is remembered,
even on a terminal, and all subsequent attempts to read will return EOF until the condition is
cleared with clearrerr(3S).

ERRORS
Because it is implemented as a macro, getc treats a stream argument with side effects
incorrectly. In particular, 'getc(*f++);' doesn't work sensibly.
NAME
getcwd – get pathname of current working directory

SYNOPSIS
integer function getcwd (dirname)
character(*) dirname

DESCRIPTION
The pathname of the default directory for creating and locating files will be returned in dirname. The value of the function will be zero if successful; an error code otherwise.

FILES
/usr/lib/libU77.a

SEE ALSO
chdir(3F), perror(3F)

BUGS
Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.
NAME
getdiskbyname - get disk description by its name

SYNOPSIS
#include <disktab.h>

struct disktab *
getdiskbyname(name)
char *name;

DESCRIPTION
getdiskbyname takes a disk name (e.g. rm03) and returns a structure describing its geometry information and the standard disk partition tables. All information obtained from the disk-
tab(5) file.

<disktab.h> has the following form:

/ *---------------------------------------------------------------*/
/ * Copyright Unpublished, MIPS Computer Systems, Inc. All Rights | */
/ * Reserved. This software contains proprietary and confidential | */
/ * information of MIPS and its suppliers. Use, disclosure or    | */
/ * reproduction is prohibited without the prior express written | */
/ * consent of MIPS.                                            | */
/ *---------------------------------------------------------------*/
/ * $Header: disktab.h,v 1.6 87/08/04 09:58:11 dce Exp $ */

/ *
disktab.h 4.3  83/08/11 */

/ *
* Disk description table, see disktab(5)
*/

ifndef mips
#include <sys/types.h>
#include <mips/dvh.h>
#endif

ifndef NPARTAB
#define NPARTAB  8
#else
*/
* Number of user partitions is the total number of partitions minus
* the volume header, sector forwarding, and entire volume partitions.
*/
#define NPARTAB (NPARTAB - 3)
#endif

#define DISKTAB "*/etc/disktab"

struct disktab {
    char *d_name; /* drive name */
    char *d_type; /* drive type */
    int d_secsize; /* sector size in bytes */
    int d_ntracks; /* # tracks/cylinder */
    int d_nsectors; /* # sectors/track */
    int d_ncylinders; /* # cylinders */
}
int d_rpm; /* revolutions/minute */
int d_badsectforw; /* supports DEC bad4 std */
int d_sectoffset; /* use sect rather than cyl offsets */
struct partition {
    int p_size; /* #sectors in partition */
    short p_bsize; /* block size in bytes */
    short p_fsize; /* frag size in bytes */
} d_partitions[NPARTTAB];

struct disktab *getdiskbyname();

SEE ALSO
    disktab(5)

ERRORS
This information should be obtained from the system for locally available disks (in particular, the disk partition tables).
NAME
getenv, setenv, unsetenv – manipulate environmental variables

SYNOPSIS
char *getenv(name)
char *name;
setenv(name, value, overwrite)
char *name, value;
int overwrite;
void unsetenv(name)
char *name;

DESCRIPTION
getenv searches the environment list (see environ(7)) for a string of the form name=value and returns a pointer to the string value if such a string is present, and 0 (NULL) if it is not.

setenv searches the environment list as getenv does; if the string name is not found, a string of the form name=value is added to the environment. If it is found, and overwrite is non-zero, its value is changed to value. setenv returns 0 on success and -1 on failure, where failure is caused by an inability to allocate space for the environment.

unsetenv removes all occurrences of the string name from the environment. There is no library provision for completely removing the current environment. It is suggested that the following code be used to do so.

static char *envinit[1];
extern char **environ;
environ = envinit;

All of these routines permit, but do not require, a trailing equals ("=") sign on name or a leading equals sign on value.

SEE ALSO
csh(1), sh(1), execve(2), environ(7)
NAME
  getenv – get value of environment variables

SYNOPSIS
  subroutine getenv (ename, evvalue)
  character(*) ename, evvalue

DESCRIPTION
  Getenv searches the environment list (see environ(7)) for a string of the form ename=value
  and returns value in evvalue if such a string is present, otherwise fills evvalue with blanks.

FILES
  /usr/lib/libU77.a

SEE ALSO
  environ(7), execve(2)
NAME
getfsent, getfsspec, getfsfile, getfstable, setfsent, endfsent – get file system descriptor file entry

SYNOPSIS
#include <fstab.h>
struct fstab *getfsent()
struct fstab *getfsspec(spec)
char *spec;
struct fstab *getfsfile(file)
char *file;
struct fstab *getfstable(type)
char *type;
int setfsent()
int endfsent()

DESCRIPTION
getfsent, getfsspec, getfstable, and getfsfile each return a pointer to an object with the following structure containing the broken-out fields of a line in the file system description file, <fstab.h>.

struct fstab {
    char *fs_spec;
    char *fs_file;
    char *fs_type;
    int fs_freq;
    int fs_passno;
};

The fields have meanings described in fstab(5).
getfsent reads the next line of the file, opening the file if necessary.
setfsent opens and rewinds the file.
endfsent closes the file.
getfsspec and getfsfile sequentially search from the beginning of the file until a matching special file name or file system file name is found, or until EOF is encountered. getfstable does likewise, matching on the file system type field.

FILES
/etc/fstab

SEE ALSO
fstab(5)

DIAGNOSTICS
Null pointer (0) returned on EOF or error.

ERRORS
All information is contained in a static area so it must be copied if it is to be saved.
NAME
getgrent, getgrgid, getgrnam, setgrent, endgrent – get group file entry

SYNOPSIS
#include <grp.h>

struct group *getgrent()

struct group *getgrgid(gid)
int gid;

struct group *getgrnam(name)
char *name;

setgrent()

endgrent()

DESCRIPTION
getgrent, getgrgid and getgrnam each return pointers to an object with the following structure
containing the broken-out fields of a line in the group file.

struct group {
  char *gr_name;
  char *gr_passwd;
  int gr_gid;
  char *gr_mem;
};

struct group *getgrent(), *getgrgid(), *getgrnam();

The members of this structure are:
gr_name    The name of the group.
gr_passwd  The encrypted password of the group.
gr_gid     The numerical group-ID.
gr_mem     Null-terminated vector of pointers to the individual member

getgrent simply reads the next line while getgrgid and getgrnam search until a matching gid or
name is found (or until EOF is encountered). Each routine picks up where the others leave
off so successive calls may be used to search the entire file.

A call to setgrent has the effect of rewinding the group file to allow repeated searches.
endgrent may be called to close the group file when processing is complete.

FILES
/etc/group
SEE ALSO
    getlogin(3), getpwnent(3), group(5)

DIAGNOSTICS
    A null pointer (0) is returned on EOF or error.

ERRORS
    All information is contained in a static area so it must be copied if it is to be saved.
NAME

gethostbyname, gethostbyaddr, gethostent, sethostent, endhostent – get network host entry

SYNOPSIS

#include <netdb.h>
extern int h_errno;
struct hostent *gethostbyname(name)
char *name;
struct hostent *gethostbyaddr(addr, len, type)
char *addr; int len, type;
struct hostent *gethostent()
sethostent(stayopen)
int stayopen;
endhostent()

DESCRIPTION

gethostbyname and gethostbyaddr each return a pointer to an object with the following structure. This structure contains either the information obtained from the name server, named(8), or broken-out fields from a line in /etc/hosts. If the local name server is not running these routines do a lookup in /etc/hosts.

struct hostent {
    char *h_name;    /* official name of host */
    char ***h_aliases;    /* alias list */
    int h_addrtype;    /* host address type */
    int h_length;    /* length of address */
    char ***h_addr_list;    /* list of addresses from name server */
};
#define h_addr h_addr_list[0] /* address, for backward compatibility */

The members of this structure are:

h_name Official name of the host.
h_aliases A zero terminated array of alternate names for the host.
h_addrtype The type of address being returned; currently always AF_INET.
h_length The length, in bytes, of the address.
h_addr_list A zero terminated array of network addresses for the host. Host addresses are returned in network byte order.
h_addr The first address in h_addr_list; this is for backward compatibility.

sethostent allows a request for the use of a connected socket using TCP for queries. If the stayopen flag is non-zero, this sets the option to send all queries to the name server using TCP and to retain the connection after each call to gethostbyname or gethostbyaddr.

endhostent closes the TCP connection.

DIAGNOSTICS

Error return status from gethostbyname and gethostbyaddr is indicated by return of a null pointer. The external integer h_errno may then be checked to see whether this is a temporary failure or an invalid or unknown host.

h_errno can have the following values:

    HOST_NOT_FOUND   No such host is known.
TRY_AGAIN

This is usually a temporary error and means that the local server did not receive a response from an authoritative server. A retry at some later time may succeed.

NO_RECOVERY

This is a non-recoverable error.

NO_ADDRESS

The requested name is valid but does not have an IP address; this is not a temporary error. This means another type of request to the name server will result in an answer.

FILES

/etc/hosts

SEE ALSO

hosts(5), resolver(3), named(8)

CAVEAT

gethostent is defined, and sethostent and endhostent are redefined, when libc is built to use only the routines to lookup in /etc/hosts and not the name server.

gethostent reads the next line of /etc/hosts, opening the file if necessary.

sethostent is redefined to open and rewind the file. If the stayopen argument is non-zero, the hosts data base will not be closed after each call to gethostbyname or gethostbyaddr. endhostent is redefined to close the file.

ERRORS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.
NAME
getlog – get user’s login name

SYNOPSIS
subroutine getlog (name)
characters(s) name

characters(s) function getlog()

DESCRIPTION
Getlog will return the user’s login name or all blanks if the process is running detached from a terminal.

FILES
/usr/lib/libU77.a

SEE ALSO
getlogin(3)
NAME
getlogin – get login name

SYNOPSIS
char *getlogin()

DESCRIPTION
getlogin returns a pointer to the login name as found in /etc/utmp. It may be used in conjunction with getpwnam to locate the correct password file entry when the same userid is shared by several login names.

If getlogin is called within a process that is not attached to a terminal, or if there is no entry in /etc/utmp for the process’s terminal, getlogin returns a NULL pointer (0). A reasonable procedure for determining the login name is to first call getlogin and if it fails, to call getpwuid(getuid()).

FILES
/etc/utmp

SEE ALSO
getpwnam(3), utmp(5), ttyslot(3)

DIAGNOSTICS
Returns a NULL pointer (0) if name not found.

ERRORS
The return values point to static data whose content is overwritten by each call.
NAME
getmntent, setmntent, addmntent, endmntent, hasmntoxt – get file system descriptor file entry

SYNOPSIS
#include <stdio.h>
#include <mntent.h>

FILE *setmntent(filep, type)
char *filep;
char *type;

struct mntent *getmntent(filep)
FILE *filep;

int addmntent(filep, mnt)
FILE *filep;
struct mntent *mnt;
char *hasmntoxt(mnt, opt)
struct mntent *mnt;
char *opt;

int endmntent(filep)
FILE *filep;

DESCRIPTION
These routines replace the getsent routines for accessing the file system description file /etc/fstab. They are also used to access the mounted file system description file /etc/mtab.

setmntent opens a file system description file and returns a file pointer which can then be used with getmntent, addmntent, or endmntent. The type argument is the same as in fopen(3). getmntent reads the next line from filep and returns a pointer to an object with the following structure containing the broken-out fields of a line in the filesystem description file, <mntent.h>. The fields have meanings described in fstab(5).

struct mntent {
    char *mnt_fsnam; /* file system name */
    char *mnt_dir; /* file system path prefix */
    char *mnt_type; /* 4.2, nfs, swap, or xx */
    char *mnt_opts; /* ro, quota, etc. */
    int mnt_freq; /* dump frequency, in days */
    int mnt_passno; /* pass number on parallel fsck */
};

addmntent adds the mntent structure mnt to the end of the open file filep. Note that filep has to be opened for writing if this is to work. hasmntoxt scans the mnt_opts field of the mntent structure mnt for a substring that matches opt. It returns the address of the substring if a match is found, 0 otherwise. Endmntent closes the file.

FILES
/etc/fstab
/etc/mtab

SEE ALSO
fstab(5), getsent(3)
DIAGNOSTICS
Null pointer (0) returned on EOF or error.

ERRORS
The returned mntent structure points to static information that is overwritten in each call.
NAME
getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent – get network entry

SYNOPSIS
#include <netdb.h>
struct netent *getnetent()

struct netent *getnetbyname(name)
char *name;

struct netent *getnetbyaddr(net, type)
long net;
int type;

setnetent(stayopen)
int stayopen;
endnetent()

DESCRIPTION
ginetent, getnetbyname, and getnetbyaddr each return a pointer to an object with the following structure containing the broken-out fields of a line in the network data base, /etc/networks.

struct netent {
    char *n_name; /* official name of net */
    char **n_aliases; /* alias list */
    int n_addrtype; /* net number type */
    unsigned long n_net; /* net number */
};

The members of this structure are:
n_name The official name of the network.
n_aliases A zero terminated list of alternate names for the network.
n_addrtype The type of the network number returned; currently only AF_INET.
n_net The network number. Network numbers are returned in machine byte order.

ginetent reads the next line of the file, opening the file if necessary.
setnetent opens and rewinds the file. If the stayopen flag is non-zero, the net data base will not be closed after each call to getnetbyname or getnetbyaddr.

Endnetent closes the file.

ginetbyname and getnetbyaddr sequentially search from the beginning of the file until a matching net name or net address and type is found, or until EOF is encountered. Network numbers are supplied in host order.

FILES
/etc/networks

SEE ALSO
networks(5)

DIAGNOSTICS
Null pointer (0) returned on EOF or error.
ERRORS

All information is contained in a static area so it must be copied if it is to be saved. Only Internet network numbers are currently understood. Expecting network numbers to fit in no more than 32 bits is probably naive.
NAME
getnetgrent, setnetgrent, endnetgrent, innetgr – get network group entry

SYNOPSIS
innetgr(netgroup, machine, user, domain)
char *netgroup, *machine, *user, *domain;
setnetgrent(netgroup)
char *netgroup
endnetgrent()
getnetgrent(machinep, userp, domainp)
char **machinep, **userp, **domainp;

DESCRIPTION
innetgr returns 1 or 0, depending on whether netgroup contains the machine, user, domain triple as a member. Any of the three strings machine, user, or domain can be NULL, in which case it signifies a wild card.

getnetgrent returns the next member of a network group. After the call, machinep will contain a pointer to a string containing the name of the machine part of the network group member, and similarly for userp and domainp. If any of machinep, userp or domainp is returned as a NULL pointer, it signifies a wild card. getnetgrent will malloc space for the name. This space is released when a endnetgrent call is made. getnetgrent returns 1 if it succeeding in obtaining another member of the network group, 0 if it has reached the end of the group.

setnetgrent establishes the network group from which getnetgrent will obtain members, and also restarts calls to getnetgrent from the beginning of the list. If the previous setnetgrent call was to a different network group, a endnetgrent call is implied. endnetgrent frees the space allocated during the getnetgrent calls.

FILES
/etc/netgroup
/etc/yp/domain/netgroup
/etc/yp/domain/netgroup.byuser
/etc/yp/domain/netgroup.byhost
NAME
getopt – get option letter from argv

SYNOPSIS
int getopt(int argc, char **argv, char *optstring);

DESCRIPTION
getopt returns the next option letter in argv that matches a letter in optstring. optstring is a
string of recognized option letters; if a letter is followed by a colon, the option is expected to
have an argument that may or may not be separated from it by white space. optarg is set to
point to the start of the option argument on return from getopt.

getopt places in optind the argv index of the next argument to be processed. Because optind is
external, it is normally initialized to zero automatically before the first call to getopt.

When all options have been processed (i.e., up to the first non-option argument), getopt
returns EOF. The special option -- may be used to delimit the end of the options; EOF will
be returned, and -- will be skipped.

DIAGNOSTICS
getopt prints an error message on stderr and returns a question mark (?) when it encounters an
option letter not included in optstring.

EXAMPLE
The following code fragment shows how one might process the arguments for a command that
can take the mutually exclusive options a and b, and the options f and o, both of which
require arguments:
main(argc, argv)
int argc;
char **argv;
{
    int c;
    extern int optind;
    extern char **optarg;

    while ((c = getopt(argc, argv, "abf:o:")) != EOF)
        switch (c) {
            case 'a':
                if (bflg)
                    errflg++;
                else
                    aflg++;
                break;
            case 'b':
                if (aflg)
                    errflg++;
                else
                    bproc();

MIPS Computer Systems, Inc. February 5, 1989 Page 1
break;
case 'f':
    ifile = optarg;
    break;
case 'o':
    ofile = optarg;
    break;
case '?':
default:
    errflg++;
    break;
}
if (errflg) {
    fprintf(stderr, "Usage: ...");
    exit(2);
}
for (; optind < argc; optind++) {
    ...
    ...
}

HISTORY
Written by Henry Spencer, working from a Bell Labs manual page. Modified by Keith Bostic to behave more like the System V version.

ERRORS
It is not obvious how "-" standing alone should be treated; this version treats it as a non-option argument, which is not always right.
Option arguments are allowed to begin with "-"; this is reasonable but reduces the amount of error checking possible.
getopt is quite flexible but the obvious price must be paid: there is much it could do that it doesn't, like checking mutually exclusive options, checking type of option arguments, etc.
NAME
getpass — read a password

SYNOPSIS

    char *getpass(prompt)
    char *prompt;

DESCRIPTION
getpass reads a password from the file /dev/tty, or if that cannot be opened, from the standard input, after prompting with the null-terminated string prompt and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters.

FILES
/dev/tty

SEE ALSO
crypt(3)

ERRORS
The return value points to static data whose content is overwritten by each call.
NAME
    getpid – get process id

SYNOPSIS
    integer function getpid()

DESCRIPTION
    getpid returns the process ID number of the current process.

FILES
    /usr/lib/libU77.a

SEE ALSO
    getpid(2)
NAME
getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent = get protocol entry

SYNOPSIS
#include <netdb.h>

struct protoent *getprotoent()
struct protoent *getprotobyname(name)
char *name;
struct protoent *getprotobynumber(proto)
int proto;
setprotoent(stayopen)
int stayopen
endprotoent()

DESCRIPTION
getprotoent, getprotobyname, and getprotobynumber each return a pointer to an object with the following structure containing the broken-out fields of a line in the network protocol data base, /etc/protocols.

struct
protoent {
  char *p_name;  /* official name of protocol */
  char **p_aliases;  /* alias list */
  int p_proto;  /* protocol number */
};

The members of this structure are:
p_name The official name of the protocol.
p_aliases A zero terminated list of alternate names for the protocol.
p_proto The protocol number.

getprotoent reads the next line of the file, opening the file if necessary.

setprotoent opens and rewinds the file. If the stayopen flag is non-zero, the net data base will not be closed after each call to getprotobyname or getprotobynumber.

endprotoent closes the file.

getprotobyname and getprotobynumber sequentially search from the beginning of the file until a matching protocol name or protocol number is found, or until EOF is encountered.

FILES
/etc/protocols

SEE ALSO
protocols(5)

DIAGNOSTICS
Null pointer (0) returned on EOF or error.

ERRORS
All information is contained in a static area so it must be copied if it is to be saved. Only the Internet protocols are currently understood.
NAME
getpw – get name from uid

SYNOPSIS
getpw(uid, buf)
char *buf;

DESCRIPTION
Getpw is made obsolete by getpwuid(3).
getpw searches the password file for the (numerical) uid, and fills in buf with the correspond-
ing line; it returns non-zero if uid could not be found. The line is null-terminated.

FILES
/etc/passwd

SEE ALSO
getpwent(3), passwd(5)

DIAGNOSTICS
Non-zero return on error.
NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent, setpwfile – get password file entry

SYNOPSIS

#include <pwd.h>

struct passwd *getpwuid(uid)
int uid;

struct passwd *getpwnam(name)
char *name;

struct passwd *getpwent()

setpwent()

endpwent()

setpwfile(name)
char *name;

DESCRIPTION

getpwent, getpwuid and getpwnam each return a pointer to an object with the following structure containing the broken-out fields of a line in the password file.

struct passwd {
    char *pw_name;
    char *pw_passwd;
    int pw_uid;
    int pw_gid;
    int pw_quota;
    char *pw_comment;
    char *pw_gecos;
    char *pw_dir;
    char *pw_shell;
};

struct passwd *getpwent(), *getpwuid(), *getpwnam();

The fields pw_quota and pw_comment are unused; the others have meanings described in passwd(5).

Searching of the password file is done using the ndbm database access routines. setpwent opens the database; endpwent closes it. getpwuid and getpwnam search the database (opening it if necessary) for a matching uid or name. E0F is returned if there is no entry.

For programs wishing to read the entire database, getpwent reads the next line (opening the database if necessary). In addition to opening the database, setpwent can be used to make .1 getpwent begin its search from the beginning of the database.
setpfile changes the default password file to name thus allowing alternate password files to be used. Note that it does not close the previous file. If this is desired, endpwent should be called prior to it.

FILES
/etc/passwd

SEE ALSO
getlogin(3), getgrent(3), passwd(5)

DIAGNOSTICS
The routines getpwent, getpwuid, and getpwnam, return a NULL pointer (0) on EOF or error.

ERRORS
All information is contained in a static area so it must be copied if it is to be saved.
NAME
getrpcnt, getrpcbyname, getrpcbyname number — get RPC entry

SYNOPSIS
#include <netdb.h>
struct rpcent *getrpcnt()
struct rpcent *getrpcbyname(name)
char *name;
struct rpcent *getrpcbyname number(number)
int number;
setrpcnt(stayopen)
int stayopen
endrpcnt()

DESCRIPTION
getrpcnt, getrpcbyname, and getrpcbyname number each return a pointer to an object with the following structure containing the broken-out fields of a line in the rpc program number database, /etc/rpc.

struct rpcent {
    char  *r_name;  /* name of server for this rpc program */
    char  **r_aliases;  /* alias list */
    long  r_number;  /* rpc program number */
};

The members of this structure are:

r_name  The name of the server for this rpc program.

r_aliases  A zero terminated list of alternate names for the rpc program.

r_number  The rpc program number for this service.

getrpcnt reads the next line of the file, opening the file if necessary.

setrpcnt opens and rewinds the file. If the stayopen flag is non-zero, the net data base will not be closed after each call to getrpcnt (either directly, or indirectly through one of the other "getrpc" calls).

eendrpcnt closes the file.

getrpcbyname and getrpcbyname number sequentially search from the beginning of the file until a matching rpc program name or program number is found, or until EOF is encountered.

FILES
/etc/rpc
/etc/yp/domainname/rpc.bynumber

SEE ALSO
rpc(5), rpcinfo(8), ypservices(8)

DIAGNOSTICS
Null pointer (0) returned on EOF or error.

ERRORS
All information is contained in a static area so it must be copied if it is to be saved.
NAME
getrpcport – get RPC port number

SYNOPSIS
int getrpcport(host, progrnum, versnum, proto)
    char *host;
    int progrnum, versnum, proto;

DESCRIPTION
getrpcport returns the port number for version versnum of the RPC program progrnum running on host and using protocol proto. It returns 0 if it cannot contact the portmapper, or if progrnum is not registered. If progrnum is registered but not with version versnum, it will return that port number.
NAME

gets, fgets – get a string from a stream

SYNOPSIS

#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
FILE *stream;

DESCRIPTION

gets reads a string into s from the standard input stream stdin. The string is terminated by a newline character, which is replaced in s by a null character. gets returns its argument.

fgets reads n-1 characters, or up through a newline character, whichever comes first, from the stream into the string s. The last character read into s is followed by a null character. fgets returns its first argument.

SEE ALSO

puts(3S), getc(3S), scanf(3S), fread(3S), ferror(3S)

DIAGNOSTICS

gets and fgets return the constant pointer NULL upon end of file or error.

ERRORS

gets deletes a newline, fgets keeps it, all in the name of backward compatibility.
NAME

getservent, getservbyname, getservbyport, setservent, endservent – get service entry

SYNOPSIS

#include <netdb.h>

struct servent *getservent()

struct servent *getservbyname(name, proto)
char *name, *proto;

struct servent *getservbyport(port, proto)
int port; char *proto;

setservent(stayopen)
int stayopen
endservent()

DESCRIPTION

getservent, getservbyname, and getservbyport each return a pointer to an object with the following structure containing the broken-out fields of a line in the network services data base, /etc/services.

struct servent {
    char *s_name; /* official name of service */
    char **s_aliases; /* alias list */
    int s_port; /* port service resides at */
    char *s_proto; /* protocol to use */
};

The members of this structure are:

s_name The official name of the service.

s_aliases A zero terminated list of alternate names for the service.

s_port The port number at which the service resides. Port numbers are returned in network byte order.

s_proto The name of the protocol to use when contacting the service.

getservent reads the next line of the file, opening the file if necessary.

setservent opens and rewinds the file. If the stayopen flag is non-zero, the net data base will not be closed after each call to getservbyname or .IR getservbyport.

endservent closes the file.

getservbyname and getservbyport sequentially search from the beginning of the file until a matching protocol name or port number is found, or until EOF is encountered. If a protocol name is also supplied (non-NULL), searches must also match the protocol.

FILES

/etc/services

SEE ALSO

getprotoent(3N), services(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

ERRORS

All information is contained in a static area so it must be copied if it is to be saved. Expecting port numbers to fit in a 32 bit quantity is probably naive.
NAME
gettyent, getttynam, settyent, endttyent — get ttys file entry

SYNOPSIS
#include <ttyent.h>
struct ttent *getttyent()
struct ttent *getttynam(name)
char *name;
settyent()
endttyent()

DESCRIPTION
gettyent, and getttynam each return a pointer to an object with the following structure containing
the broken-out fields of a line from the tty description file.

struct ttent {
  char *ty_name; /* terminal device name */
  char *ty_getty; /* command to execute, usually getty */
  char *ty_type; /* terminal type for termcap (3X) */
  int ty_status; /* status flags (see below for defines) */
  char *ty_window; /* command to start up window manager */
  char *ty_comment; /* usually the location of the terminal */
};

#define TTY_ON 0x1 /* enable logins (startup getty) */
#define TTY_SECURE 0x2 /* allow root to login */

extern struct ttent *getttyent();
extern struct ttent *getttynam();

ty_name is the name of the character-special file in the directory “/dev”. For various
reasons, it must reside in the directory “/dev”.

ty_getty is the command (usually getty(8)) which is invoked by init to initialize tty line
characteristics. In fact, any arbitrary command can be used; a typical use is to
initiate a terminal emulator in a window system.

ty_type is the name of the default terminal type connected to this tty line. This is typically
a name from the termcap(5) data base. The environment variable
‘TERM’ is initialized with this name by getty(8) or login(1).

ty_status is a mask of bit fields which indicate various actions to be allowed on this tty
line. The following is a description of each flag.

TTY_ON Enables logins (i.e., init(8) will start the specified “getty”
command on this entry).

TTY_SECURE Allows root to login on this terminal. Note that ‘TTY_ON’
must be included for this to be useful.

ty_window    is the command to execute for a window system associated with the line. The
            window system will be started before the command specified in the ty_getty
            entry is executed. If none is specified, this will be null.

ty_comment   is the trailing comment field, if any; a leading delimiter and white space will be
            removed.

gettyent reads the next line from the ttys file, opening the file if necessary; settyent rewinds
the file; endttyent closes it.

gettynam searches from the beginning of the file until a matching name is found (or until EOF
is encountered).

FILES

/etc/ttys

SEE ALSO

login(1), ttyslot(3), ttys(5), gettytab(5), termcap(5), getty(8), init(8)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

ERRORS

All information is contained in a static area so it must be copied if it is to be saved.
NAME
getusershell, setusershell, endusershell - get legal user shells

SYNOPSIS
  char *getusershell()
  setusershell()
  endusershell()

DESCRIPTION
getusershell returns a pointer to a legal user shell as defined by the system manager in the file
/etc/shells. If /etc/shells does not exist, the two standard system shells /bin/sh and /bin/csh are
returned.

getusershell reads the next line (opening the file if necessary); setusershell rewinds the file;
endusershell closes it.

FILES
/etc/shells

DIAGNOSTICS
The routine getusershell returns a null pointer (0) on EOF or error.

ERRORS
All information is contained in a static area so it must be copied if it is to be saved.
NAME
  getuid, getgid – get user or group ID of the caller

SYNOPSIS
  integer function getuid()

  integer function getgid()

DESCRIPTION
  These functions return the real user or group ID of the user of the process.

FILES
  /usr/lib/libU77.a

SEE ALSO
  getuid(2)
NAME
getwd – get current working directory pathname

SYNOPSIS
char *getwd(pathname)
char *pathname;

DESCRIPTION
getwd copies the absolute pathname of the current working directory to pathname and returns a pointer to the result.

LIMITATIONS
Maximum pathname length is MAXPATHLEN characters (1024), as defined in <sys/param.h>.

DIAGNOSTICS
getwd returns zero and places a message in pathname if an error occurs.
NAME
hypot, cabs – Euclidean distance, complex absolute value

SYNOPSIS
#include <math.h>

double hypot(x,y)
double x,y;

float hypot(float x, float y)
double x,y;

double cabs(z)
struct {double x,y;} z;

float fcabs(z)
struct {float x,y;} z;

DESCRIPTION
Hypot(x,y), fhypot(x,y), cabs(x,y) and fcabs(x,y) return \( \sqrt{x^2+y^2} \) computed in such a way that underflow will not happen, and overflow occurs only if the final result deserves it.

Fhypot and fcabs are the same functions as hypot and cabs but for the float data type.

\( \text{hypot}(\infty,v) = \text{hypot}(v,\infty) = +\infty \) for all \( v \), including \( NaN \).

DIAGNOSTICS
When the correct value would overflow, hypot returns +\( \infty \).

ERROR (due to Roundoff, etc.)
Below 0.97 \( ulps \). Consequently hypot(5.0,12.0) = 13.0 exactly; in general, hypot and cabs return an integer whenever an integer might be expected.

The same cannot be said for the shorter and faster version of hypot and cabs that is provided in the comments in cabs.c; its error can exceed 1.2 \( ulps \).

NOTES
As might be expected, hypot(\( v,NaN \)) and hypot(\( NaN,v \)) are \( NaN \) for all finite \( v \). Programmers might be surprised at first to discover that hypot(\( \pm\infty,NaN \)) = +\( \infty \). This is intentional; it happens because hypot(\( \infty,v \)) = +\( \infty \) for all \( v \), finite or infinite. Hence hypot(\( \infty,v \)) is independent of \( v \). The IEEE \( NaN \) is designed to disappear when it turns out to be irrelevant, as it does in hypot(\( \infty,NaN \)).

SEE ALSO
math(3M), sqrt(3M)

AUTHOR
W. Kahan
NAME
   idate, itime — return date or time in numerical form

SYNOPSIS
   subroutine idate (iarray)
   integer iarray(3)

   subroutine itime (iarray)
   integer iarray(3)

DESCRIPTION
   idate returns the current date in *iarray*. The order is: day, mon, year. Month will be in the range 1-12. Year will be ≥ 1969.

   itime returns the current time in *iarray*. The order is: hour, minute, second.

FILES
   /usr/lib/libU77.a

SEE ALSO
   ctime(3F), fdate(3F)
NAME

copysign, drem, finite, logb, scalb – copysign, remainder, exponent manipulations

SYNOPSIS

#include <math.h>

double copysign(x,y)
double x,y;

double drem(x,y)
double x,y;

int finite(x)
double x;

double logb(x)
double x;

double scalb(x,n)
double x,n;

int n;

DESCRIPTION

These functions are required for, or recommended by the IEEE standard 754 for floating-point arithmetic.

Copysign(x,y) returns x with its sign changed to y’s.

Drem(x,y) returns the remainder r := x – n*y where n is the integer nearest the exact value of x/y; moreover if \(|n - x/y| = 1/2\) then n is even. Consequently the remainder is computed exactly and \(|r| \leq |y|/2\). But drem(x,0) is exceptional; see below under DIAGNOSTICS.

Finite(x) = 1 just when \(-\infty < x < +\infty,

= 0 otherwise (when |x| = \infty or x is NaN)

Logb(x) returns x’s exponent n, a signed integer converted to double-precision floating-point and so chosen that \(1 \leq |x|/2^n < 2\) unless \(x = 0\) or \(|x| = \infty\) or x lies between 0 and the Underflow Threshold.

Scalb(x,n) = x*(2**n) computed, for integer n, without first computing \(2^n\).

DIAGNOSTICS

IEEE 754 defines drem(x,0) and drem(\infty,y) to be invalid operations that produce a NaN.

IEEE 754 defines logb(\pm\infty) = +\infty and logb(0) = -\infty, and requires the latter to signal Division-by-Zero.

SEE ALSO

floor(3M), fp_class(3), math(3M)

AUTHOR

Kwok-Choi Ng

BUGS

IEEE 754 currently specifies that logb(denormalized no.) = logb(tiniest normalized no. > 0) but the consensus has changed to the specification in the new proposed IEEE standard p854, namely that logb(x) satisfy

\[ 1 \leq \text{scalb}(\lfloor x \rfloor, -\logb(x)) < \text{Radix} \quad \ldots = 2 \text{ for IEEE 754} \]

for every x except 0, \(\infty\) and NaN. Almost every program that assumes 754’s specification will work correctly if logb follows 854’s specification instead.

IEEE 754 requires copysign(x,NaN) = ±x but says nothing else about the sign of a NaN.
NAME
inet_addr, inet_network, inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof – Internet address manipulation routines

SYNOPSIS
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

unsigned long inet_addr(cp)
char *cp;

unsigned long inet_network(cp)
char *cp;

char *inet_ntoa(in)
struct in_addr in;

struct in_addr inet_makeaddr(net, lna)
int net, lna;

int inet_lnaof(in)
struct in_addr in;

int inet_netof(in)
struct in_addr in;

DESCRIPTION
The routines inet_addr and inet_network each interpret character strings representing numbers expressed in the Internet standard “.” notation, returning numbers suitable for use as Internet addresses and Internet network numbers, respectively. The routine inet_ntoa takes an Internet address and returns an ASCII string representing the address in “.” notation. The routine inet_makeaddr takes an Internet network number and a local network address and constructs an Internet address from it. The routines inet_netof and inet_lnaof break apart Internet host addresses, returning the network number and local network address part, respectively.

All Internet address are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

INTERNET ADDRESSES
Values specified using the “.” notation take one of the following forms:
   a.b.c.d
   a.b.c
   a.b
   a

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on the VAX the bytes referred to above appear as “d.c.b.a”. That is, VAX bytes are ordered from right to left.

When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three part address format convenient for specifying Class B network addresses as “128.net.host”.

When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as “net.host”.

When only one part is given, the value is stored directly in the network address without any byte rearrangement.
All numbers supplied as "parts" in a "." notation may be decimal, octal, or hexadecimal, as specified in the C language (i.e., a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

SEE ALSO
gethostbyname(3N), getnetent(3N), hosts(5), networks(5),

DIAGNOSTICS
The value -1 is returned by inet_addr and inet_network for malformed requests.

ERRORS
The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed. The string returned by inet_ntoa resides in a static memory area. 

inet_addr should return a struct in_addr.

inet_netof does not understand subnets, and will return the network number incorrectly in a subnetted environment.
NAME
initgroups – initialize group access list

SYNOPSIS
initgroups(name, basegid)
char *name;
int basegid;

DESCRIPTION
initgroups reads through the group file and sets up, using the setgroups(2) call, the group
access list for the user specified in name. The basegid is automatically included in the groups
list. Typically this value is given as the group number from the password file.

FILES
/etc/group

SEE ALSO
setgroups(2)

DIAGNOSTICS
initgroups returns -1 if it was not invoked by the super-user.

ERRORS
initgroups uses the routines based on getgrent(3). If the invoking program uses any of these
routines, the group structure will be overwritten in the call to initgroups.
NAME
insque, remque – insert/remove element from a queue

SYNOPSIS
struct qelem {
    struct qelem *q_forw;
    struct qelem *q_back;
    char q_data[];
};
insque(elem, pred)
struct qelem *elem, *pred;
remque(elem)
struct qelem *elem;

DESCRIPTION
insque and remque manipulate queues built from doubly linked lists. Each element in the
queue must in the form of “struct qelem”. insque inserts elem in a queue immediately after
pred; remque removes an entry elem from a queue.

SEE ALSO
NAME
intro – introduction to C library functions

DESCRIPTION
This section describes functions that may be found in various libraries. The library functions
are those other than the functions which directly invoke UNIX system primitives, described in
section 2. Most of these functions are accessible from the C library, libc, which is automatically
loaded by the C compiler cc(1), and the Pascal compiler pc(1). The link editor ld(1)
searches this library under the ‘-lc’ option. The C library also includes all the functions
described in section 2.

A subset of these functions are available from Fortran; they are described separately in
intro(3F).

The functions described in this section are grouped into various sections:

(3) The straight "3" functions are the standard C library functions.
(3N) These functions constitute the internet network library.
(3S) These functions constitute the 'standard I/O package', see stdio(3S) for more details.
     Declarations for these functions may be obtained from the include file <stdio.h>.
(3C) These routines are included for compatibility with other systems. In particular, a
     number of system call interfaces provided in previous releases of 4BSD have been
     included for source code compatibility. Use of these routines should, for the most
     part, be avoided. The manual page entry for each compatibility routine indicates the
     proper interface to use.
(3M) These functions constitute the math library, libm. When functions in the math library
     (see math(3M)) are passed values that are undefined or would generate answers that
     are out of range, they return the values as defined by the IEEE 754-1985 standard for
     binary floating-point arithmetic. These are usually infinities or Nan's (not-a-number's).
     See the man page for the specific function and what it returns in these cases. The
     math library is loaded as needed by the Pascal compiler pc(1). C programs that wish
     to use this library need to specify the "-lm" option.
(3X) These functions constitute minor libraries and other miscellaneous run-time facilities.
     Most are available only when programming in C. These functions include libraries
     that provide device independent plotting functions, terminal independent screen
     management routines for two dimensional non-bitmap display terminals, and functions
     for managing data bases with inverted indexes. These functions are located in
     separate libraries indicated in each manual entry.

FILES
/usr/lib/libc.a the C library
/usr/lib/libm.a the math library

SEE ALSO
stdio(3S), math(3M), intro(2), cc(1), ld(1), nm(1)

LIST OF FUNCTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Appears on Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort</td>
<td>abort.3</td>
<td>generate a fault</td>
</tr>
<tr>
<td>abs</td>
<td>abs.3</td>
<td>integer absolute value</td>
</tr>
<tr>
<td>acos</td>
<td>sin.3m</td>
<td>inverse trigonometric function</td>
</tr>
<tr>
<td>acosh</td>
<td>asinh.5m</td>
<td>inverse hyperbolic function</td>
</tr>
<tr>
<td>alarm</td>
<td>alarm.3c</td>
<td>schedule signal after specified time</td>
</tr>
<tr>
<td>alloca</td>
<td>malloc.3</td>
<td>memory allocator</td>
</tr>
<tr>
<td>arc</td>
<td>plot.3x</td>
<td>graphics interface</td>
</tr>
</tbody>
</table>

MIPS Computer Systems, Inc. - February 5, 1989 Page 1
asctime
asin
asinh
assert
atan
atanh
atan2
atof
atoi
atol
bcmp
bcopy
bzero
cabs
calloc
cbrt
cell
circle
clearrerr
closedir
closelog
closepl
cnt
copysign
cos
cosh
crypt
ctime
curses
dbminit
delete
drem
ecvt
data
ecrypt
end
endfsent
endgrent
endhostent
endnetent
endprotoent
endwent
endservent
environ
erase
erf
erfc
etext
exec
exece
exect
execte
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execlp</td>
<td>execute a file</td>
</tr>
<tr>
<td>execl</td>
<td>execute a file</td>
</tr>
<tr>
<td>exec</td>
<td>execute a file</td>
</tr>
<tr>
<td>execv</td>
<td>execute a file</td>
</tr>
<tr>
<td>execvp</td>
<td>execute a file</td>
</tr>
<tr>
<td>exit</td>
<td>terminate a process after flushing any pending output</td>
</tr>
<tr>
<td>exp</td>
<td>exponential</td>
</tr>
<tr>
<td>expm1</td>
<td>exp(x)−1</td>
</tr>
<tr>
<td>fabs</td>
<td>absolute value</td>
</tr>
<tr>
<td>facos</td>
<td>inverse trigonometric function</td>
</tr>
<tr>
<td>fasin</td>
<td>inverse trigonometric function</td>
</tr>
<tr>
<td>fatan</td>
<td>inverse trigonometric function</td>
</tr>
<tr>
<td>fatan2</td>
<td>inverse trigonometric function</td>
</tr>
<tr>
<td>fcabs</td>
<td>complex absolute value</td>
</tr>
<tr>
<td>fceil</td>
<td>integer no less than</td>
</tr>
<tr>
<td>fcclose</td>
<td>close or flush a stream</td>
</tr>
<tr>
<td>fcos</td>
<td>trigonometric function</td>
</tr>
<tr>
<td>fcvt</td>
<td>output conversion</td>
</tr>
<tr>
<td>feof</td>
<td>stream status inquiries</td>
</tr>
<tr>
<td>ferror</td>
<td>stream status inquiries</td>
</tr>
<tr>
<td>fetch</td>
<td>data base subroutines</td>
</tr>
<tr>
<td>fexp</td>
<td>exponential</td>
</tr>
<tr>
<td>fexpm1</td>
<td>exp(x)−1</td>
</tr>
<tr>
<td>floor</td>
<td>integer no greater than</td>
</tr>
<tr>
<td>flush</td>
<td>close or flush a stream</td>
</tr>
<tr>
<td>ffs</td>
<td>bit and byte string operations</td>
</tr>
<tr>
<td>fgetc</td>
<td>get character or word from stream</td>
</tr>
<tr>
<td>fget</td>
<td>get a string from a stream</td>
</tr>
<tr>
<td>fhypot</td>
<td>Euclidean distance</td>
</tr>
<tr>
<td>fileno</td>
<td>stream status inquiries</td>
</tr>
<tr>
<td>finite</td>
<td>is floating-point value finite</td>
</tr>
<tr>
<td>firstkey</td>
<td>data base subroutines</td>
</tr>
<tr>
<td>flog</td>
<td>natural logarithm</td>
</tr>
<tr>
<td>flog1p</td>
<td>log(1+x)</td>
</tr>
<tr>
<td>floor</td>
<td>integer no greater than</td>
</tr>
<tr>
<td>fmod</td>
<td>floating-point remainder</td>
</tr>
<tr>
<td>fopen</td>
<td>open a stream</td>
</tr>
<tr>
<td>fprintf</td>
<td>formatted output conversion</td>
</tr>
<tr>
<td>fpgetc</td>
<td>put character or word on a stream</td>
</tr>
<tr>
<td>fpgets</td>
<td>put a string on a stream</td>
</tr>
<tr>
<td>fread</td>
<td>buffered binary input/output</td>
</tr>
<tr>
<td>free</td>
<td>memory allocator</td>
</tr>
<tr>
<td>frexp</td>
<td>split into mantissa and exponent</td>
</tr>
<tr>
<td>fscanf</td>
<td>formatted input conversion</td>
</tr>
<tr>
<td>fseek</td>
<td>reposition a stream</td>
</tr>
<tr>
<td>fsin</td>
<td>trigonometric function</td>
</tr>
<tr>
<td>fsinh</td>
<td>hyperbolic function</td>
</tr>
<tr>
<td>ftan</td>
<td>trigonometric function</td>
</tr>
<tr>
<td>ftanh</td>
<td>hyperbolic function</td>
</tr>
<tr>
<td>fsqrt</td>
<td>square root</td>
</tr>
<tr>
<td>ftell</td>
<td>reposition a stream</td>
</tr>
<tr>
<td>ftime</td>
<td>get date and time</td>
</tr>
<tr>
<td>ftrunc</td>
<td>floating-point truncation</td>
</tr>
</tbody>
</table>
fwrite
fread.3s
buffered binary input/output
output conversion
gcvt
ecvt.3
getc
getc.3s
getchar
getc.3s
getdiskbyname
getdisk.3x
getenv
getenv.3
getfsent
getfsent.3x
getfsfile
getfsent.3x
getfsynspec
getfsent.3x
getfstype
getfsent.3x
getgrent
getgrent.3
getgrgid
getgrent.3
getgrnam
getgrent.3
gethostbyaddr
gethostbyname.3n
get host entry
gethostbyname
gethostbyname.3n
get host entry
gethostent
gethostbyname.3n
get host entry
getlogin
getlogin.3
get login name
getnetbyaddr
getnetent.3n
get network entry
gethostbyname
gethostbyname.3n
get host entry
gethostent
gethostbyname.3n
get host entry
getnetent
getnetent.3n
get network entry
getnetpass
getpass.3
read a password
getprotobyname
getprotoent.3n
get protocol entry
getprotoent
getprotoent.3n
get protocol entry
getpw
getpw.3
get name from uid
getpwent
getpwent.3
get password file entry
getpwnam
getpwent.3
get password file entry
getpwuid
getpwent.3
get password file entry
gets
gets.3s
get a string from a stream
getservbyname
getservent.3n
get service entry
getservbyport
getservent.3n
get service entry
getservent
getservent.3n
get service entry
getw
getc.3s
get character or word from stream
getwd
getwd.3
get current working directory pathname
gmtime
cftime.3
convert date and time to ASCII
gtty
stty.3c
set and get terminal state (defunct)
htonl
byteorder.3n
convert values between host and network byte order
htons
byteorder.3n
convert values between host and network byte order
hypot
hypot.3m
Euclidean distance
index
string.3
string operations
inet_addr
inet.3n
Internet address manipulation routines
inet_lnaof
inet.3n
Internet address manipulation routines
inet_makeaddr
inet.3n
Internet address manipulation routines
inet_netof
inet.3n
Internet address manipulation routines
inet_network
inet.3n
Internet address manipulation routines
initgroups
initgroups.3x
initialize group access list
initstate
random.3
better random number generator
insque
insque.3
insert/remove element from a queue
isalnum
cctype.3
character classification macros
isalpha
cctype.3
character classification macros
isascii
cctype.3
character classification macros
isatty
ttname.3
find name of a terminal
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iscntrl</td>
<td>character classification macros</td>
</tr>
<tr>
<td>isdigit</td>
<td>character classification macros</td>
</tr>
<tr>
<td>islower</td>
<td>character classification macros</td>
</tr>
<tr>
<td>ispunct</td>
<td>character classification macros</td>
</tr>
<tr>
<td>isspace</td>
<td>character classification macros</td>
</tr>
<tr>
<td>isupper</td>
<td>character classification macros</td>
</tr>
<tr>
<td>j0</td>
<td>bessel function</td>
</tr>
<tr>
<td>j1</td>
<td>bessel function</td>
</tr>
<tr>
<td>jn</td>
<td>bessel function</td>
</tr>
<tr>
<td>label</td>
<td>graphics interface</td>
</tr>
<tr>
<td>ldexp</td>
<td>split into mantissa and exponent</td>
</tr>
<tr>
<td>lgamma</td>
<td>log gamma function; (formerly gamma.3m)</td>
</tr>
<tr>
<td>lib2648</td>
<td>subroutines for the HP 2648 graphics terminal</td>
</tr>
<tr>
<td>line</td>
<td>graphics interface</td>
</tr>
<tr>
<td>linemode</td>
<td>graphics interface</td>
</tr>
<tr>
<td>localtime</td>
<td>convert date and time to ASCII</td>
</tr>
<tr>
<td>log</td>
<td>natural logarithm</td>
</tr>
<tr>
<td>logb</td>
<td>exponent extraction</td>
</tr>
<tr>
<td>log10</td>
<td>logarithm to base 10</td>
</tr>
<tr>
<td>log1p</td>
<td>log(1+x)</td>
</tr>
<tr>
<td>longjmp</td>
<td>non-local goto</td>
</tr>
<tr>
<td>malloc</td>
<td>memory allocator</td>
</tr>
<tr>
<td>mktemp</td>
<td>make a unique file name</td>
</tr>
<tr>
<td>modf</td>
<td>split into mantissa and exponent</td>
</tr>
<tr>
<td>moncontrol</td>
<td>prepare execution profile</td>
</tr>
<tr>
<td>monitor</td>
<td>prepare execution profile</td>
</tr>
<tr>
<td>monstartup</td>
<td>prepare execution profile</td>
</tr>
<tr>
<td>move</td>
<td>graphics interface</td>
</tr>
<tr>
<td>nextkey</td>
<td>data base subroutines</td>
</tr>
<tr>
<td>nice</td>
<td>set program priority</td>
</tr>
<tr>
<td>nlist</td>
<td>get entries from name list</td>
</tr>
<tr>
<td>ntohl</td>
<td>convert values between host and network byte order</td>
</tr>
<tr>
<td>ntohs</td>
<td>convert values between host and network byte order</td>
</tr>
<tr>
<td>opendir</td>
<td>directory operations</td>
</tr>
<tr>
<td>openlog</td>
<td>control system log</td>
</tr>
<tr>
<td>openpl</td>
<td>graphics interface</td>
</tr>
<tr>
<td>pause</td>
<td>stop until signal</td>
</tr>
<tr>
<td>pclose</td>
<td>initiate I/O to/from a process</td>
</tr>
<tr>
<td>perror</td>
<td>system error messages</td>
</tr>
<tr>
<td>point</td>
<td>graphics interface</td>
</tr>
<tr>
<td>popen</td>
<td>initiate I/O to/from a process</td>
</tr>
<tr>
<td>pow</td>
<td>exponential x**y</td>
</tr>
<tr>
<td>printf</td>
<td>formatted output conversion</td>
</tr>
<tr>
<td>psignal</td>
<td>system signal messages</td>
</tr>
<tr>
<td>putc</td>
<td>put character or word on a stream</td>
</tr>
<tr>
<td>putchar</td>
<td>put character or word on a stream</td>
</tr>
<tr>
<td>puts</td>
<td>put a string on a stream</td>
</tr>
<tr>
<td>putw</td>
<td>put character or word on a stream</td>
</tr>
<tr>
<td>qsort</td>
<td>quicker sort</td>
</tr>
<tr>
<td>rand</td>
<td>random number generator</td>
</tr>
<tr>
<td>random</td>
<td>better random number generator</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>rcmd</td>
<td>routines for returning a stream to a remote command</td>
</tr>
<tr>
<td>re_comp</td>
<td>regular expression handler</td>
</tr>
<tr>
<td>re_exec</td>
<td>regular expression handler</td>
</tr>
<tr>
<td>readdir</td>
<td>directory operations</td>
</tr>
<tr>
<td>realloc</td>
<td>memory allocator</td>
</tr>
<tr>
<td>remque</td>
<td>insert/remove element from a queue</td>
</tr>
<tr>
<td>rewind</td>
<td>reposition a stream</td>
</tr>
<tr>
<td>rewinddir</td>
<td>directory operations</td>
</tr>
<tr>
<td>rexec</td>
<td>return stream to a remote command</td>
</tr>
<tr>
<td>rindex</td>
<td>string operations</td>
</tr>
<tr>
<td>rint</td>
<td>round to nearest integer</td>
</tr>
<tr>
<td>rresvport</td>
<td>routines for returning a stream to a remote command</td>
</tr>
<tr>
<td>ruserok</td>
<td>routines for returning a stream to a remote command</td>
</tr>
<tr>
<td>scalb</td>
<td>exponent adjustment</td>
</tr>
<tr>
<td>scandir</td>
<td>scan a directory</td>
</tr>
<tr>
<td>scanf</td>
<td>formatted input conversion</td>
</tr>
<tr>
<td>seekdir</td>
<td>directory operations</td>
</tr>
<tr>
<td>setbuf</td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>setbuffer</td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>setegid</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>seteuid</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>setsent</td>
<td>get file system descriptor file entry</td>
</tr>
<tr>
<td>setgid</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>setgrent</td>
<td>get group file entry</td>
</tr>
<tr>
<td>sethostent</td>
<td>get network host entry</td>
</tr>
<tr>
<td>setjmp</td>
<td>non-local goto</td>
</tr>
<tr>
<td>setkey</td>
<td>DES encryption</td>
</tr>
<tr>
<td>setlinebuf</td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>setnetent</td>
<td>get network entry</td>
</tr>
<tr>
<td>setprotoent</td>
<td>get protocol entry</td>
</tr>
<tr>
<td>setpwent</td>
<td>get password file entry</td>
</tr>
<tr>
<td>setrgid</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>setruid</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>setservent</td>
<td>get service entry</td>
</tr>
<tr>
<td>setstate</td>
<td>better random number generator</td>
</tr>
<tr>
<td>setuid</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>signal</td>
<td>simplified software signal facilities</td>
</tr>
<tr>
<td>sin</td>
<td>trigonometric function</td>
</tr>
<tr>
<td>sinh</td>
<td>hyperbolic function</td>
</tr>
<tr>
<td>sleep</td>
<td>suspend execution for interval</td>
</tr>
<tr>
<td>space</td>
<td>graphics interface</td>
</tr>
<tr>
<td>sprintf</td>
<td>formatted output conversion</td>
</tr>
<tr>
<td>sqrt</td>
<td>square root</td>
</tr>
<tr>
<td>srand</td>
<td>random number generator</td>
</tr>
<tr>
<td>srandom</td>
<td>better random number generator</td>
</tr>
<tr>
<td>sscanf</td>
<td>formatted input conversion</td>
</tr>
<tr>
<td>stdio</td>
<td>standard buffered input/output package</td>
</tr>
<tr>
<td>store</td>
<td>data base subroutines</td>
</tr>
<tr>
<td>strcat</td>
<td>string operations</td>
</tr>
<tr>
<td>strcmp</td>
<td>string operations</td>
</tr>
<tr>
<td>strcpy</td>
<td>string operations</td>
</tr>
<tr>
<td>strlen</td>
<td>string operations</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>strcat</td>
<td>string operations</td>
</tr>
<tr>
<td>strncmp</td>
<td>string operations</td>
</tr>
<tr>
<td>strncpy</td>
<td>string operations</td>
</tr>
<tr>
<td>stty</td>
<td>set and get terminal state (defunct)</td>
</tr>
<tr>
<td>swab</td>
<td>swap bytes</td>
</tr>
<tr>
<td>sys_errlist</td>
<td>system error messages</td>
</tr>
<tr>
<td>sys_nerr</td>
<td>system error messages</td>
</tr>
<tr>
<td>sys_siglist</td>
<td>system signal messages</td>
</tr>
<tr>
<td>syslog</td>
<td>control system log</td>
</tr>
<tr>
<td>system</td>
<td>issue a shell command</td>
</tr>
<tr>
<td>tan</td>
<td>trigonometric function</td>
</tr>
<tr>
<td>tanh</td>
<td>hyperbolic function</td>
</tr>
<tr>
<td>telldir</td>
<td>directory operations</td>
</tr>
<tr>
<td>tgetent</td>
<td>terminal independent operation routines</td>
</tr>
<tr>
<td>tgetflag</td>
<td>terminal independent operation routines</td>
</tr>
<tr>
<td>tgetnum</td>
<td>terminal independent operation routines</td>
</tr>
<tr>
<td>tgetstr</td>
<td>terminal independent operation routines</td>
</tr>
<tr>
<td>tgoto</td>
<td>terminal independent operation routines</td>
</tr>
<tr>
<td>time</td>
<td>get date and time</td>
</tr>
<tr>
<td>times</td>
<td>get process times</td>
</tr>
<tr>
<td>timezone</td>
<td>convert date and time to ASCII</td>
</tr>
<tr>
<td>tputs</td>
<td>terminal independent operation routines</td>
</tr>
<tr>
<td>trunc</td>
<td>floating-point truncation</td>
</tr>
<tr>
<td>ttyname</td>
<td>find name of a terminal</td>
</tr>
<tr>
<td>ttyslot</td>
<td>find name of a terminal</td>
</tr>
<tr>
<td>ungetc</td>
<td>push character back into input stream</td>
</tr>
<tr>
<td>utime</td>
<td>set file times</td>
</tr>
<tr>
<td>malloc</td>
<td>aligned memory allocator</td>
</tr>
<tr>
<td>varargs</td>
<td>variable argument list</td>
</tr>
<tr>
<td>vlimit</td>
<td>control maximum system resource consumption</td>
</tr>
<tr>
<td>vtimes</td>
<td>get information about resource utilization</td>
</tr>
<tr>
<td>y0</td>
<td>bessel function</td>
</tr>
<tr>
<td>y1</td>
<td>bessel function</td>
</tr>
<tr>
<td>yn</td>
<td>bessel function</td>
</tr>
</tbody>
</table>
NAME
intro – introduction to RPC service library functions

DESCRIPTION
These functions constitute the RPC service library, *librpcsvc*. In order to get the link editor to
load this library, use the `-l rpcsvc` option of *cc*. Declarations for these functions may be
obtained from various include files `<rpcsvc/*,h>`.

LIST OF FUNCTIONS

<table>
<thead>
<tr>
<th>routine</th>
<th>on page</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ether</td>
<td>ether(3R)</td>
<td>monitor traffic on the Ethernet</td>
</tr>
<tr>
<td>getrpcport</td>
<td>getrpcport(3R)</td>
<td>get RPC port number</td>
</tr>
<tr>
<td>havedisk</td>
<td>rstat(3R)</td>
<td>determine if remote machine has disk</td>
</tr>
<tr>
<td>rex</td>
<td>rex(3r)</td>
<td>remote execution protocol</td>
</tr>
<tr>
<td>rusers</td>
<td>rusers(3R)</td>
<td>return number of users on remote machine</td>
</tr>
<tr>
<td>rquota</td>
<td>rquota(3R)</td>
<td>implement quotas on remote machines</td>
</tr>
<tr>
<td>rstat</td>
<td>rstat(3R)</td>
<td>get performance data from remote kernel</td>
</tr>
<tr>
<td>rusers</td>
<td>rusers(3R)</td>
<td>return information about users on remote machine</td>
</tr>
<tr>
<td>rwall</td>
<td>rwall(3R)</td>
<td>write to specified remote machines</td>
</tr>
<tr>
<td>spray</td>
<td>spray(3R)</td>
<td>scatter data in order to check the network</td>
</tr>
<tr>
<td>yppassword</td>
<td>yppassword(3R)</td>
<td>update user password in yellow pages</td>
</tr>
</tbody>
</table>
NAME
intro – introduction to FORTRAN library functions

DESCRIPTION
This section describes functions that are in the Fortran runtime library.

The math intrinsics required by the 1977 Fortran standard are available, although not described here. In addition, the abs, sqrt, exp, log, sin, and cos intrinsics have been extended for double complex values. They can be referenced using the generic names listed above, or they can be referenced using their specific names that consist of the generic names preceded by either cd or z. For example, if zz is double complex, then sqrt(zz), zsqrt(zz), or csqrt(zz) compute the square root of zz. The dcomplex intrinsic forms a double complex value from two double precision variables or expressions, and the name of the specific function for the conjugate of a double complex value is aconij.

Most of these functions are in libU77.a. Some are in libF77.a or libU77.a.

For efficiency, the SCCS ID strings are not normally included in the a.out file. To include them, simply declare

external f77lid

in any f77 module.

LIST OF FUNCTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Appears on Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort</td>
<td>abort.3f</td>
<td>abnormal termination</td>
</tr>
<tr>
<td>access</td>
<td>access.3f</td>
<td>determine accessibility of a file</td>
</tr>
<tr>
<td>alarm</td>
<td>alarm.3f</td>
<td>execute a subroutine after a specified time</td>
</tr>
<tr>
<td>chdir</td>
<td>chdir.3f</td>
<td>change default directory</td>
</tr>
<tr>
<td>chmod</td>
<td>chmod.3f</td>
<td>change mode of a file</td>
</tr>
<tr>
<td>ctime</td>
<td>time.3f</td>
<td>return system time</td>
</tr>
<tr>
<td>dtime</td>
<td>etime.3f</td>
<td>return elapsed execution time</td>
</tr>
<tr>
<td>etime</td>
<td>etime.3f</td>
<td>return elapsed execution time</td>
</tr>
<tr>
<td>fdate</td>
<td>fdate.3f</td>
<td>return date and time in an ASCII string</td>
</tr>
<tr>
<td>fgenc</td>
<td>getc.3f</td>
<td>get a character from a logical unit</td>
</tr>
<tr>
<td>flush</td>
<td>flush.3f</td>
<td>flush output to a logical unit</td>
</tr>
<tr>
<td>fork</td>
<td>fork.3f</td>
<td>create a copy of this process</td>
</tr>
<tr>
<td>fprintf</td>
<td>putc.3f</td>
<td>write a character to a fortran logical unit</td>
</tr>
<tr>
<td>fseek</td>
<td>fseek.3f</td>
<td>reposition a file on a logical unit</td>
</tr>
<tr>
<td>fstat</td>
<td>stat.3f</td>
<td>get file status</td>
</tr>
<tr>
<td>ftell</td>
<td>fseek.3f</td>
<td>reposition a file on a logical unit</td>
</tr>
<tr>
<td>gerror</td>
<td>perror.3f</td>
<td>get system error messages</td>
</tr>
<tr>
<td>getarg</td>
<td>getarg.3f</td>
<td>return command line arguments</td>
</tr>
<tr>
<td>getc</td>
<td>getc.3f</td>
<td>get a character from a logical unit</td>
</tr>
<tr>
<td>getcwd</td>
<td>getcwd.3f</td>
<td>get pathname of current working directory</td>
</tr>
<tr>
<td>getenv</td>
<td>getenv.3f</td>
<td>get value of environment variables</td>
</tr>
<tr>
<td>getgid</td>
<td>getuid.3f</td>
<td>get user or group ID of the caller</td>
</tr>
<tr>
<td>getlog</td>
<td>getlog.3f</td>
<td>get user's login name</td>
</tr>
<tr>
<td>getpid</td>
<td>getpid.3f</td>
<td>get process id</td>
</tr>
<tr>
<td>getuid</td>
<td>getuid.3f</td>
<td>get user or group ID of the caller</td>
</tr>
<tr>
<td>gmtime</td>
<td>time.3f</td>
<td>return system time</td>
</tr>
<tr>
<td>largc</td>
<td>getarg.3f</td>
<td>return command line arguments</td>
</tr>
<tr>
<td>idate</td>
<td>idate.3f</td>
<td>return date or time in numerical form</td>
</tr>
<tr>
<td>ierrno</td>
<td>perror.3f</td>
<td>get system error messages</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>irand</td>
<td>return random values</td>
<td></td>
</tr>
<tr>
<td>isatty</td>
<td>find name of a terminal port</td>
<td></td>
</tr>
<tr>
<td>itime</td>
<td>return date or time in numerical form</td>
<td></td>
</tr>
<tr>
<td>kill</td>
<td>send a signal to a process</td>
<td></td>
</tr>
<tr>
<td>len</td>
<td>tell about character objects</td>
<td></td>
</tr>
<tr>
<td>link</td>
<td>make a link to an existing file</td>
<td></td>
</tr>
<tr>
<td>loc</td>
<td>return the address of an object</td>
<td></td>
</tr>
<tr>
<td>ltime</td>
<td>return system time</td>
<td></td>
</tr>
<tr>
<td>perror</td>
<td>get system error messages</td>
<td></td>
</tr>
<tr>
<td>putc</td>
<td>write a character to a fortran logical unit</td>
<td></td>
</tr>
<tr>
<td>qsort</td>
<td>quick sort</td>
<td></td>
</tr>
<tr>
<td>rand</td>
<td>return random values</td>
<td></td>
</tr>
<tr>
<td>signal</td>
<td>change the action for a signal</td>
<td></td>
</tr>
<tr>
<td>sleep</td>
<td>suspend execution for an interval</td>
<td></td>
</tr>
<tr>
<td>stat</td>
<td>get file status</td>
<td></td>
</tr>
<tr>
<td>system</td>
<td>execute a UNIX command</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>return system time</td>
<td></td>
</tr>
<tr>
<td>ttynam</td>
<td>find name of a terminal port</td>
<td></td>
</tr>
<tr>
<td>unlink</td>
<td>remove a directory entry</td>
<td></td>
</tr>
<tr>
<td>wait</td>
<td>wait for a process to terminate</td>
<td></td>
</tr>
</tbody>
</table>
NAME
j0, j1, jn, y0, y1, yn – bessel functions

SYNOPSIS
#include <math.h>
double j0(x)
double x;
double j1(x)
double x;
double jn(n, x)
int n;
double x;
double y0(x)
double x;
double y1(x)
double x;
double yn(n, x)
int n;
double x;

DESCRIPTION
j0 and j1 return Bessel functions of x of the first kind of orders 0 and 1 respectively. jn returns the Bessel function of x of the first kind of order n.

y0 and y1 return Bessel functions of x of the second kind of orders 0 and 1 respectively. yn returns the Bessel function of x of the second kind of order n. The value of x must be positive.

DIAGNOSTICS
Non-positive arguments cause y0, y1 and yn to return a quiet NaN.

BUGS
Arguments too large in magnitude cause j0, j1, y0 and y1 to return zero with no indication of the total loss of precision.

SEE ALSO
math(3M)
NAME
   kill – send a signal to a process

SYNOPSIS
   function kill (pid, signum)
   integer pid, signum

DESCRIPTION
   Pid must be the process id of one of the user’s processes. Signum must be a valid signal
   number (see sigvec(2)). The returned value will be 0 if successful; an error code otherwise.

FILES
   /usr/lib/libU77.a

SEE ALSO
   kill(2), sigvec(2), signal(3F), fork(3F), perror(3F)
NAME
ldahread – read the archive header of a member of an archive file

SYNOPSIS
#include <stdio.h>
#include <ar.h>
#include <filehdr.h>
#include <sysms.h>
#include <ldfcn.h>

int ldahread (ldptr, arhead)
LDFILE *ldptr;
ARCHDR *arhead;

DESCRIPTION
If TYPE(ldptr) is the archive file magic number, ldahread reads the archive header of the common object file currently associated with ldptr into the area of memory beginning at arhead.

Ldahread returns SUCCESS or FAILURE. If TYPE(ldptr) does not represent an archive file or if it cannot read the archive header, Ldahread fails.

The program must be loaded with the object file access routine library libmld.a.

SEE ALSO
ldclose(3X), ldopen(3X), ar(4), ldfcn(4), and intro(4).
NAME
   ldclose, ldaclose - close a common object file

SYNOPSIS
   #include <stdio.h>
   #include <filehdr.h>
   #include <syms.h>
   #include <ldfcn.h>

   int ldclose (ldptr)
   LDFILE *ldptr;

   int ldaclose (ldptr)
   LDFILE *ldptr;

DESCRIPTION
   Ldopen(3X) and ldclose provide uniform access to simple object files and object files that are members of archive files. An archive of common object files can be processed as if it is a series of simple common object files.

   If TYPE(ldptr) does not represent an archive file, ldclose closes the file and frees the memory allocated to the LDFILE structure associated with ldptr. If TYPE(ldptr) is the magic number for an archive file and if archive has more files, ldclose reinitializes OFFSET(ldptr) to the file address of the next archive member and returns FAILURE. The LDFILE structure is prepared for a later ldopen(3X). In all other cases, ldclose returns SUCCESS.

   Ldaclose closes the file and frees the memory allocated to the LDFILE structure associated with ldptr regardless of the value of TYPE(ldptr). Ldaclose always returns SUCCESS. The function is often used with ldaopen.

   The program must be loaded with the object file access routine library libmlld.a.

SEE ALSO
   fclose(3S), ldopen(3X), ldfcn(4).
NAME
ldfthread – read the file header of a common object file

SYNOPSIS
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldfthread (ldptr, filehead)
LDFILE *ldptr;
FILHDR *filehead;

DESCRIPTION
Ldfthread reads the file header of the common object file currently associated with ldptr. It reads the file header into the area of memory beginning at filehead.

Ldfthread returns SUCCESS or FAILURE. If ldfthread cannot read the file header, it fails.

Usually, ldfthread can be avoided by using the macro HEADER(ldptr) defined in <ldfcn.h> (see ldfcn(4)). Note that the information in HEADER is swapped, if necessary. The information in any field, filename, of the file header can be accessed using HEADER(ldptr).filename.

The program must be loaded with the object file access routine library libmld.a.

SEE ALSO
ldclose(3X), ldopen(3X), ldfcn(4).
NAME

ldgetaux – retrieve an auxiliary entry, given an index

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <sym.h>
#include <ldfcn.h>

pAUXU ldgetaux (ldptr, iaux)
LDFILE ldptr;
long iaux;

DESCRIPTION

Ldgetaux returns a pointer to an auxiliary table entry associated with iaux. The AUXU is contained in a static buffer. Because the buffer can be overwritten by later calls to ldgetaux, it must be copied by the caller if the aux is to be saved or changed.

Note that auxiliary entries are not swapped as this routine cannot detect what manifestation of the AUXU union is retrieved. If LDAUXXSWAP(ldptr, ldf) is non-zero, a further call to swap_aux is required. Before calling the swap_aux routine, the caller should copy the aux.

If the auxiliary cannot be retrieved, Ldgetaux returns NULL (defined in <stdio.h>) for an object file. This occurs when:

- the auxiliary table cannot be found
- the iaux offset into the auxiliary table is beyond the end of the table

Typically, ldgetaux is called immediately after a successful call to ldthread to retrieve the data type information associated with the symbol table entry filled by ldthread. The index field of the symbol, PSYM, is the iaux when data type information is required. If the data type information for a symbol is not present, the index field is indexNil and ldgetaux should not be called.

The program must be loaded with the object file access routine library libmld.a.

SEE ALSO

ldclose(3X), ldapen(3X), ldthseek(3X), ldthread(3X), ldfcn(4).
NAME
ldgetname — retrieve symbol name for object file symbol table entry

SYNOPSIS
#include <stdio.h>
#include <filehdr.h>
#include <sym.h>
#include <ldfcn.h>

char *ldgetname (ldptr, symbol)
LDFILE * ldptr ;
pSYMR * symbol ;

DESCRIPTION
ldgetname returns a pointer to the name associated with symbol as a string. The string is contained in a static buffer. Because the buffer can be overwritten by later calls to ldgetname, the caller must copy the buffer if the name is to be saved.

If the name cannot be retrieved, ldgetname returns NULL (defined in <stdio.h>) for an object file. This occurs when:

- the string table cannot be found
- the name’s offset into the string table is beyond the end of the string table

Typically, ldgetname is called immediately after a successful call to lditbread. Ldgetname retrieves the name associated with the symbol table entry filled by lditbread.

The program must be loaded with the object file access routine library libmld.a.

SEE ALSO
ldclose(3X), ldopen(3X), ldttbseek(3X), ldttbread(3X), ldffc(4).
NAME

ldgetpd – retrieve procedure descriptor given a procedure descriptor index

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <sym.h>
#include <ldfcn.h>

long ldgetpd (ldptr, ipd, ppd)
LDFILE ldptr;
long ipd;
pPDR ipd;

DESCRIPTION

_Ldgetpd_ returns a SUCCESS or FAILURE depending on whether the procedure descriptor with index _ipd_ can be accessed. If it can be accessed, the structure pointed to by _ppd_ is filled with the contents of the corresponding procedure descriptor. The _isym_, _iline_, and _iopt_ fields of the procedure descriptor are updated to be used in further LD routine calls. The _adr_ field is updated from the symbol referenced by the _isym field_.

The PDR cannot be retrieved when:

- The procedure descriptor table cannot be found.
- The ipd offset into the procedure descriptor table is beyond the end of the table.
- The file descriptor that the ipd offset falls into cannot be found.

Typically, _ldgetpd_ is called while traversing the table that runs from 0 to _SYMHEADER(ldptr).ipdMax - 1_.

The program must be loaded with the object file access routine library _libmld.a_.

SEE ALSO

ldclose(3X), ldopen(3X), ldthseek(3X), ldthbread(3X), ldfcn(4).
NAME

ldread, ldlinit, ldlinem — manipulate line number entries of a common object file function

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldread (ldptr, fcnindx, linenum, linent)
LDFILE *ldptr;
long fcnindx;
unsigned short linenum;
LINER linent;

int ldlinit (ldptr, fcnindx)
LDFILE *ldptr;
long fcnindx;

int ldlinem (ldptr, linenum, linent)
LDFILE *ldptr;
unsigned short linenum;
LINER linent;

DESCRIPTION

Ldread searches the line number entries of the common object file currently associated with ldptr. Ldread begins its search with the line number entry for the beginning of a function and confines its search to the line numbers associated with a single function. The function is identified by fcnindx, which is the index of its local symbols entry in the object file symbol table. Ldread reads the entry with the smallest line number equal to or greater than linenum into linent.

Ldlinit and ldlinem together do exactly the same function as ldread. After an initial call to ldread or ldlinit, ldlinem can be used to retrieve a series of line number entries associated with a single function. Ldlinit simply finds the line number entries for the function identified by fcnindx. Ldlinem finds and reads the entry with the smallest line number equal to or greater than linenum into linent.

Ldread, ldlinit, and ldlinem each return either SUCCESS or FAILURE. If no line number entries exist in the object file, if fcnindx does not index a function entry in the symbol table, or if it finds no line number equal to or greater than linenum, ldread fails. If no line number entries exist in the object file or if fcnindx does not index a function entry in the symbol table, ldlinit fails. If it finds no line number equal to or greater than linenum, ldlinem fails.

The programs must be loaded with the object file access routine library libmld.a.

SEE ALSO

ldclose(3X), ldopen(3X), ldtbindex(3X), ldfcn(4).
NAME
ldseek, ldnseek – seek to line number entries of a section of a common object file

SYNOPSIS
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;

DESCRIPTION
Ldseek seeks to the line number entries of the section specified by sectindx of the common object file currently associated with ldptr.

Ldnseek seeks to the line number entries of the section specified by sectname.

Ldseek and ldnseek return SUCCESS or FAILURE. NOTE: Line numbers are not associated with sections in the MIPS symbol table; therefore, the second argument is ignored, but maintained for historical purposes.

If they cannot seek to the specified line number entries, both routines fail.

The program must be loaded with the object file access routine library libmlfd.a.

SEE ALSO
lclose(3X), lopen(3X), ldshread(3X), ldfcn(4).
NAME
ldohseek – seek to the optional file header of a common object file

SYNOPSIS
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldohseek (ldptr)
LDFILE *ldptr;

DESCRIPTION
Ldohseek seeks to the optional file header of the common object file currently associated with ldptr.

Ldohseek returns SUCCESS or FAILURE. If the object file has no optional header or if it cannot seek to the optional header, ldohseek fails.

The program must be loaded with the object file access routine library libmld.a.

SEE ALSO
ldclose(3X), ldopen(3X), ldffhread(3X), ldfcn(4).
NAME
  ldopen, ldaopen – open a common object file for reading

SYNOPSIS
  #include <stdio.h>
  #include <fshldr.h>
  #include <sys.h>
  #include <ldfcn.h>

  LDFILE *ldopen (filename, ldptr)
  char *filename;
  LDFILE *ldptr;

  LDFILE *ldaopen (filename, oldptr)
  char *filename;
  LDFILE *oldptr;
  ld readst (ldptr, flags)
  LDFILE *ldptr;
  intflags;

DESCRIPTION
  Ldopen and ldclose(3X) provide uniform access to simple object files and to object files that are members of archive files. An archive of common object files can be processed as if it were a series of simple common object files.

  If ldptr has the value NULL, ldopen opens filename, allocates and initializes the LDFILE structure, and returns a pointer to the structure to the calling program.

  If ldptr is valid and TYPE(ldptr) is the archive magic number, ldopen reinitializes the LDFILE structure for the next archive member of filename.

  Ldopen and ldclose work in concert. Ldclose returns FAILURE only when TYPE(ldptr) is the archive magic number and there is another file in the archive to be processed. Only then should ldopen be called with the current value of ldptr. In all other cases, and particularly when a new filename is opened, ldopen should be called with a NULL ldptr argument.

  The following is a prototype for the use of ldopen and ldclose:

  /* for each filename to be processed */
  ldptr = NULL;
  do
    if ( (ldptr = ldopen(filename, ldptr)) != NULL )
      {
        /* check magic number */
        /* process the file */
      }
  while (ldclose(ldptr) == FAILURE );

  If the value of oldptr is not NULL, ldaopen opens filename anew and allocates and initializes a new LDFILE structure, copying the fields from oldptr. Ldaopen returns a pointer to the new LDFILE structure. This new pointer is independent of the old pointer, oldptr. The two pointers can be used concurrently to read separate parts of the object file. For example, one pointer can be used to step sequentially through the relocation information while the other is used to read indexed symbol table entries.

  Ldopen and ldaopen open filename for reading. If filename cannot be opened or if memory for the LDFILE structure cannot be allocated, both functions return NULL. A successful open does not ensure that the given file is a common object file or an archived object file.
Ldopen causes the symbol table header and file descriptor table to be read. Further access, using ldptr, causes other appropriate sections of the symbol table to be read (for example, if you call ldibread, the symbols or externals are read). To force sections for each symbol table in memory, call ldreadst with ST_<P> constants ORed together from st_support.h.

The program must be loaded with the object file access routine library libmld.a.

SEE ALSO
fopen(3S), ldclose(3X), ldfcn(4).
NAME

ldrseek, ldnrseek - seek to relocation entries of a section of a common object file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldrseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnrseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;

DESCRIPTION

Ldrseek seeks to the relocation entries of the section specified by sectindx of the common object file currently associated with ldptr.

Ldnrseek seeks to the relocation entries of the section specified by sectname.

Ldrseek and ldnrseek return SUCCESS or FAILURE. If sectindx is greater than the number of sections in the object file, ldrseek fails; if there is no section name corresponding with sectname, ldnrseek fails. If the specified section has no relocation entries or if it cannot seek to the specified relocation entries, either function fails.

NOTE: The first section has an index of one.

The program must be loaded with the object file access routine library libmld.a.

SEE ALSO

ldclose(3X), ldopen(3X), ldshread(3X), ldfcn(4).
NAME

ldshread, ldnshread — read an indexed/named section header of a common object file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <scnhdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldshread (ldptr, sectindx, secthead)
LDFILE *ldptr;
unsigned short sectindx;
SCNHDR *secthead;

int ldnshread (ldptr, sectname, secthead)
LDFILE *ldptr;
char *sectname;
SCNHDR *secthead;

DESCRIPTION

Ldshread reads the section header specified by sectindx of the common object file currently associated with ldptr into the area of memory beginning at secthead.

Ldnshread reads the section header specified by sectname into the area of memory beginning at secthead.

Ldshread and ldnshread return SUCCESS or FAILURE. If sectindx is greater than the number of sections in the object file, ldshread fails; If there is no section name corresponding with sectname, ldnshread fails. If it cannot read the specified section header, either function fails.

NOTE: The first section header has an index of one.

The program must be loaded with the object file access routine library libmld.a.

SEE ALSO

ldclose(3X), ldopen(3X), ldfcn(4).
NAME

ldsseek, ldnsseek – seek to an indexed/named section of a common object file

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldsseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnsseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;

DESCRIPTION

Ldsseek seeks to the section specified by sectindx of the common object file currently associated with ldptr.

Ldnsseek seeks to the section specified by sectname.

Ldsseek and ldnsseek return SUCCESS or FAILURE. If sectindx is greater than the number of sections in the object file, ldsseek fails; if there is no section name corresponding with sectname, ldnsseek fails. If there is no section data for the specified section or if it cannot seek to the specified section, either function fails.

NOTE: The first section has an index of one.

The program must be loaded with the object file access routine library libmlld.a.

SEE ALSO

ldclose(3X), ldopen(3X), ldshread(3X), ldfcn(4).
NAME
ldtbread – read an indexed symbol table entry of a common object file

SYNOPSIS
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldtbreader (ldptr, symindex, symbol)
LDFILE *ldptr;
long symindex;
SYM *symbol;

DESCRIPTION
Ldtbreader reads the symbol table entry specified by symindex of the common object file currently associated with ldptr into the area of memory beginning at symbol.

Ldtbreader returns SUCCESS or FAILURE. If symindex is greater than the number of symbols in the object file or if it cannot read the specified symbol table entry, ldtbreader fails.

The local and external symbols are concatenated into a linear list. Symbols are accessible from symnum zero to SYMHEADER(ldptr).symMax+SYMHEADER(ldptr).extiMax. The index and iss fields of the SYMR are made absolute (rather than file relative) so that routines ldgetname(3X), ldgetaux(3X), and ldtbreader (this routine) proceed normally given those indices. Only the "sym" part of externals is returned.

NOTE: The first symbol in the symbol table has an index of zero.

The program must be loaded with the object file access routine library libmld.a.

SEE ALSO
ldclose(3X), ldgetname(3X), ldopen(3X), ldtbseek(3X), ldgetname(3X), ldfcn(4).
NAME
ldtbseek – seek to the symbol table of a common object file

SYNOPSIS
#include <stdio.h>
#include <filehdr.h>
#include <sysm.h>
#include <$dfcn.h>

int ldtbseek (ldptr)
LDFILE *ldptr;

DESCRIPTION
Ldtbseek seeks to the symbol table of the object file currently associated with ldptr.
Ldtbseek returns SUCCESS or FAILURE. If the symbol table has been stripped from
the object file or if it cannot seek to the symbol table, ldtbseek fails.
The program must be loaded with the object file access routine library libld.a.

SEE ALSO
ldclose(3X), ldopen(3X), ldtbread(3X), ldfcn(4).
NAME
len – return length of Fortran string

SYNOPSIS
character*B ch
integer i
i = len(ch)

DESCRIPTION
len returns the length of string ch.
NAME
lgamma – log gamma function

SYNOPSIS
#include <math.h>
double lgamma(x)
double x;

DESCRIPTION
Lgamma returns ln |Γ(x)| where
Γ(x) = ∫₀^∞ t^(x-1) e^-t dt for x > 0 and
Γ(x) = π/(Γ(1-x) sin(πx)) for x < 1.

The external integer signgam returns the sign of Γ(x).

IDIOSYNCRASIES
Do not use the expression signgam*exp(lgamma(x)) to compute g := Γ(x). Instead use a program like this (in C):

g = lgamma(x); g = signgam*exp(g);

Only after lgamma has returned can signgam be correct. Note too that Γ(x) must overflow when x is large enough, underflow when -x is large enough, and spawn a division by zero when x is a nonpositive integer.

The following C program fragment might be used to calculate Γ if the overflow needs to be detected:

if ((y = lgamma(x)) > LN_MAXDOUBLE)
  error();
y = signgam * exp(y);

where LN_MAXDOUBLE is the least value that causes exp(3M) to overflow, and is defined in the <values.h> header file.

Only in the UNIX math library for C was the name gamma ever attached to lnΓ. Elsewhere, for instance in IBM’s FORTRAN library, the name GAMMA belongs to Γ and the name ALGAMA to lnΓ in single precision; in double the names are DGLAMA and DGLGAMA. Why should C be different?

Archaeological records suggest that C’s gamma originally delivered ln(Γ(|x|)). Later, the program gamma was changed to cope with negative arguments x in a more conventional way, but the documentation did not reflect that change correctly. The most recent change corrects inaccurate values when x is almost a negative integer, and lets Γ(x) be computed without conditional expressions. Programmers should not assume that lgamma has settled down.

At some time in the future, the name gamma will be rehabilitated and used for the gamma function, just as is done in FORTRAN. The reason for this is not so much compatibility with FORTRAN as a desire to achieve greater speed for smaller values of |x| and greater accuracy for larger values.

Meanwhile, programmers who have to use the name gamma in its former sense, for what is now lgamma, have two choices:

1) Change your source to use lgamma instead of gamma.
2) Add the following program to your others:

#include <math.h>
double gamma(x)
double x;
{
  return (lgamma(x));
}
DIAGNOSTICS

Gamma returns +\infty for negative integer arguments.

SEE ALSO

math(3M)
NAME
lib2648 – subroutines for the HP 2648 graphics terminal

SYNOPSIS
#include <stdio.h>

typedef char *bitmap;
FILE *trace;

cc file.c -l2648

DESCRIPTION
lib2648 is a general purpose library of subroutines useful for interactive graphics on the
Hewlett-Packard 2648 graphics terminal. To use it you must call the routine ttyinit() at the
beginning of execution, and done() at the end of execution. All terminal input and output
must go through the routines rawchar, readline, outchar, and outstr.

lib2648 does the necessary E/F handshaking if getenv("TERM") returns "hp2648", as it will
if set by tset(1). Any other value, including for example "2648", will disable handshaking.

Bit matrix routines are provided to model the graphics memory of the 2648. These routines
are generally useful, but are specifically useful for the update function which efficiently
changes what is on the screen to what is supposed to be on the screen. The primitive bit
matrix routines are neumat, mat, and setmat.

The file trace, if non-null, is expected to be a file descriptor as returned by fopen. If so,
lib2648 will trace the progress of the output by writing onto this file. It is provided to make
debugging output feasible for graphics programs without messiing up the screen or the escape
sequences being sent. Typical use of trace will include:

switch (argv[1][1]) {
    case 'P':
        trace = fopen("trace", "w");
        break;
    ...
    if (trace)
        fprintf(trace, "x is %d, y is %s\n", x, y);
    ...
        dumpmat("before update", xmat);

ROUTINES
agoto(x, y)
    Move the alphanumeric cursor to position (x, y), measured from the upper left corner
    of the screen.

aoff() Turn the alphanumeric display off.

aon() Turn the alphanumeric display on.

areaclear(rmin, cmin, rmax, cmax)
    Clear the area on the graphics screen bordered by the four arguments. In normal
    mode the area is set to all black, in inverse video mode it is set to all white.

beep() Ring the bell on the terminal.

bitcopy(dest, src, rows, cols) bitmap dest,
    Copy a rows by cols bit matrix from src to (user provided) dest.

cleara()
    Clear the alphanumeric display.

clear()
Clear the graphics display. Note that the 2648 will only clear the part of the screen
that is visible if zoomed in.

curoff()
Turn the graphics cursor off.

curon()
Turn the graphics cursor on.

dispmsg(str, x, y, maxlen) char *str;
Display the message str in graphics text at position (x, y). The maximum message
length is given by maxlen, and is needed for dispmsg to know how big an area to clear
before drawing the message. The lower left corner of the first character is at (x, y).

done() Should be called before the program exits. Restores the tty to normal, turns off
graphics screen, turns on alphanumeric screen, flushes the standard output, etc.

draw(x, y)
Draw a line from the pen location to (x, y). As with all graphics coordinates, (x, y) is
measured from the bottom left corner of the screen. (x, y) coordinates represent the
first quadrant of the usual Cartesian system.

drawbox(r, c, color, rows, cols)
Draw a rectangular box on the graphics screen. The lower left corner is at location (r,
c). The box is rows rows high and cols columns wide. The box is drawn if color is 1,
erased if color is 0. (r, c) absolute coordinates represent row and column on the
screen, with the origin at the lower left. They are equivalent to (x, y) except for being
reversed in order.

dumpmat(msg, m, rows, cols) char *msg; bitmat m;
If trace is non-null, write a readable ASCII representation of the matrix m on trace.
Msg is a label to identify the output.

emptyrow(m, rows, cols, r) bitmat m;
Returns 1 if row r of matrix m is all zero, else returns 0. This routine is provided
because it can be implemented more efficiently with a knowledge of the internal
representation than a series of calls to mat.

cerror(msg) char *msg;
Default error handler. Calls message(msg) and returns. This is called by certain rou-
tines in lib2648. It is also suitable for calling by the user program. It is probably a
good idea for a fancy graphics program to supply its own error procedure which uses
setjmp(3) to restart the program.

gdefault()
Set the terminal to the default graphics modes.

goff() Turn the graphics display off.

gon() Turn the graphics display on.
koff() Turn the keypad off.
kon() Turn the keypad on. This means that most special keys on the terminal (such as the
alphanumeric arrow keys) will transmit an escape sequence instead of doing their func-
tion locally.

tline(x1, y1, x2, y2)
Draw a line in the current mode from (x1, y1) to (x2, y2). This is equivalent to
move(x1, y1); draw(x2, y2); except that a bug in the terminal involving repeated lines
from the same point is compensated for.
lowlleft()
Move the alphanumeric cursor to the lower left (home down) position.

mat(m, rows, cols, r, c) bitmat m;
Used to retrieve an element from a bit matrix. Returns 1 or 0 as the value of the [r, c] element of the rows by cols matrix m. Bit matrices are numbered (r, c) from the upper left corner of the matrix, beginning at (0, 0). R represents the row, and c represents the column.

message(str) char *str;
Display the text message str at the bottom of the graphics screen.

minmax(g, rows, cols, rmin, cmin, rmax, cmax) bitmat g;
int *rmin, *cmin, *rmax, *cmax;
Find the smallest rectangle that contains all the 1 (on) elements in the bit matrix g. The coordinates are returned in the variables pointed to by rmin, cmin, rmax, cmax.

move(x, y)
Move the pen to location (x, y). Such motion is internal and will not cause output until a subsequent sync().

movecurs(x, y)
Move the graphics cursor to location (x, y).

bitmat newmat(rows, cols)
Create (with malloc(3)) a new bit matrix of size rows by cols. The value created (e.g. a pointer to the first location) is returned. A bit matrix can be freed directly with free.

outchar(c) char c;
Print the character c on the standard output. All output to the terminal should go through this routine or outstr.

outstr(str) char *str;
Print the string str on the standard output by repeated calls to outchar.

printg()
Print the graphics display on the printer. The printer must be configured as device 6 (the default) on the HPB.

char rawchar()
Read one character from the terminal and return it. This routine or readline should be used to get all input, rather than getchar(3).

rboff() Turn the rubber band line off.

rbon() Turn the rubber band line on.

char *rdchar(c) char c;
Return a readable representation of the character c. If c is a printing character it returns itself, if a control character it is shown in the `X notation, if negative an apostrophe is prepended. Space returns ^, rubout returns ^?.

NOTE: A pointer to a static place is returned. For this reason, it will not work to pass rdchar twice to the same fprintf/sprintf call. You must instead save one of the values in your own buffer with strcpy.

readline(prompt, msg, maxlen) char *prompt, *msg;
Display prompt on the bottom line of the graphics display and read one line of text from the user, terminated by a newline. The line is placed in the buffer msg, which has size maxlen characters. Backspace processing is supported.

setclear()
Set the display to draw lines in erase mode. (This is reversed by inverse video mode.)

```c
setmat(m, rows, cols, r, c, val) bitmat m;
```

The basic operation to store a value in an element of a bit matrix. The \([r, c]\) element of \(m\) is set to \(val\), which should be either 0 or 1.

```c
setset()
```

Set the display to draw lines in normal (solid) mode. (This is reversed by inverse video mode.)

```c
setxor()
```

Set the display to draw lines in exclusive or mode.

```c
sync()
```

Force all accumulated output to be displayed on the screen. This should be followed by `fflush(stdout)`. The cursor is not affected by this function. Note that it is normally never necessary to call `sync`, since `rawchar` and `readline` call `sync()` and `fflush(stdout)` automatically.

```c
togvid()
```

Toggle the state of video. If in normal mode, go into inverse video mode, and vice versa. The screen is reversed as well as the internal state of the library.

```c
ttyinit()
```

Set up the terminal for processing. This routine should be called at the beginning of execution. It places the terminal in `CBREAK` mode, turns off echo, sets the proper modes in the terminal, and initializes the library.

```c
update(mold, mnew, rows, cols, baser, basec) bitmat mold, mnew;
```

Make whatever changes are needed to make a window on the screen look like \(mnew\). \(Mold\) is what the window on the screen currently looks like. The window has size \(rows\) by \(cols\), and the lower left corner on the screen of the window is \([\text{baser}, \text{basec}]\). Note: `update` was not intended to be used for the entire screen. It would work but be very slow and take 64K bytes of memory just for `mold` and `mnew`. It was intended for 100 by 100 windows with objects in the center of them, and is quite fast for such windows.

```c
vidinv()
```

Set inverse video mode.

```c
vidnorm()
```

Set normal video mode.

```c
zermat(m, rows, cols) bitmat m;
```

Set the bit matrix \(m\) to all zeros.

```c
zoomn(size)
```

Set the hardware zoom to value \(size\), which can range from 1 to 15.

```c
zoomoff()
```

Turn zoom off. This forces the screen to zoom level 1 without affecting the current internal zoom number.

```c
zoomon()
```

Turn zoom on. This restores the screen to the previously specified zoom size.

**DIAGNOSTICS**

The routine `error` is called when an error is detected. The only error currently detected is overflow of the buffer provided to `readline`.

Subscripts out of bounds to `setmat` return without setting anything.
FILES
/usr/lib/lib2648.a

SEE ALSO
fed(1)

AUTHOR
Mark Horton

ERROR
This library is not supported. It makes no attempt to use all of the features of the terminal, only those needed by fed. Contributions from users will be accepted for addition to the library.

The HP 2648 terminal is somewhat unreliable at speeds over 2400 baud, even with the ^E/^F handshaking. In an effort to improve reliability, handshaking is done every 32 characters. (The manual claims it is only necessary every 80 characters.) Nonetheless, I/O errors sometimes still occur.

There is no way to control the amount of debugging output generated on trace without modifying the source to the library.
NAME
VADS libraries – overview of VADS libraries

DESCRIPTION
VADS includes libraries containing packages and functions that may be referenced by user applications and a directory of examples using them.

Libraries contained in the current release of the VADS are listed below. The exact contents varies with each implementation.

- standard — predefined Ada packages and additional packages to implement them
- verdixlib — Verdix-supplied packages
- publiclib* — public domain packages written in Ada
- examples* — sample Ada program files

*Note: publiclib and examples are neither supported nor warranted by VERDIX.
NAME
link – make a link to an existing file

SYNOPSIS
function link (name1, name2)
    character(*) name1, name2

integer function symlink (name1, name2)
    character(*) name1, name2

DESCRIPTION
Name1 must be the pathname of an existing file. Name2 is a pathname to be linked to file
name1. Name2 must not already exist. The returned value will be 0 if successful; a system
error code otherwise.

Symlink creates a symbolic link to name1.

FILES
/usr/lib/libU77.a

SEE ALSO
link(2), symlink(2), perror(3F), unlink(3F)

BUGS
Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.
NAME
loc – return the address of an object

SYNOPSIS
function loc (arg)

DESCRIPTION
The returned value will be the address of arg.

FILES
/usr/lib/libU77.a
NAME
lockf – advisory record locking on files

SYNOPSIS

```c
#include <unistd.h>

#define F_ULOCK  0 /* Unlock a previously locked section */
#define F_LOCK   1 /* Lock a section for exclusive use */
#define F_TLOCK  2 /* Test and lock a section (non-blocking) */
#define F_TEST   3 /* Test section for other process' locks */

lockf(fd, cmd, size)
int fd, cmd;
long size;
```

DESCRIPTION

`lockf` may be used to test, apply, or remove an advisory record lock on the file associated with the open descriptor `fd`. (See `fcntl(2)` for more information about advisory record locking.)

A lock is obtained by specifying a `cmd` parameter of `F_LOCK` or `F_TLOCK`. To unlock an existing lock, the `F_ULOCK` `cmd` is used. `F_TEST` is used to detect if a lock by another process is present on the specified segment.

`F_LOCK` and `F_TLOCK` requests differ only by the action taken if the lock may not be immediately granted. `F_TLOCK` will cause the function to return -1 and set `errno` to `EAGAIN` if the section is already locked by another process. `F_LOCK` will cause the process to sleep until the lock may be granted or a signal is caught.

`size` is the number of contiguous bytes to be locked or unlocked. The lock starts at the current file offset in the file and extends forward for a positive `size` or backward for a negative `size` (preceeding but not including the current offset). A segment need not be allocated to the file in order to be locked; however, a segment may not extend to a negative offset relative to the beginning of the file. If `size` is zero, the lock will extend from the current offset through the end-of-file. If such a lock starts at offset 0, then the entire file will be locked (regardless of future file extensions).

NOTES

The descriptor `fd` must have been opened with `O_WRONLY` or `O_RDWR` permission in order to establish locks with this function call.

All locks associated with a file for a given process are removed when the file is closed or the process terminates. Locks are not inherited by the child process in a `fork(2)` system call.

RETURN VALUE

Zero is returned on success, -1 on error, with an error code stored in `errno`.

ERRORS

`lockf` will fail if one or more of the following are true:

- **EBADF**
  - `fd` is not a valid open descriptor.

- **EBADF**
  - `cmd` is `F_LOCK` or `F_TLOCK` and the process does not have write permission on the file.

- **EAGAIN**
  - `cmd` is `F_TLOCK` or `F_TEST` and the section is already locked by another process.

- **EINVAL**
  - `cmd` is `F_LOCK` and a signal interrupted the process while it was waiting for the lock to be granted.

- **ENOMEM**
  - `cmd` is `F_LOCK`, `F_TLOCK`, or `F_ULOCK` and there are no more file lock entries available.
SEE ALSO
fcntl(2), lockd(8C)

ERRORS
File locks obtained through the `lockf` mechanism do not interact in any way with those acquired via `flock(2)`. They do, however, work correctly with the locks claimed by `fcntl(2)`.
NAME
malloc, free, realloc, calloc, alloca – memory allocator

SYNOPSIS
char *malloc(size)
unsigned size;
free(ptr)
char *ptr;
char *realloc(ptr, size)
char *ptr;
unsigned size;
char *calloc(nelem, elsize)
unsigned nelem, elsize;
char *alloca(size)
int size;

DESCRIPTION
malloc and free provide a general-purpose memory allocation package. malloc returns a
pointer to a block of at least size bytes beginning on a word boundary.
The argument to free is a pointer to a block previously allocated by malloc; this space is made
available for further allocation, but its contents are left undisturbed.
Needless to say, grave disorder will result if the space assigned by malloc is overrun or if some
random number is handed to free.
malloc maintains multiple lists of free blocks according to size, allocating space from the
appropriate list. It calls sbrk (see brk(2)) to get more memory from the system when there is
no suitable space already free.
realloc changes the size of the block pointed to by ptr to size bytes and returns a pointer to the
(possibly moved) block. The contents will be unchanged up to the lesser of the new and
old sizes.
In order to be compatible with older versions, realloc also works if ptr points to a block freed
since the last call of malloc, realloc or calloc; sequences of free, malloc and realloc were previously
used to attempt storage compaction. This procedure is no longer recommended.
calloc allocates space for an array of nelem elements of size elsize. The space is initialized to
zeros.
alloca allocates size bytes of space in the stack frame of the caller. This temporary space is
automatically freed on return.
Each of the allocation routines returns a pointer to space suitably aligned (after possible
pointer coercion) for storage of any type of object. If the space is of pagesize or larger, the
memory returned will be page-aligned.

SEE ALSO
brk(2), pagesize(2)

DIAGNOSTICS
malloc, realloc and calloc return a null pointer (0) if there is no available memory or if the
arena has been detectably corrupted by storing outside the bounds of a block. malloc may be
recompiled to check the arena very stringently on every transaction; those sites with a source
code license may check the source code to see how this can be done.
ERRORS

When realloc returns 0, the block pointed to by ptr may be destroyed.

The current implementation of malloc does not always fail gracefully when system memory limits are approached. It may fail to allocate memory when larger free blocks could be broken up, or when limits are exceeded because the size is rounded up. It is optimized for sizes that are powers of two.

alloca is machine dependent; its use is discouraged.
NAME
math – introduction to mathematical library functions

DESCRIPTION
These functions constitute the C math library libm. There are two versions of the math library libm.a and libm43.a.

The first, libm.a, contains routines written in MIPS assembly language and tuned for best performance and includes many routines for the float data type. The routines in there are based on the algorithms of Cody and Waite or those in the 4.3 BSD release, whichever provides the best performance with acceptable error bounds. Those routines with Cody and Waite implementations are marked with a ‘v’ in the list of functions below.

The second version of the math library, libm43.a, contains routines all based on the original codes in the 4.3 BSD release. The difference between the two version's error bounds is typically around 1 unit in the last place, whereas the performance difference may be a factor of two or more.

The link editor searches this library under the “-lm” (or “-lm43”) option. Declarations for these functions may be obtained from the include file <math.h>. The Fortran math library is described in “man 3f intro”.

LIST OF FUNCTIONS
The cycle counts of all functions are approximate; cycle counts often depend on the value of argument. The error bound sometimes applies only to the primary range.

<table>
<thead>
<tr>
<th>Name</th>
<th>Appears on Page</th>
<th>Description</th>
<th>Error Bound (ULPs)</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>acos</td>
<td>sin.3m</td>
<td>inverse trigonometric function</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>acosh</td>
<td>asinh.3m</td>
<td>inverse hyperbolic function</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>asin</td>
<td>sin.3m</td>
<td>inverse trigonometric function</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>asinh</td>
<td>asinh.3m</td>
<td>inverse hyperbolic function</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>atan</td>
<td>sin.3m</td>
<td>inverse trigonometric function</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>atanh</td>
<td>asinh.3m</td>
<td>inverse hyperbolic function</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>atan2</td>
<td>sin.3m</td>
<td>inverse trigonometric function</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>cabs</td>
<td>hypot.3m</td>
<td>complex absolute value</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>cbzt</td>
<td>sqrt.3m</td>
<td>cube root</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ceil</td>
<td>floor.3m</td>
<td>integer no less than</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>copysign</td>
<td>ieee.3m</td>
<td>copy sign bit</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cos</td>
<td>sin.3m</td>
<td>trigonometric function</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>cosh</td>
<td>sinh.3m</td>
<td>hyperbolic function</td>
<td>?</td>
<td>3</td>
</tr>
<tr>
<td>drem</td>
<td>ieee.3m</td>
<td>remainder</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>erf</td>
<td>erf.3m</td>
<td>error function</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>erfc</td>
<td>erf.3m</td>
<td>complementary error function</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>exp</td>
<td>exp.3m</td>
<td>exponential</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>expm1</td>
<td>exp.3m</td>
<td>exp(x)-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>fabs</td>
<td>floor.3m</td>
<td>absolute value</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fatan</td>
<td>sin.3m</td>
<td>inverse trigonometric function</td>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>fcosh</td>
<td>sin.3m</td>
<td>trigonometric function</td>
<td>1</td>
<td>87</td>
</tr>
<tr>
<td>fcosh*</td>
<td>sinh.3m</td>
<td>hyperbolic function</td>
<td>?</td>
<td>105</td>
</tr>
<tr>
<td>fexp</td>
<td>exp.3m</td>
<td>exponential</td>
<td>1</td>
<td>79</td>
</tr>
<tr>
<td>flog</td>
<td>exp.3m</td>
<td>natural logarithm</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>floor</td>
<td>floor.3m</td>
<td>integer no greater than</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fsin</td>
<td>sin.3m</td>
<td>trigonometric function</td>
<td>1</td>
<td>68</td>
</tr>
</tbody>
</table>
## MATH (3M) RISC/os Programmer's Reference MATH (3M)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsinh</td>
<td>hyperbolic function</td>
<td>??</td>
</tr>
<tr>
<td>fsqrt</td>
<td>square root</td>
<td>1</td>
</tr>
<tr>
<td>ftnz</td>
<td>trigonometric function</td>
<td>??</td>
</tr>
<tr>
<td>ftnh</td>
<td>hyperbolic function</td>
<td>??</td>
</tr>
<tr>
<td>hypot</td>
<td>Euclidean distance</td>
<td>1</td>
</tr>
<tr>
<td>j0</td>
<td>bessel function</td>
<td>??</td>
</tr>
<tr>
<td>j1</td>
<td>bessel function</td>
<td>??</td>
</tr>
<tr>
<td>jn</td>
<td>bessel function</td>
<td>??</td>
</tr>
<tr>
<td>lgamma</td>
<td>log gamma function</td>
<td>??</td>
</tr>
<tr>
<td>log</td>
<td>natural logarithm</td>
<td>2</td>
</tr>
<tr>
<td>logb</td>
<td>exponent extraction</td>
<td>0</td>
</tr>
<tr>
<td>log10</td>
<td>logarithm to base 10</td>
<td>3</td>
</tr>
<tr>
<td>log1p</td>
<td>log(1+x)</td>
<td>1</td>
</tr>
<tr>
<td>pow</td>
<td>exponential x**y</td>
<td>60-500</td>
</tr>
<tr>
<td>rint</td>
<td>round to nearest integer</td>
<td>0</td>
</tr>
<tr>
<td>scalb</td>
<td>exponent adjustment</td>
<td>0</td>
</tr>
<tr>
<td>sin</td>
<td>trigonometric function</td>
<td>2</td>
</tr>
<tr>
<td>sinh</td>
<td>hyperbolic function</td>
<td>??</td>
</tr>
<tr>
<td>sqrt</td>
<td>square root</td>
<td>1</td>
</tr>
<tr>
<td>tanh</td>
<td>hyperbolic function</td>
<td>??</td>
</tr>
<tr>
<td>y0</td>
<td>bessel function</td>
<td>??</td>
</tr>
<tr>
<td>y1</td>
<td>bessel function</td>
<td>??</td>
</tr>
<tr>
<td>yn</td>
<td>bessel function</td>
<td>??</td>
</tr>
</tbody>
</table>

### NOTES

In 4.3 BSD, distributed from the University of California in late 1985, most of the foregoing functions come in two versions, one for the double-precision "D" format in the DEC VAX-11 family of computers, another for double-precision arithmetic conforming to the IEEE Standard 754 for Binary Floating-Point Arithmetic. The two versions behave very similarly, as should be expected from programs more accurate and robust than was the norm when UNIX was born. For instance, the programs are accurate to within the numbers of ulps tabulated above; an ulp is one Unit in the Last Place. And the programs have been cured of anomalies that afflicted the older math library libm in which incidents like the following had been reported:

- \[ \sqrt{(-1.0)} = 0.0 \text{ and } \log(-1.0) = -1.7e38. \]
- \[ \cos(1.0e-11) > \cos(0.0) > 1.0. \]
- \[ \text{pow}(x,1.0) \neq x \text{ when } x = 2.0, 3.0, 4.0, ..., 9.0. \]
- \[ \text{pow}(-1.0,1.0e10) \text{ trapped on Integer Overflow.} \]
- \[ \sqrt{1.0e30} \text{ and } \sqrt{1.0e-30} \text{ were very slow.} \]

MIPS machines conform to the IEEE Standard 754 for Binary Floating-Point Arithmetic, to which only the notes for IEEE floating-point apply and are included here.

### IEEE STANDARD 754 Floating-Point Arithmetic:

This standard is on its way to becoming more widely adopted than any other design for computer arithmetic.

The main virtue of 4.3 BSD's libm codes is that they are intended for the public domain; they may be copied freely provided their provenance is always acknowledged, and provided users assist the authors in their researches by reporting experience with the codes. Therefore no user of UNIX on a machine that conforms to IEEE 754 need use anything worse than the new libm.

---

Properties of IEEE 754 Double-Precision:
Wordsize: 64 bits, 8 bytes. Radix: Binary.
Precision: 53 significant bits, roughly like 16 significant decimals.
If \( x \) and \( x' \) are consecutive positive Double-Precision numbers (they differ by 1 ulp), then
\[
1.1e-16 < 0.5^{53} < (x'-x)/x \leq 0.5^{52} < 2.3e-16.
\]
Range: Overflow threshold \( = 2.0^{1024} = 1.8e308 \)
Underflow threshold \( = 2.0^{1022} = 2.2e-308 \)
Overflow goes by default to a signed \( \infty \).
Underflow is \textit{Gradual}, rounding to the nearest integer multiple of \( 0.5^{1074} = 4.9e-324 \).

Zero is represented ambiguously as \(+0\) or \(-0\).
Its sign transforms correctly through multiplication or division, and is
preserved by addition of zeros with like signs; but \( x-x \) yields \(+0\) for every finite \( x \). The only operations that reveal zero’s sign are division by zero and
\( \text{copysign}(x,\pm0) \). In particular, comparison \((x > y, x \geq y, \text{ etc.})\) cannot be
affected by the sign of zero; but if finite \( x = y \) then \( \infty = 1/(x-y) \neq -1/(y-x) = -\infty \).

\( \infty \) is signed.

it persists when added to itself or to any finite number. Its sign transforms
correctly through multiplication and division, and \((\text{finite})/\pm\infty = \pm 0\)
\((\text{nonzero})/0 = \pm \infty \). But \( \infty-\infty, \infty\times0 \) and \( \infty/\infty \) are, like \( 0/0 \) and \( \text{sqrt}(-3) \),
invalid operations that produce \( NaN \).

Reserved operands:
there are \( 2^{53}-2 \) of them, all called \( NaN \) (Not a Number). Some, called Signaling \( NaNs \), trap any floating-point operation performed upon them; they
could be used to mark missing or uninitialized values, or nonexistent elements
of arrays. The rest are Quiet \( NaNs \); they are the default results of Invalid
Operations, and propagate through subsequent arithmetic operations. If \( x \neq x \)
then \( x \) is \( NaN \); every other predicate \((x > y, x = y, x < y, \ldots)\) is FALSE if
\( NaN \) is involved.

NOTE: Trichotomy is violated by \( NaN \).
Besides being FALSE, predicates that entail ordered comparison,
rather than mere \((in)equality\), signal Invalid Operation when \( NaN \) is
involved.

Rounding:
Every algebraic operation \((+, -, *, /, \sqrt{\cdot})\) is rounded by default to within half
an \( ulp \), and when the rounding error is exactly half an \( ulp \) then the rounded
value’s least significant bit is zero. This kind of rounding is usually the best
kind, sometimes provably so; for instance, for every \( x = 1.0, 2.0, 3.0, 4.0, \ldots \),
\( 2.0^{52} \), we find \((x/3.0)\times3.0 = x \) and \((x/10.0)\times10.0 = x \) and \( \ldots \) despite that
both the quotients and the products have been rounded. Only rounding like
IEEE 754 can do that. But no single kind of rounding can be proved best for
every circumstance, so IEEE 754 provides rounding towards zero or towards
\( +\infty \) or towards \( -\infty \) at the programmer’s option. And the same kinds of
rounding are specified for Binary-Decimal Conversions, at least for magnitudes
between roughly \( 1.0e-10 \) and \( 1.0e37 \).
Exceptions:

IEEE 754 recognizes five kinds of floating-point exceptions, listed below in declining order of probable importance.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Default Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid Operation</td>
<td>NaN, or FALSE</td>
</tr>
<tr>
<td>Overflow</td>
<td>±∞</td>
</tr>
<tr>
<td>Divide by Zero</td>
<td>±∞</td>
</tr>
<tr>
<td>Underflow</td>
<td>Gradual Underflow</td>
</tr>
<tr>
<td>Inexact</td>
<td>Rounded value</td>
</tr>
</tbody>
</table>

NOTE: An Exception is not an Error unless handled badly. What makes a class of exceptions exceptional is that no single default response can be satisfactory in every instance. On the other hand, if a default response will serve most instances satisfactorily, the unsatisfactory instances cannot justify aborting computation every time the exception occurs.

For each kind of floating-point exception, IEEE 754 provides a Flag that is raised each time its exception is signaled, and stays raised until the program resets it. Programs may also test, save and restore a flag. Thus, IEEE 754 provides three ways by which programs may cope with exceptions for which the default result might be unsatisfactory:

1) Test for a condition that might cause an exception later, and branch to avoid the exception.

2) Test a flag to see whether an exception has occurred since the program last reset its flag.

3) Test a result to see whether it is a value that only an exception could have produced.

CAUTION: The only reliable ways to discover whether Underflow has occurred are to test whether products or quotients lie closer to zero than the underflow threshold, or to test the Underflow flag. (Sums and differences cannot underflow in IEEE 754; if \( x \neq y \) then \( x-y \) is correct to full precision and certainly nonzero regardless of how tiny it may be.) Products and quotients that underflow gradually can lose accuracy gradually without vanishing, so comparing them with zero (as one might on a VAX) will not reveal the loss. Fortunately, if a gradually underflowed value is destined to be added to something bigger than the underflow threshold, as is almost always the case, digits lost to gradual underflow will not be missed because they would have been rounded off anyway. So gradual underflows are usually provably ignorable. The same cannot be said of underflows flushed to 0.

At the option of an implementor conforming to IEEE 754, other ways to cope with exceptions may be provided:

4) ABORT. This mechanism classifies an exception in advance as an incident to be handled by means traditionally associated with error-handling statements like "ON ERROR GO TO ...". Different languages offer different forms of this statement, but most share the following characteristics:

- No means is provided to substitute a value for the offending operation's result and resume computation from what may be the middle of an expression. An exceptional result is abandoned.
- In a subprogram that lacks an error-handling statement, an exception causes the subprogram to abort within whatever program called it, and so on back up the
chain of calling subprograms until an error-handling statement is encountered or
the whole task is aborted and memory is dumped.

5) STOP. This mechanism, requiring an interactive debugging environment, is more
for the programmer than the program. It classifies an exception in advance as a
symptom of a programmer's error; the exception suspends execution as near as it
can to the offending operation so that the programmer can look around to see
how it happened. Quite often the first several exceptions turn out to be quite
unexceptionable, so the programmer ought ideally to be able to resume execution
after each one as if execution had not been stopped.

6) ... Other ways lie beyond the scope of this document.

The crucial problem for exception handling is the problem of Scope, and the problem's solu-
tion is understood, but not enough manpower was available to implement it fully in time to be
distributed in 4.3 BSD's libm. Ideally, each elementary function should act as if it were indi-
visible, or atomic, in the sense that ...

i) No exception should be signaled that is not deserved by the data supplied to that func-
tion.

ii) Any exception signaled should be identified with that function rather than with one of its
subroutines.

iii) The internal behavior of an atomic function should not be disrupted when a calling pro-
gram changes from one to another of the five or so ways of handling exceptions listed
above, although the definition of the function may be correlated intentionally with excep-
tion handling.

Ideally, every programmer should be able conveniently to turn a debugged subprogram into one
that appears atomic to its users. But simulating all three characteristics of an atomic function
is still a tedious affair, entailing hosts of tests and saves–restores; work is under way to ameliorate the inconvenience.

Meanwhile, the functions in libm are only approximately atomic. They signal no inappropriate
exception except possibly ...

Over/Underflow
when a result, if properly computed, might have lain barely within range, and
Inexact in cabs, cbrt, hypot, log10 and pow
when it happens to be exact, thanks to fortuitous cancellation of errors.

Otherwise, ...

Invalid Operation is signaled only when
any result but NaN would probably be misleading.
Overflow is signaled only when
the exact result would be finite but beyond the overflow threshold.
Divide–by–Zero is signaled only when
a function takes exactly infinite values at finite operands.
Underflow is signaled only when
the exact result would be nonzero but tinier than the underflow threshold.
Inexact is signaled only when
greater range or precision would be needed to represent the exact result.

Exceptions on MIPS machines:
The exception enables and the flags that are raised when an exception occurs (as well
as the rounding mode) are in the floating–point control and status register. This regis-
ter can be read or written by the routines described on the man page fpe(3). This
register's layout is described in the file <mips/fpu.h> in UMIPS–BSD releases and in
in UMIPS–SYSV releases.

A full implementation of IEEE 754 "user trap handlers" is under development at MIPS computer systems. At which time all functions in libm will appear atomic and the full functionality of user trap handlers will be supported in those language without other floating–point error handling intrinsics (i.e. ADA, PI/1, etc). For a description of these trap handlers see section 8 of the IEEE 754 standard.

What is currently available is only the raw interface which was only intended to be used by the code to implement IEEE user trap handlers. IEEE floating–point exceptions are enabled by setting the enable bit for that exception in the floating–point control and status register. If an exception then occurs the UNIX signal SIGFPE is sent to the process. It is up to the signal handler to determine the instruction that caused the exception and to take the action specified by the user. The instruction that caused the exception is in one of two places. If the floating–point board is used (the floating–point implementation revision register indicates this in it's implementation field) then the instruction that caused the exception is in the floating–point exception instruction register. In all other implementations the instruction that caused the exception is at the address of the program counter as modified by the branch delay bit in the cause register. Both the program counter and cause register are in the sigcontext structure passed to the signal handler (see signal(3)). If the program is to be continued past the instruction that caused the exception the program counter in the signal context must be advanced. If the instruction is in a branch delay slot then the branch must be emulated to determine if the branch is taken and then the resulting program counter can be calculated (see emulate_branch(3) and the NOTES (MIPS) section in signal(3)).

BUGS
When signals are appropriate, they are emitted by certain operations within the codes, so a subroutine–trace may be needed to identify the function with its signal in case method 5) above is in use. And the codes all take the IEEE 754 defaults for granted; this means that a decision to trap all divisions by zero could disrupt a code that would otherwise get correct results despite division by zero.

SEE ALSO
fpc(3), signal(3), emulate_branch(3)

R2010 Floating Point Coprocessor Architecture
R2360 Floating Point Board Product Description


AUTHOR
W. Kahan, with the help of Z–S. Alex Liu, Stuart I. McDonald, Dr. Kwok–Choi Ng, Peter Tang.
NAME
memory: memccpy, memchr, memcmp, memcpy, memset — memory operations

SYNOPSIS
#include <memory.h>

char *memccpy (s1, s2, c, n) char *s1, *s2; int c, n;

char *memchr (s, c, n) char *s; int c, n;

int memcmp (s1, s2, n) char *s1, *s2; int n;

char *memcpy (s1, s2, n) char *s1, *s2; int n;

char *memset (s, c, n) char *s; int c, n;

DESCRIPTION
These functions operates efficiently as possible on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

memccpy copies characters from memory area s2 into s1, stopping after the first occurrence of character c has been copied, or after n characters have been copied, whichever comes first. It returns a pointer to the character after the copy of c in s1, or a NULL pointer if c was not found in the first n characters of s2.

memchr returns a pointer to the first occurrence of character c in the first n characters of memory area s, or a NULL pointer if c does not occur.

memcmp compares its arguments, looking at the first n characters only, and returns an integer less than, equal to, or greater than 0, according as s1 is lexicographically less than, equal to, or greater than s2.

memcpy copies n characters from memory area s2 to s1. It returns s1.
memset sets the first n characters in memory area s to the value of character c. It returns s.

For user convenience, all these functions are declared in the optional <memory.h> header file.

CAVEATS memcmp is implemented by using the most natural character comparison on the machine. Thus, the sign of the value returned when one of the characters has its high order bit set is not the same in all implementations and should not be relied upon.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.
NAME
mktemp – make a unique file name

SYNOPSIS
char *mktemp(template)
char *template;

mkstemp(template)
char *template;

DESCRIPTION
mktemp creates a unique file name, typically in a temporary filesystem, by replacing template with a unique file name, and returns the address of the template. The template should contain a file name with six trailing X's, which are replaced with the current process id and a unique letter. mkstemp makes the same replacement to the template but returns a file descriptor for the template file open for reading and writing. mkstemp avoids the race between testing whether the file exists and opening it for use.

SEE ALSO
getpid(2), open(2)

DIAGNOSTICS
mktemp returns an open file descriptor upon success. It returns -1 if no suitable file could be created.
NAME
monitor, monstartup, moncontrol – prepare execution profile

SYNOPSIS
monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (slowpc)(), (shighpc)();
short buffer[];

monstartup(lowpc, highpc)
int (slowpc)(), (shighpc)();

moncontrol(mode)

DESCRIPTION
These functions use the profil(2) system call to control program-counter sampling. Using the option -p when compiling or linking a program (see The MIPS Languages Programmer Guide) automatically generates calls to these functions. You need not call them explicitly unless you want finer control.

Typically, you would call either monitor or monstartup to initialize pc-sampling and enable it; call moncontrol to disable or reenable it; and call monitor again at the end of execution to disable sampling and record the samples in a file.

Your initial call to monitor enables pc-sampling. Lowpc and highpc specify the range of addresses to be sampled; the lowest address is that of lowpc and the highest is just below highpc. Buffer is the address of a (user allocated) array of bufsize short integers, which holds a record of the samples; for best results, the buffer should not be less than a few times smaller than the range of addresses sampled. nfunc is ignored.

The environment variable PROFDIR determines the name of the output file and whether pc-sampling takes place: if it is not set, the file is named "mon.out"; if set to the empty string, no pc-sampling occurs; if set to a non-empty string, the file is named "string/pid.progname", where "pid" is the process id of the executing program and "progname" is the program's name as it appears in argv[0]. The subdirectory "string" must already exist.

To profile the entire program, use:
extern eprol(), etext();

... monitor(eprol, etext, buf, bufsize, 0);

eprol lies just below the user program text, and etext lies just above it, as described in end(3). (Because the user program does not necessarily start at a low memory address, using a small number in place of eprol is dangerous).

monstartup is an alternate form of monitor that calls sbrk(2) for you to allocate the buffer.
moncontrol selectively disables and re-enables pc-sampling within a program, allowing you to measure the cost of particular operations. moncontrol(0) disables pc-sampling, and moncontrol(1) reenables it.

To stop execution monitoring and write the results in the output file, use:

monitor(0);
FILES
mon.out   default name for output file
libprof1.a routines for pc-sampling

SEE ALSO
cc(1), prof(1), profil(2), sbrk(2), end(3), ld(1) and The MIPS Languages Programmer Guide.
NAME
  mount – keep track of remotely mounted filesystems

SYNOPSIS
  #include <rpcsvc/mount.h>

RPC INFO
  program number:
    MOUNTPROG
  xdr routines:
    xdr_exportbody(xdrs, ex)
      XDR *xdrs;
      struct exports *ex;
    xdr_exports(xdrs, ex);
      XDR *xdrs;
      struct exports **ex;
    xdr_fhandle(xdrs, fh);
      XDR *xdrs;
      fhandle_t *fp;
    xdr_fhstatus(xdrs, fhs);
      XDR *xdrs;
      struct fhstatus *fhs;
    xdr_groups(xdrs, gr);
      XDR *xdrs;
      struct groups *gr;
    xdr_mountbody(xdrs, ml)
      XDR *xdrs;
      struct mountlist *ml;
    xdr_mountlist(xdrs, ml);
      XDR *xdrs;
      struct mountlist **ml;
    xdr_path(xdrs, path);
      XDR *xdrs;
      char **path;

procs:
  MOUNTPROC_MNT
    argument of xdr_path, returns fhstatus.
    Requires unix authentication.
  MOUNTPROC_DUMP
    no args, returns struct mountlist
  MOUNTPROC_UMNT
    argument of xdr_path, no results.
    requires unix authentication.
  MOUNTPROC_UMNTALL
    no arguments, no results.
    requires unix authentication.
    umounts all remote mounts of sender.
  MOUNTPROC_EXPORT
  MOUNTPROC_EXPORTALL
    no args, returns struct exports

versions:
  MOUNTVERS_ORIG
structures:
struct mountlist {
    char *ml_name;
    char *ml_path;
    struct mountlist *ml_nxt;
};
struct fhstatus {
    int fhs_status;
    fhandle_t fhs_fh;
};
/*
* List of exported directories
* An export entry with ex_groups
* NULL indicates an entry which is exported to the world.
*/
struct exports {
    dev_t ex_dev;      /* dev of directory */
    char *ex_name;     /* name of directory */
    struct groups *ex_groups;    /* groups allowed to mount this entry */
    struct exports *ex_next;
};
struct groups {
    char *g_name;
    struct groups *g_next;
};

SEE ALSO
mount(8), showmount(8), mountd(8C),
NAME
madd, msub, mult, mdiv, pow, gcd, invert, rpow, msqrt, mcmp, move, min, omin, fmin, m_in, mout, omout, fmout, m_out, sdiv, itom – multiple precision integer arithmetic

SYNOPSIS
#include <mp.h>
#include <stdio.h>
typedef struct mint { int len; short *val;} MINT;
madd(a, b, c)
msub(a, b, c)
mult(a, b, c)
mdiv(a, b, q, r)
pow(a, b, m, c)
gcd(a, b, c)
invert(a, b, c)
rpow(a, n, c)
msqrt(a, b, r)
mcmp(a, b)
move(a, b)
min(a)
omin(a)
fmin(a, f)
m_in(a, n, f)
mout(a)
omout(a)
fmout(a, f)
m_out(a, n, f)
MINT *a, *b, *c, *m, *q, *r;
FILE *f;
int n;
sdiv(a, n, q, r)
MINT *a, *q;
short n;
short *r;
MINT *itom(n)

DESCRIPTION
These routines 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 function itom which sets the initial value to n. After that, space is managed automatically by the routines.

madd, msub and mult assign to c the sum, difference and product, respectively, of a and b.
mdiv assigns to q and r the quotient and remainder obtained from dividing a by b. sdiv is like mdiv except that the divisor is a short integer n and the remainder is placed in a short whose address is given as r. msqrt produces the integer square root of a in b and places the remainder in r. rpow calculates in c the value of a raised to the ("regular" integral) power n, while pow calculates this with a full multiple precision exponent b and the result is reduced modulo m. gcd returns the greatest common denominator of a and b in c, and invert computes c such that a*c mod b = 1, for a and b relatively prime. mcmp returns a negative, zero or positive integer value when a is less than, equal to or greater than b, respectively. move copies a to b. min and mout do decimal input and output while omin and omout do octal input and output. More generally, fmin and fmout do decimal input and output using file f, and m_in and m_out do I/O 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.

Programs which use the multiple-precision arithmetic library must be loaded using the loader flag \(-\text{imp}\).

FILES

\begin{verbatim}
/usr/include/mp.h include file
/usr/lib/libmp.a object code library
\end{verbatim}

SEE ALSO

dc(1), bc(1)

DIAGNOSTICS

Illegal operations and running out of memory produce messages and core images.

ERRORS

Bases for input and output should be \(\leq 10\).

dc(1) and bc(1) don't use this library.
The input and output routines are a crock.
pow is also the name of a standard math library routine.
NAME
dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey,
dbm_error, dbm_clearerr – data base subroutines

SYNOPSIS
#include <ndbm.h>

typedef struct {
    char *dptr;
    int dsize;
} datum;

DBM *dbm_open(file, flags, mode)
    char *file;
    int flags, mode;

void dbm_close(db)
    DBM *db;

datum dbm_fetch(db, key)
    DBM *db;
    datum key;

int dbm_store(db, key, content, flags)
    DBM *db;
    datum key, content;
    int flags;

int dbm_delete(db, key)
    DBM *db;
    datum key;

datum dbm_firstkey(db)
    DBM *db;

datum dbm_nextkey(db)
    DBM *db;

int dbm_error(db)
    DBM *db;

int dbm_clearerr(db)
    DBM *db;

DESCRIPTION
These functions maintain key/content pairs in a data base. The functions will handle very
large (a billion blocks) databases and will access a keyed item in one or two file system
accesses. This package replaces the earlier dbm(3x) library, which managed only a single data-
base.

Keys and contents are described by the datum typedef. A datum specifies a string of dsize
bytes pointed to by dptr. Arbitrary binary data, as well as normal ASCII strings, are allowed.
The data base 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 dbm_open. This will open and/or
create the files file.dir and file.pag depending on the flags parameter (see open(2)).

Once open, the data stored under a key is accessed by dbm_fetch and data is placed under a
key by dbm_store. The flags field can be either DBM_INSERT or DBM_REPLACE.
DBM_INSERT will only insert new entries into the database and will not change an existing
entry with the same key. DBM_REPLACE will replace an existing entry if it has the same key.
A key (and its associated contents) is deleted by `dbm_delete`. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of `dbm_firstkey` and `dbm_nextkey`. `dbm_firstkey` will return the first key in the database. `dbm_nextkey` will return the next key in the database. This code will traverse the data base:

```c
for (key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

dbm_error returns non-zero when an error has occurred reading or writing the database. dbm_clearerr resets the error condition on the named database.

**DIAGNOSTICS**

All functions that return an `int` indicate errors with negative values. A zero return indicates ok. Routines that return a `datum` indicate errors with a null (0) `dptr`. If `dbm_store` called with a `flags` value of `DBM_INSERT` finds an existing entry with the same key it returns 1.

**ERRORS**

The `.pag` file will contain holes so that its apparent size is about four times its actual content. Older UNIX systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp, cat, tp, tar, ar) without filling in the holes.

dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 4096 bytes). Moreover all key/content pairs that hash together must fit on a single block. `dbm_store` will return an error in the event that a disk block fills with inseparable data.

`dbm_delete` does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by `dbm_firstkey` and `dbm_nextkey` depends on a hashing function, not on anything interesting.

**SEE ALSO**

`dbm(3X)`
NAME

nice – set program priority

SYNOPSIS

nice(incr)

DESCRIPTION

This interface is obsoleted by setpriority(2).

The scheduling priority of the process is augmented by incr. Positive priorities get less service than normal. Priority 10 is recommended to users who wish to execute long-running programs without flak from the administration.

Negative increments are ignored except on behalf of the super-user. The priority is limited to the range −20 (most urgent) to 20 (least).

The priority of a process is passed to a child process by fork(2). For a privileged process to return to normal priority from an unknown state, nice should be called successively with arguments −40 (goes to priority −20 because of truncation), 20 (to get to 0), then 0 (to maintain compatibility with previous versions of this call).

SEE ALSO

nice(1), setpriority(2), fork(2), renice(8)
NAME
nlist – get entries from name list

SYNOPSIS
#include <nlist.h>

nlist(filename, nl)
char *filename;
struct nlist nl[];
cc ... -lmld

DESCRIPTION
NOTE: The nlist subroutine has moved from the standard C library to the “mld” library due to the difference in the object file format. Programs that need to use nlist must be linked with the -lmld option.

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of an array of structures containing names, types and values. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to 0. For the structure declaration, see /usr/include/nlist.h.

This subroutine is useful for examining the system name list kept in the file /vmunix. In this way programs can obtain system addresses that are up to date.

SEE ALSO
a.out(5)

DIAGNOSTICS
If the file cannot be found or if it is not a valid namelist -1 is returned; otherwise, the number of unfound namelist entries is returned.

The type entry is set to 0 if the symbol is not found.
NAME
ns_addr, ns_ntoa – Xerox NS(tm) address conversion routines

SYNOPSIS
#include <sys/types.h>
#include <netns/ns.h>
struct ns_addr ns_addr(ep)
char *ep;
char *ns_ntoa(ns)
struct ns_addr ns;

DESCRIPTION
The routine ns_addr interprets character strings representing XNS addresses, returning binary information suitable for use in system calls. ns_ntoa 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 ns_addr. Any fields lacking super-decimal digits will have a trailing “H” appended.

Unfortunately, no universal standard exists for representing XNS addresses. An effort has been made to insure that ns_addr be compatible with most formats in common use. It will first separate an address into 1 to 3 fields using a single delimiter chosen from period (“.”), colon (“:”) or pound-sign (“#”). 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, in which case the field is assumed to be a number in decimal notation with hyphens separating the millennia. 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 there are any super-decimal digits present. It is interpreted as octal if there is a leading “0” and there are no super-octal digits. Otherwise, it is converted as a decimal number.

SEE ALSO
hosts(5), networks(5),

DIAGNOSTICS
None (see ERRORS).

ERRORS
The string returned by ns_ntoa resides in a static memory area.
ns_addr should diagnose improperly formed input, and there should be an unambiguous way to recognize this.
NAME
pause – stop until signal

SYNOPSIS
pause()

DESCRIPTION
pause never returns normally. It is used to give up control while waiting for a signal from
kill(2) or an interval timer, see setitimer(2). Upon termination of a signal handler started dur-
ing a pause, the pause call will return.

RETURN VALUE
Always returns -1.

ERRORS
pause always returns:

EINTR The call was interrupted.

SEE ALSO
kill(2), select(2), sigpause(2)
NAME
perror, sys_errlist, sys_nerr - system error messages

SYNOPSIS
perror(s)
char *s;
int sys_nerr;
char *sys_errlist[];

DESCRIPTION
perror produces a short error message on the standard error file describing the last error
encountered during a call to the system from a C program. First the argument string s is
printed, then a colon, then the message and a new-line. Most usefully, the argument string is
the name of the program which incurred the error. The error number is taken from the exte-

nal variable errno (see intro(2)), which is set when errors occur but not cleared when non-
erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings sys_errlist is pro-
vided; errno can be used as an index in this table to get the message string without the new-
line. sys_nerr is the number of messages provided for in the table; it should be checked
because new error codes may be added to the system before they are added to the table.

SEE ALSO
intro(2), psignal(3)
NAME
perror, gerror, ierrno – get system error messages

SYNOPSIS

subroutine perror (string)
character*(*) string

subroutine gerror (string)
character*(*) string

c character*(*) function gerrno()

definition ierrno()

DESCRIPTION

Perror will write a message to fortran logical unit 0 appropriate to the last detected system error. String will be written preceding the standard error message.

Gerror returns the system error message in character variable string. Gerror may be called either as a subroutine or as a function.

Ierrno will return the error number of the last detected system error. This number is updated only when an error actually occurs. Most routines and I/O statements that might generate such errors return an error code after the call; that value is a more reliable indicator of what caused the error condition.

FILES
/usr/lib/libU77.a

SEE ALSO
intro(2), perror(3)
D. L. Wasley, Introduction to the f77 I/O Library

BUGS

String in the call to perror can be no longer than 127 characters.
The length of the string returned by gerror is determined by the calling program.

NOTES

UNIX system error codes are described in intro(2). The f77 I/O error codes and their meanings are:

100  "error in format"
101  "illegal unit number"
102  "formatted i/o not allowed"
103  "unformatted i/o not allowed"
104  "direct i/o not allowed"
105  "sequential i/o not allowed"
106  "can't backspace file"
107  "off beginning of record"
108  "can't stat file"
109  "no * after repeat count"
110  "off end of record"
111  "truncation failed"
112  "incomprehensible list input"
113  "out of free space"
114  "unit not connected"
115  "invalid data for integer format term"
116  "invalid data for logical format term"
117  "new' file exists"
118  "can't find 'old' file"
119  "opening too many files or unknown system error"
120  "requires seek ability"
121  "illegal argument"
122  "negative repeat count"
123  "illegal operation for unit"
124  "invalid data for d, e, f, or g format term"
NAME
plot: openpl, erase, label, line, circle, arc, move, cont, point, linemod, space, closepl – graphics interface

SYNOPSIS
openpl()
erase()
label(s)
char s[];
line(x1, y1, x2, y2)
circle(x, y, r)
ar(x, y, x0, y0, x1, y1)
moves(x, y)
cont(x, y)
point(x, y)
linemod(s)
char s[];
space(x0, y0, x1, y1)
closepl()

DESCRIPTION
These subroutines generate graphic output in a relatively device-independent manner. See plot(5) for a description of their effect. openpl must be used before any of the others to open the device for writing. closepl flushes the output.

String arguments to label and linemod are null-terminated, and do not contain newlines.

Various flavors of these functions exist for different output devices. They are obtained by the following ld(1) options:

- lplot
  device-independent graphics stream on standard output for plot(1) filters

- l300
  GSI 300 terminal

- l300s
  GSI 300S terminal

- l450
  GSI 450 terminal

- l4013
  Tektronix 4013 terminal

- l4014
  Tektronix 4014 and 4015 terminals with the Enhanced Graphics Module (Use -l4013 for 4014's or 4015's without the Enhanced Graphics Module)

- lplotaed
  AED 512 color graphics terminal

- lplotbg
  BBN bitgraph graphics terminal

- lplotdumb
  Dumb terminals without cursor addressing or line printers

- lplot
  DEC Gigi terminals

- lv10
  DEC v100 terminals

- lplot2648
  Hewlett Packard 2648 graphics terminal

- lplot7221
  Hewlett Packard 7221 graphics terminal

- lplotimagen
  Imagen laser printer (default 240 dots-per-inch resolution).
On many devices, it is necessary to pause after \textit{erase()}, otherwise plotting commands are lost. The pause is normally done by the tty driver if at login time, \textit{tset} found a \textit{df} field in the \textit{termcap}(5) entry for the terminal. If a pause is needed but not automatically being generated, add
\begin{verbatim}
flush(stdout);
sleep(1);
\end{verbatim}
after each \textit{erase()}.

\textbf{SEE ALSO}
plot(5), plot(1G), plot(3F), graph(1G)
NAME
popen, pclose – initiate I/O to/from a process

SYNOPSIS
#include <stdio.h>

FILE *popen(command, type);
char *command, *type;

pclose(stream)
FILE *stream;

DESCRIPTION
The arguments to popen are pointers to null-terminated strings containing respectively a shell command line and an I/O mode, either "r" for reading or "w" for writing. It creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer that can be used (as appropriate) to write to the standard input of the command or read from its standard output.

A stream opened by popen should be closed by pclose, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type "r" command may be used as an input filter, and a type "w" as an output filter.

SEE ALSO
pipe(2), fopen(3S), fclose(3S), system(3), wait(2), sh(1)

DIAGNOSTICS
popen returns a null pointer if files or processes cannot be created, or the shell cannot be accessed.

pclose returns −1 if stream is not associated with a 'popened' command.

ERRORS
Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be forestalled by careful buffer flushing, for instance, with fflush, see fclose(3S).

popen always calls sh, never calls csh.
NAME
printf, fprintf, snprintf – formatted output conversion

SYNOPSIS
#include <stdio.h>

printf(format [, arg ] ...)
char *format;

fprintf(stream, format [, arg ] ...)
FILE *stream;
char *format;

snprintf(s, format [, arg ] ...)
char *s, format;

#include <varargs.h>
_doprnt(format, args, stream)
char *format;
va_list *args;
FILE *stream;

DESCRIPTION
printf places output on the standard output stream stdout. fprintf places output on the named output stream. snprintf places ‘output’ in the string s, followed by the character ‘\0’. It returns the first argument. All of these routines work by calling the internal routine _doprnt, using the variable-length argument facilities of varargs(3).

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive arg printf.

Each conversion specification is introduced by the character %. The remainder of the conversion specification includes in the following order

- Zero or more of following flags:
  - a ‘#’ character specifying that the value should be converted to an “alternate form”. For c, d, s, and u, conversions, this option has no effect. For o conversions, the precision of the number is increased to force the first character of the output string to a zero. For x(X) conversion, a non-zero result has the string 0x(0X) prepended to it. For e, E, f, g, and G, conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point only appears in the results of those conversions if a digit follows the decimal point). For g and G conversions, trailing zeros are not removed from the result as they would otherwise be.
  - a minus sign ‘-’ which specifies left adjustment of the converted value in the indicated field;
  - a ‘+’ character specifying that there should always be a sign placed before the number when using signed conversions.
  - a space specifying that a blank should be left before a positive number during a signed conversion. A ‘+’ overrides a space if both are used.
  - an optional digit string specifying a field width; if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;
bullet an optional period ‘.’ which serves to separate the field width from the next digit string;

bullet an optional digit string specifying a precision which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;

bullet the character l specifying that a following d, o, x, or u corresponds to a long integer arg.

bullet a character which indicates the type of conversion to be applied.

A field width or precision may be ‘*’ instead of a digit string. In this case an integer arg supplies the field width or precision.

The conversion characters and their meanings are

- **dox** The integer arg is converted to decimal, octal, or hexadecimal notation respectively.
- **f** The float or double arg is converted to decimal notation in the style ‘[-]ddd.ddd’ where the number of d’s after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.
- **e** The float or double arg is converted in the style ‘[-]d.dddexe±dd’ where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced.
- **g** The float or double arg is printed in style d, in style f, or in style e, whichever gives full precision in minimum space.
- **c** The character arg is printed.
- **s** Arg is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.
- **u** The unsigned integer arg is converted to decimal and printed (the result will be in the range 0 through MAX_UINT, where MAX_UINT equals 4294967295 on a MIPS R2000).
- **%** Print a ‘%’; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by printf are printed by putc(3S).

**Examples**

To print a date and time in the form ‘Sunday, July 3, 10:02’, where weekday and month are pointers to null-terminated strings:

```c
printf("%s, %s %d, %02d:%02d", weekday, month, day, hour, min);
```

To print π to 5 decimals:

```c
printf("pi = %.5f", 4*atan(1.0));
```

**SEE ALSO**

putc(3S), scanf(3S), ecvt(3S)

**ERRORS**

Very wide fields (>128 characters) fail.
NAME
psignal, sys_siglist – system signal messages

SYNOPSIS
psignal(sig, s)
unsigned sig;
char *s;
char *sys_siglist[];

DESCRIPTION
psignal produces a short message on the standard error file describing the indicated signal.
First the argument string s is printed, then a colon, then the name of the signal and a new-
line. Most usefully, the argument string is the name of the program which incurred the signal.
The signal number should be from among those found in <signal.h>.

To simplify variant formatting of signal names, the vector of message strings sys_siglist is pro-
vided; the signal number can be used as an index in this table to get the signal name without
the newline. The define NSIG defined in <signal.h> is the number of messages provided for
in the table; it should be checked because new signals may be added to the system before they
are added to the table.

SEE ALSO
sigvec(2), perror(3)
publiclib - public domain packages written in Ada

DESCRIPTION

publiclib contains the packages CHARACTER_TYPE and VSTRINGS.

NOTE: These packages are neither supported by nor warranted by MIPS.

CHARACTER_TYPE provided the following character handling functions.
ISLPHA
ISUPPER
ISLOWER
ISDIGIT
ISXDIGIT
ISALNUM
ISSPACE
ISFUNCT
ISPRINT
ISCNTRL
ISAASCII
TOUPPER
TOLOWER
TOASCII

VSTRINGS provides string replacement, searching, concatenation, and other string functions with a simple syntax and the ability to transfer data between its own data representation and the predefined Ada type STRING.

TYPES AND FUNCTIONS

subtype ASCII_INTEGER in TOASCII function

FILES

/usr/vads5/publiclib/*

SEE ALSO

examples, standard, verdirlib
NAME
putc, fputc – write a character to a fortran logical unit

SYNOPSIS
   integer function putc (char)
   character char

   integer function fputc (lunit, char)
   character char

DESCRIPTION
These functions write a character to the file associated with a fortran logical unit bypassing normal fortran I/O. Putc writes to logical unit 6, normally connected to the control terminal output.

The value of each function will be zero unless some error occurred; a system error code otherwise. See perror(3F).

FILES
/usr/lib/libU77.a

SEE ALSO
putc(3S), intro(2), perror(3F)
NAME
putc, putchar, fputc, putw — put character or word on a stream

SYNOPSIS
#include <stdio.h>
int putc(c, stream)
char c;
FILE *stream;
int putchar(c)
int fputc(c, stream)
FILE *stream;
int putw(w, stream)
FILE *stream;

DESCRIPTION
putc appends the character c to the named output stream. It returns the character written.
putchar(c) is defined as putc(c, stdout).

fputc behaves like putc, but is a genuine function rather than a macro.
putw appends word (that is, int) w to the output stream. It returns the word written. putw
neither assumes nor causes special alignment in the file.

SEE ALSO
fopen(3S), fclose(3S), getc(3S), puts(3S), printf(3S), fread(3S)

DIAGNOSTICS
These functions return the constant EOF upon error. Since this is a good integer, ferror(3S)
should be used to detect putw errors.

ERRORS
Because it is implemented as a macro, putc treats a stream argument with side effects impro-
perly. In particular
putc(c, stdout);
doesn’t work sensibly.
Errors can occur long after the call to putc.
NAME
puts, fputs – put a string on a stream

SYNOPSIS
    #include <stdio.h>
    puts(s)
    char *s;
    fputs(s, stream)
    char *s;
    FILE *stream;

DESCRIPTION
    puts copies the null-terminated string s to the standard output stream stdout and appends a
    newline character.
    fputs copies the null-terminated string s to the named output stream.
    Neither routine copies the terminal null character.

SEE ALSO
    fopen(3S), gets(3S), putc(3S), printf(3S), ferror(3S)
    fread(3S) for fwrite

ERRORS
    puts appends a newline, fputs does not, all in the name of backward compatibility.
NAME
qsort – quicker sort

SYNOPSIS
qsort(base, nel, width, compar)
char *base;
int (*compar)();

DESCRIPTION
qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to
the base of the data; the second is the number of elements; the third is the width of an ele-
ment in bytes; the last is the name of the comparison routine to be called with two arguments
which are pointers to the elements being compared. The routine must return an integer less
than, equal to, or greater than 0 according as the first argument is to be considered less than,
equal to, or greater than the second.

SEE ALSO
sort(1)
NAME
qsort – quick sort

SYNOPSIS
subroutine qsort (array, len, isize, compar)
external compar
integer[*2] compar

DESCRIPTION
One dimensional array contains the elements to be sorted. len is the number of elements in
the array. isize is the size of an element, typically -

4 for integer and real
8 for double precision or complex
16 for double complex
(length of character object) for character arrays

Compar is the name of a user supplied integer or integer*2 function that will determine the
sorting order. You must declare compar as external with the "external" statement to be recog-
nized as a function. This function will be called with 2 arguments that will be elements of
array. The function must return -

  negative if arg 1 is considered to precede arg 2
  zero if arg 1 is equivalent to arg 2
  positive if arg 1 is considered to follow arg 2

On return, the elements of array will be sorted.

FILES
/usr/lib/libU77.a

SEE ALSO
qsort(3)
NAME
rand, srand – random number generator

SYNOPSIS
srand(seed)
int seed;
rand()

DESCRIPTION
The newer random(3) should be used in new applications; rand remains for compatibility.

rand uses a multiplicative congruential random number generator with period $2^{32}$ to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$.

The generator is reinitialized by calling srand with 1 as argument. It can be set to a random starting point by calling srand with whatever you like as argument.

SEE ALSO
random(3)
NAME
rand, irand, srand – random number generator

SYNOPSIS
integer iseed, i, irand
double precision s, rand

call srand(iseed)
i = irand()
x = rand()

DESCRIPTION
Irand generates successive pseudo-random integers in the range from 0 to 2**15–1. rand generates pseudo-random numbers distributed in [−, 1.0]. Srand uses its integer argument to reinitialize the seed for successive invocations of irand and rand.

SEE ALSO
rand(3C).
NAME
random, srandom, initstate, setstate – better random number generator; routines for changing
generators

SYNOPSIS
long random()

srandom(seed)
int seed;

char *initstate(seed, state, n)
unsigned seed;
char *state;
int n;
char *setstate(state)
char *state;

DESCRIPTION
random uses a non-linear additive feedback random number generator employing a default
table of size 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 very large, approximately
$16 	imes (2^{31}-1)$.

random/srandom have (almost) the same calling sequence and initialization properties as
rand/srand. The difference is that rand(3) produces a much less random sequence – in fact,
the low dozen bits generated by rand go through a cyclic pattern. All the bits generated by
random are usable. For example, “random()&01” will produce a random binary value.

Unlike srand, srandom does not return the old seed; the reason for this is that the amount of
state information used is much more than a single word. (Two other routines are provided to
deal with restarting/changing random number generators). Like rand(3), however, random
will by default produce a sequence of numbers that can be duplicated by calling srandom with
I as the seed.

The initstate routine allows a state array, passed in as an argument, to be initialized for future
use. The size of the state array (in bytes) is used by initstate to decide how sophisticated a
random number generator it should use – the more state, the better the random numbers will
be. (Current “optimal” values for the amount of state information are 8, 32, 64, 128, and 256
bytes; other amounts will be rounded down to the nearest known amount. Using less than 8
bytes will cause an error). The seed for the initialization (which specifies a starting point for
the random number sequence, and provides for restarting at the same point) is also an argument.
Initstate returns a pointer to the previous state information array.

Once a state has been initialized, the setstate routine provides for rapid switching between
states. Setstate returns a pointer to the previous state array; its argument state array is used for
further random number generation until the next call to initstate or setstate.

Once a state array has been initialized, it may be restarted at a different point either by calling
initstate (with the desired seed, the state array, and its size) or by calling both setstate (with the
state array) and srandom (with the desired seed). The advantage of calling both setstate and
srandom is that the size of the state array does not have to be remembered after it is initial-
ized.

With 256 bytes of state information, the period of the random number generator is greater
than $2^{60}$, which should be sufficient for most purposes.

AUTHOR
Earl T. Cohen
DIAGNOSTICS
If `initstate` is called with less than 8 bytes of state information, or if `setstate` detects that the state information has been garbled, error messages are printed on the standard error output.

SEE ALSO
`rand(3)`

ERRORS
About 2/3 the speed of `rand(3C)`. 
NAME
ranhashinit, ranhash, ranlookup – access routine for the symbol table definition file in archives

SYNOPSIS
#include <ar.h>
int ranhashinit(pran, pstr, size)
struct ranlib *pran;
char *pstr;
int size;
ranhash(name)
char *name;
struct ranlib *ranhash(name)
char *name;

DESCRIPTION
Ranhashinit initializes static information for future use by ranhash and ranlookup. Pran points to an array of ranlib structures. Pstr points to the corresponding ranlib string table (these are only used by ranlookup). Size is the size of the hash table and should be a power of 2. If the size isn't a power of 2, a 1 is returned; otherwise, a 0 is returned.

Ranhash returns a hash number given a name. It uses a multiplicative hashing algorithm and the size argument to ranhashinit.

Ranlookup looks up name in the ranlib table specified by ranhashinit. It uses the ranhash routine as a starting point. Then, it does a rehash from there. This routine returns a pointer to a valid ranlib entry on a match. If no matches are found (the "emptiness" can be inferred if the ran_off field is zero), the empty ranlib structure hash table should be sparse. This routine does not expect to run out of places to look in the table. For example, if you collide on all entries in the table, an error is printed to stderr and a zero is returned.

AUTHOR
Mark I. Himelstein

SEE ALSO
ar(1), ar.h(5).
NAME
rcmd, resvport, ruserok — routines for returning a stream to a remote command

SYNOPSIS
rem = rcmd(ahost, inport, locuser, remuser, cmd, fd2p);
char *ahost;
int inport;
char *locuser, *remuser, *cmd;
int *fd2p;
s = resvport(port);
int *port;
ruserok(rhost, superuser, ruser, luser);
char *rhost;
int superuser;
char *ruser, *luser;

DESCRIPTION
rcmd is a routine used by the super-user to execute a command on a remote machine using an
authentication scheme based on reserved port numbers. resvport is a routine which returns a
descriptor to a socket with an address in the privileged port space. ruserok is a routine used
by servers to authenticate clients requesting service with rcmd. All three functions are present
in the same file and are used by the rsd8(8C) server (among others).

rcmd looks up the host *ahost using gethostbyname(3N), returning −1 if the host does not
exist. Otherwise *ahost is set to the standard name of the host and a connection is established
to a server residing at the well-known Internet port inport.

If the connection succeeds, a socket in the Internet domain of type SOCK_STREAM is
returned to the caller, and given to the remote command as stdin and stdout. If fd2p is non-
zero, then an auxiliary channel to a control process will be set up, and a descriptor for it will
be placed in *fd2p. The control process will return diagnostic output from the command (unit 2)
on this channel, and will also accept bytes on this channel as being UNIX signal numbers,
to be forwarded to the process group of the command. If fd2p is 0, then the stderr (unit 2 of
the remote command) will be made the same as the stdout and no provision is made for sending
arbitrary signals to the remote process, although you may be able to get its attention by
using out-of-band data.

The protocol is described in detail in rsd8(8C).

The resvport routine is used to obtain a socket with a privileged address bound to it. This
socket is suitable for use by rcmd and several other routines. Privileged Internet ports are
those in the range 0 to 1023. Only the super-user is allowed to bind an address of this sort to
a socket.

ruserok takes a remote host’s name, as returned by a gethostbyaddr(3N) routine, two user
names and a flag indicating whether the local user’s name is that of the super-user. It then
checks the files /etc/hosts.equiv and, possibly, .rhosts in the user’s home directory to see if the
request for service is allowed. A 0 is returned if the machine name is listed in the
“hosts.equiv” file, or the host and remote user name are found in the “.rhosts” file; otherwise
ruserok returns −1. If the superuser flag is 1, the checking of the “host.equiv” file is bypassed.
If the local domain (as obtained from gethostname(2)) is the same as the remote domain, only
the machine name need be specified.

SEE ALSO
rlogin(1C), rsh(1C), intro(2), rexec(3), hosts(5), rexecd(8C), rlogin(8C), rsd8(8C)
DIAGNOSTICS

rcmd returns a valid socket descriptor on success. It returns -1 on error and prints a diagnostic message on the standard error.

resvport returns a valid, bound socket descriptor on success. It returns -1 on error with the global value errno set according to the reason for failure. The error code EAGAIN is overloaded to mean "All network ports in use."
NAME
re_comp, re_exec - regular expression handler

SYNOPSIS
char *re_comp(s)
char *s;
re_exec(s)
char **s;

DESCRIPTION
re_comp compiles a string into an internal form suitable for pattern matching. re_exec checks the argument string against the last string passed to re_comp.

re_comp returns 0 if the string s was compiled successfully; otherwise a string containing an error message is returned. If re_comp is passed 0 or a null string, it returns without changing the currently compiled regular expression.

re_exec returns 1 if the string s matches the last compiled regular expression, 0 if the string s failed to match the last compiled regular expression, and -1 if the compiled regular expression was invalid (indicating an internal error).

The strings passed to both re_comp and re_exec may have trailing or embedded newline characters; they are terminated by nulls. The regular expressions recognized are described in the manual entry for ed(1), given the above difference.

SEE ALSO
ed(1), ex(1), egrep(1), fgrep(1), grep(1)

DIAGNOSTICS
re_exec returns -1 for an internal error.
re_comp returns one of the following strings if an error occurs:

No previous regular expression,
Regular expression too long,
unmatched \, missing },
too many \(\) pairs,
unmatched \).
NAME
    res_mkquery, res_send, res_init, dn_comp, dn_expand – resolver routines

SYNOPSIS
    #include <sys/types.h>
    #include <netinet/in.h>
    #include <arpa/nameser.h>
    #include <resolv.h>

    res_mkquery(op, dname, class, type, data, datalen, newrr, buf, buflen)
    int op;
    char *dname;
    int class, type;
    char *data;
    int datalen;
    struct rrec *newrr;
    char *buf;
    int buflen;

    res_send(msg, msglen, answer, anslen)
    char *msg;
    int msglen;
    char *answer;
    int anslen;

    res_init()

    dn_comp(exp_dn, comp_dn, length, dnptrs, lastdnptr)
    char *exp_dn, *comp_dn;
    int length;
    char **dnptrs, **lastdnptr;

    dn_expand(msg, comorig, comp_dn, exp_dn, length)
    int length;

DESCRIPTION
    These routines are used for making, sending and interpreting packets to Internet domain name servers. Global information that is used by the resolver routines is kept in the variable _res. Most of the values have reasonable defaults and can be ignored. Options stored in _res.options are defined in resolv.h and are as follows. Options are a simple bit mask and are or’ed in to enable.

RES_INIT
    True if the initial name server address and default domain name are initialized (i.e., res_init has been called).

RES_DEBUG
    Print debugging messages.

RES_AAAONLY
    Accept authoritative answers only. res_send will continue until it finds an authoritative answer or finds an error. Currently this is not implemented.

RES_USEVC
    Use TCP connections for queries instead of UDP.

RES_STAYOPEN
    Used with RES_USEVC to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal
mode used.

RES_IGNTC
Unused currently (ignore truncation errors, i.e., don’t retry with TCP).

RES_RECURSE
Set the recursion desired bit in queries. This is the default. (res_send does not do
iterative queries and expects the name server to handle recursion.)

RES_DEFNAMES
Append the default domain name to single label queries. This is the default.

res_init
reads the initialization file to get the default domain name and the Internet address of the
initial hosts running the name server. If this line does not exist, the host running the resolver is
tried. res_mkquery makes a standard query message and places it in buf. res_mkquery will
return the size of the query or -1 if the query is larger than bufsize. op is usually QUERY but
can be any of the query types defined in nameser.h. dname is the domain name. If dname
consists of a single label and the RES_DEFNAMES flag is enabled (the default), dname will be
appended with the current domain name. The current domain name is defined in a system file
and can be overridden by the environment variable LOCALDOMAIN. newrr is currently
unused but is intended for making update messages.

res_send sends a query to name servers and returns an answer. It will call res_init if RES_INIT
is not set, send the query to the local name server, and handle timeouts and retries. The
length of the message is returned or -1 if there were errors.

dn_expand expands the compressed domain name comp_dn to a full domain name. Expanded
names are converted to upper case. msg is a pointer to the beginning of the message, exp_dn
is a pointer to a buffer of size length for the result. The size of compressed name is returned
or -1 if there was an error.

dn_comp compresses the domain name exp_dn and stores it in comp_dn. The size of the
compressed name is returned or -1 if there were errors. length is the size of the comp_dn.
dnprts is a list of pointers to previously compressed names in the current message. The first
pointer points to the beginning of the message and the list ends with NULL. lastdnptr is a
pointer to the end of the array pointed to dnptrs. A side effect is to update the list of pointers
for labels inserted into the message by dn_comp as the name is compressed. If dnptr is NULL,
we don’t try to compress names. If lastdnptr is NULL, we don’t update the list.

FILES
/etc/resolv.conf see resolver(5)

SEE ALSO
named(8), resolver(5), RFC882, RFC883, RFC973, RFC974, SMM:11 Name Server Opera-
tions Guide for BIND
NAME
rex – remote execution protocol

SYNOPSIS
#include <sys/ioctl.h>
#include <rpcsvc/rex.h>

DESCRIPTION
This server will execute commands remotely. The working directory and environment of
the command can be specified, and the standard input and output of the command can be arbi-
trarily redirected. An option is provided for interactive I/O for programs that expect to be
running on terminals. Note that this service is only provided with the TCP transport.

RPC INFO
program number:
  REXPROG

xdr routines:
  int xdr_rex_start(xdrs, start);
  XDR *xdrs;
  struct rex_start *start;

  int xdr_rex_result(xdrs, result);
  XDR *xdrs;
  struct rex_result *result;

  int xdr_rex_ttymode(xdrs, mode);
  XDR *xdrs;
  struct rex_ttymode *mode;

  int xdr_rex_ttysize(xdrs, size);
  XDR *xdrs;
  struct ttysize *size;

procs:
  REXPROC_START
  Takes rex_start structure, starts a command executing,
  and returns a rex_result structure.

  REXPROC_WAIT
  Takes no arguments, waits for a command to finish executing,
  and returns a rex_result structure.

  REXPROC_MODES
  Takes a rex_ttymode structure, and sends the tty modes.

  REXPROC_WINCH
  Takes a ttysize structure, and sends window size information.

versions:
  REXVERS_ORIG
    Original version

structures:
#define REX_INTERACTIVE 1 /* Interactive mode */

struct rex_start {
  char **rst_cmd; /* list of command and args */
  char **rst_host; /* working directory host name */
  char **rst_fstname; /* working directory file system name */
  char **rst_dirwithin; /* working directory within file system */
  char **rst_env; /* list of environment */
  u_short rst_port0; /* port for stdin */
  u_short rst_port1; /* port for stdin */
u_short rst_port2;
  u_long rst_flags;
};

struct rex_result {
  int rlt_stat;
  char *rlt_message;
};

struct rex_ttymode {
  struct sgttyb basic;
  struct tchars more;
  struct itchars yetmore;
  u_long andmore;
};

SEE ALSO
on(1C), rexd(8C)
NAME
reexec – return stream to a remote command

SYNOPSIS
rem = reexec(ahost, inport, user, passwd, cmd, fd2p);
char **ahost;
int inport;
char **user, **passwd, **cmd;
int *fd2p;

DESCRIPTION
reexec looks up the host *ahost using gethostbyname(3N), returning −1 if the host does not
eexist. Otherwise *ahost is set to the standard name of the host. If a username and password
are both specified, then these are used to authenticate to the foreign host; otherwise the
environment and then the user’s .netrc file in his home directory are searched for appropriate
information. If all this fails, the user is prompted for the information.

The port inport specifies which well-known DARPA Internet port to use for the connection;
the call “getservbyname("exec", "tcp")” (see getservent(3N)) will return a pointer to a structure,
which contains the necessary port. The protocol for connection is described in detail in
reexecd(8C).

If the connection succeeds, a socket in the Internet domain of type SOCK_STREAM is
returned to the caller, and given to the remote command as stdin and stdout. If fd2p is non-
zero, then an auxiliary channel to a control process will be setup, and a descriptor for it will
be placed in *fd2p. The control process will return diagnostic output from the command (unit
2) on this channel, and will also accept bytes on this channel as being UNIX signal numbers,
to be forwarded to the process group of the command. The diagnostic information returned
does not include remote authorization failure, as the secondary connection is set up after
authorization has been verified. If fd2p is 0, then the stderr (unit 2 of the remote command)
will be made the same as the stdout and no provision is made for sending arbitrary signals to
the remote process, although you may be able to get its attention by using out-of-band data.

SEE ALSO
rcmd(3), reexecd(8C)
NAME
rnusers, rusers – return information about users on remote machines

SYNOPSIS
#include <rpcsvc/rusers.h>

rnusers(host)
    char *host

rusers(host, up)
    char *host
    struct utmpidlearr *up;

DESCRIPTION
Rnusers returns the number of users logged on to host (-1 if it cannot determine that number).

rusers fills the utmpidlearr structure with data about host, and returns 0 if successful. The relevant structures are:

struct utmparr {
    struct utmp *uta_arr;
    int uta_cnt
};

struct utmpidle {
    struct utmp ui_utmp;
    unsigned ui_idle;
};

struct utmpidlearr {
    struct utmpidle *ui_arr;
    int ui_arr;
};

RPC INFO
program number:
    RUSERSPROG

xdr routines:
    int xdr_utmp(xdrs, up)
        XDR *xdrs;
        struct utmp *up;
    int xdr_utmpidle(xdrs, ui);
        XDR *xdrs;
        struct utmpidle *ui;
    int xdr_utmpptr(xdrs, up);
        XDR *xdrs;
        struct utmpidle *up;
    int xdr_utmpidleptr(xdrs, up);
        XDR *xdrs;
        struct utmpidle **up;
    int xdr_uttmparr(xdrs, up);
        XDR *xdrs;
        struct utmparr *up;
    int xdr_utmparrarr(xdrs, up);
        XDR *xdrs;
        struct utmpidlearr *up;

procs:
RUSERSPROC_NUM
   No arguments, returns number of users as an unsigned long.
RUSERSPROC_NAMES
   No arguments, returns utmparr or utmpidlearr, depending on version number.
RUSERSPROC_ALLNAMES
   No arguments, returns utmparr or utmpidlearr, depending on version number.
   Returns listing even for utmp entries satisfying nonuser() in utmp.h.

versions:
   RUSERSVERS_ORIG
   RUSERSVERS_IDLE

structures:

SEE ALSO
   rusers(1C)
NAME
rpc – library routines for remote procedure calls

DESCRIPTION
These routines allow C programs to make procedure calls on other machines across the net-
work. First, the client calls a procedure to send a data packet to the server. Upon receipt of
the packet, the server calls a dispatch routine to perform the requested service, and then
sends back a reply. Finally, the procedure call returns to the client.

FUNCTIONS
auth_destroy()
authnone_create()
authunix_create()
authunix_create_default()
callrpc()
cnt_broadcast()
cnt_call()
cnt_destroy()
cnt_freeres()
cnt_geterr()
cnt_pcreateerror()
cnt_errno()
cnt_serrno()
cnt_serror()
cntraw_create()
cnttcp_create()
cntudp_create()
get_myaddress()
pmap_getmaps()
pmap_getport()
pmap_rmtcall()
pmap_set()
pmap_unset()
registerrpc()
rpc_createerr
svc_destroy()
svc_fds
svc_freeargs()
svc_getargs()
svc_getcaller()
svc_getreq()
svc_register()
svc_run()
svc_sendreply()
svc_unregister()
svcerr_auth()
svcerr_decode()
svcerr_noproc()
svcerr_noprog()
svcerr_progers()
svcerr_systemerr()
svcerr_weakauth()
svcraw_create()
destroy authentication information handle
return RPC authentication handle with no checking
return RPC authentication handle with UNIX permissions
return default UNIX authentication handle
call remote procedure, given [prognum, versnum, procnm]
broadcast remote procedure call everywhere
call remote procedure associated with client handle
destroy client’s RPC handle
free data allocated by RPC/XDR system when decoding results
copy error information from client handle to error structure
print message to stderr about why client handle creation failed
print message to stderr corresponding to condition given
print message to stderr about why RPC call failed
print message to a string corresponding to condition given
print message to a string
create toy RPC client for simulation
create RPC client using TCP transport
create RPC client using UDP transport
get the machine’s IP address
return list of RPC program-to-port mappings
return port number on which waits supporting service
instructs portmapper to make an RPC call
establish mapping between [prognum, versnum, procnm] and port
destroy mapping between [prognum, versnum, procnm] and port
register procedure with RPC service package
global variable indicating reason why client creation failed
destroy RPC service transport handle
global variable with RPC service file descriptor mask
free data allocated by RPC/XDR system when decoding arguments
decodes the arguments of an RPC request
get the network address of the caller of a procedure
returns when all associated sockets have been serviced
associates progmnum and versnum with service dispatch procedure
wait for RPC requests to arrive and call appropriate service
send back results of a remote procedure call
remove mapping of [prognum, versnum] to dispatch routines
called when refusing service because of authentication error
called when service cannot decode its parameters
called when service hasn’t implemented the desired procedure
called when program is not registered with RPC package
called when version is not registered with RPC package
called when service detects system error
called when refusing service because of insufficient authentication
creates a toy RPC service transport for testing
svctcp_create() creates an RPC service based on TCP transport
svcudp_create() creates an RPC service based on UDP transport
xdr_accepted_reply() generates RPC-style replies without using RPC package
xdr_authunix_parms() generates UNIX credentials without using RPC package
xdr_callhdr() generates RPC-style headers without using RPC package
xdr_callmsg() generates RPC-style messages without using RPC package
xdrOpaque_auth() describes RPC messages, externally
describes parameters for portmap procedures, externally
xdr_pmap() describes a list of port mappings, externally
xdr_pmaplist() generates RPC-style rejections without using RPC package
xdr_rejected_reply() generates RPC-style replies without using RPC package
xdr_replymsg() registers RPC service transport with RPC package
xprt_register() unregisters RPC service transport from RPC package
xprt_unregister()
NAME
rquota – implement quotas on remote machines

SYNOPSIS
#include <rpcsvc/rquota.h>

RPC INFO
program number:
RQUOTA PROG

xdr routines:
  xdr_getquota_args(xdrs, gqa);
      XDR *xdrs;
      struct getquota_args *gqa;
  xdr_getquota_rslt(xdrs, gqr);
      XDR *xdrs;
      struct getquota_rslt *gqr;
  xdr_rquota(xdrs, rq);
      XDR *xdrs;
      struct rquota *rq;

procs:
RQUOTA PROG_GETQUOTA
RQUOTA PROG_GETACTIVEQUOTA
    Arguments of struct getquota_args.
    Returns struct getquota_rslt.
    Uses UNIX authentication.
    Returns quota only on filesystems with quota active.

versions:
RQUOTA VERS ORIG

structures:
struct getquota_args {
    char *gqa_pathp;  /* path to filesystem of interest */
    int gqa_uid;    /* inquire about quota for uid */
};
/*
 * remote quota structure
 */
struct rquota {
    int rq_bsize;      /* block size for block counts */
    bool_t rq_active;  /* indicates whether quota is active */
    u_long rq_bhardlimit;   /* absolute limit on disk blks alloc */
    u_long rq_bsoftlimit;  /* preferred limit on disk blks */
    u_long rq_curblocks;   /* current block count */
    u_long rq_fhardlimit;  /* absolute limit on allocated files */
    u_long rq_fsoftlimit;  /* preferred file limit */
    u_long rq_curfiles;    /* current # allocated files */
    u_long rq_btimeleft;   /* time left for excessive disk use */
    u_long rq_ftimeleft;   /* time left for excessive files */
};
enum gqr_status {
    Q_OK = 1,     /* quota returned */
    Q_NOQUOTA = 2,  /* no quota for uid */
    Q_EPERM = 3  /* no permission to access quota */
};
};
struct getquota_rslt {
    enum gqr_status gqr_status;  /* discriminant */
    struct rquota gqr_rquota;    /* valid if status == Q_OK */
};

SEE ALSO
    quota(1), quotactl(2)
NAME
rwall - write to specified remote machines

SYNOPSIS
#include <rproc/rwall.h>
rwall(host, msg);
    char *host, *msg;

DESCRIPTION
rwall causes host to print the string msg to all its users. It returns 0 if successful.

RPC INFO
program number:
    WALLPROG
procs:
    WALLPROC_WALL
        Takes string as argument (wrapstring), returns no arguments.
        Executes wall on remote host with string.
versions:
    RSTATVERS_ORIG

SEE ALSO
rwall(1), shutdown(8), rwalld(8C)
NAME
scandir, alphasort - scan a directory

SYNOPSIS
#include <sys/types.h>
#include <sys/dir.h>

scandir(dirname, namelist, select, compar)
char *dirname;
struct direct **(namelist[]);
int (*select)();
int (*compar)();
alphasort(d1, d2)
struct direct **d1, **d2;

DESCRIPTION
scandir reads the directory dirname and builds an array of pointers to directory entries using malloc(3). It returns the number of entries in the array and a pointer to the array through namelist.

The select parameter is a pointer to a user supplied subroutine which is called by scandir to select which entries are to be included in the array. The select routine is passed a pointer to a directory entry and should return a non-zero value if the directory entry is to be included in the array. If select is null, then all the directory entries will be included.

The compar parameter is a pointer to a user supplied subroutine which is passed to qsort(3) to sort the completed array. If this pointer is null, the array is not sorted. alphasort is a routine which can be used for the compar parameter to sort the array alphabetically.

The memory allocated for the array can be deallocated with free (see malloc(3)) by freeing each pointer in the array and the array itself.

SEE ALSO
directory(3), malloc(3), qsort(3), dir(5)

DIAGNOSTICS
Returns -1 if the directory cannot be opened for reading or if malloc(3) cannot allocate enough memory to hold all the data structures.
NAME

cscanf, fscanf, sscanf – formatted input conversion

SYNOPSIS

#include <stdio.h>

scanf(format [ , pointer ] . . . )
char *format;

fscanf(stream, format [ , pointer ] . . . )
FILE *stream;
char *format;

sscanf(s, format [ , pointer ] . . . )
char *s, *format;

DESCRIPTION

scanf reads from the standard input stream stdin. fscanf reads from the named input stream. sscanf reads from the character string s. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string format, described below, and a set of pointer arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines, which match optional white space in the input.
2. An ordinary character (not %) which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

% a single '%' is expected in the input at this point; no assignment is done.
d a decimal integer is expected; the corresponding argument should be an integer pointer.
o an octal integer is expected; the corresponding argument should be an integer pointer.
x a hexadecimal integer is expected; the corresponding argument should be an integer pointer.
s a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating '\0', which will be added. The input field is terminated by a space character or a newline.
e a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try '%s'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.
e a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a float. The input format for floating point numbers is an optionally signed string of digits possibly containing a decimal point, followed by an optional exponent field consisting of an E or e followed by
an optionally signed integer.

[ indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters d, o and x may be capitalized or preceded by l to indicate that a pointer to long rather than to int is in the argument list. Similarly, the conversion characters e or f may be capitalized or preceded by l to indicate a pointer to double rather than to float.

The conversion characters d, o and x may be preceded by h to indicate a pointer to short rather than to int.

The scanf functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant EOF is returned upon end of input; note that this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```c
int i; float x; char name[50];
scanf("%d%f%s", &i, &x, name);
```

with the input line

```
25 54.32E−1 thompson
```

will assign to i the value 25, x the value 5.432, and name will contain 'thompson\0'. Or,

```c
int i; float x; char name[50];
scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input

```
56789 0123 56a72
```

will assign 56 to i, 789.0 to x, skip '0123', and place the string '56\0' in name. The next call to getchar will return 'a'.

SEE ALSO

atof(3), getc(3S), printf(3S)

DIAGNOSTICS

The scanf functions return EOF on end of input, and a short count for missing or illegal data items.

ERRORS

The success of literal matches and suppressed assignments is not directly determinable.
NAME
setbuf, setbuffer, setlinebuf – assign buffering to a stream

SYNOPSIS
#include <stdio.h>
setbuf(stream, buf)
FILE *stream;
char *buf;
setbuffer(stream, buf, size)
FILE *stream;
char *buf;
int size;
setlinebuf(stream)
FILE *stream;

DESCRIPTION
The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a newline is encountered or input is read from stdin. flush (see fclose(3S)) may be used to force the block out early. Normally all files are block buffered. A buffer is obtained from malloc(3) upon the first getc or putc(3S) on the file. If the standard stream stdout refers to a terminal it is line buffered. The standard stream stderr is always unbuffered.

setbuf is used after a stream has been opened but before it is read or written. The character array buf is used instead of an automatically allocated buffer. If buf is the constant pointer NULL, input/output will be completely unbuffered. A manifest constant BUFSIZ tells how big an array is needed:

char buf[BUFSIZ];

setbuffer, an alternate form of setbuf, is used after a stream has been opened but before it is read or written. The character array buf whose size is determined by the size argument is used instead of an automatically allocated buffer. If buf is the constant pointer NULL, input/output will be completely unbuffered.

setlinebuf is used to change stdout or stderr from block buffered or unbuffered to line buffered. Unlike setbuf and setbuffer it can be used at any time that the file descriptor is active.

A file can be changed from unbuffered or line buffered to block buffered by using freopen (see fopen(3S)). A file can be changed from block buffered or line buffered to unbuffered by using freopen followed by setbuf with a buffer argument of NULL.

SEE ALSO
fopen(3S), getc(3S), putc(3S), malloc(3), fclose(3S), puts(3S), printf(3S), fread(3S)

ERRORS
The standard error stream should be line buffered by default.
The setbuffer and setlinebuf functions are not portable to non-4.2BSD versions of UNIX. On 4.2BSD and 4.3BSD systems, setbuf always uses a suboptimal buffer size and should be avoided. setbuffer is not usually needed as the default file I/O buffer sizes are optimal.
NAME

setjmp, longjmp — non-local goto

SYNOPSIS

#include <setjmp.h>

setjmp(env)
jmp_buf env;

longjmp(env, val)
jmp_buf env;
_setjmp(env)
jmp_buf env;
_longjmp(env, val)
jmp_buf env;

DESCRIPTION

These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

setjmp saves its stack environment in env for later use by longjmp. It returns value 0.

longjmp restores the environment saved by the last call of setjmp. It then returns in such a way that execution continues as if the call of setjmp had just returned the value val to the function that invoked setjmp, which must not itself have returned in the interim. All accessible data have values as of the time longjmp was called.

setjmp and longjmp save and restore the signal mask sigmask(2), while _setjmp and _longjmp manipulate only the C stack and registers.

ERRORS

If the contents of the jmp_buf are corrupted, or correspond to an environment that has already returned, longjmp calls the routine longjmperror. If longjmperror returns the program is aborted. The default version of longjmperror prints the message “longjmp botch” to standard error and returns. User programs wishing to exit more gracefully can write their own versions of longjmperror.

SEE ALSO

sigvec(2), sigstack(2), signal(3)

BUGS

The System V version of longjmp() will turn a return value of 0 into a 1, whereas the BSD version always returns the value requested. A number of programs in BSD systems rely on the current behavior.
NAME
  setuid, seteuid, setruid, setgid, setegid, setrgid – set user and group ID

SYNOPSIS
  #include <sys/types.h>
  setuid(uid)
  seteuid(euid)
  setruid(ruid)
  uid_t uid, euid, ruid;
  setgid(gid)
  setegid(egid)
  setrgid(rgid)
  gid_t gid, egid, rgid;

DESCRIPTION
  setuid (setgid) sets both the real and effective user ID (group ID) of the current process to as specified.
  seteuid (setegid) sets the effective user ID (group ID) of the current process.
  setruid (setrgid) sets the real user ID (group ID) of the current process.
  These calls are only permitted to the super-user or if the argument is the real or effective ID.

SEE ALSO
  seteuid(2), setrgid(2), getuid(2), getgid(2)

DIAGNOSTICS
  Zero is returned if the user (group) ID is set; −1 is returned otherwise.
NAME
gethostsex - get the byte sex of the host machine
swap_*() - swap the sex of the specified structure

SYNOPSIS
#include <sex.h>
#include <filehdr.h>
#include <authhdr.h>
#include <scnhdr.h>
#include <sym.h>
#include <symconst.h>
#include <cmplrs/stsupport.h>
#include <reloc.h>
#include <ar.h>

int gethostsex()
long swap_word(word)
long word;
short swap_half(half)
short half;
void swap_filehdr(pfilehdr, destsex)
FILEHDR *pfilehdr;
long destsex;
void swap_authhdr(paouthdr, destsex)
AOUTHDR *paouthdr;
long destsex;
void swap_scnhdr(pscnhdr, destsex)
SCNHDR *pscnhdr;
long destsex;
void swap_hdr(phdr, destsex)
PHDRR phdr;
long destsex;
void swap_fd(pfd, count, destsex)
pFDR pfd;
long count;
long destsex;
void swap_fi(pfi, count, destsex)
pFTT pfi;
long count;
long destsex;
void swap_sym(psym, count, destsex)
pSYMNR psym;
long count;
long destsex;
void swap_ext(pext, count, destsex)
pEXTR pext;
long count;
long destsex;
void swap_pd(ppd, count, destsex)
pPDR pppd;
long count;
long destsex;
void swap_dn(pdn, count, destsex)
pRNDXR pdn;
long count;
long destsex;
void swap_opt(popt, count, destsex)
pOPTR popt;
long count;
long destsex;
void swap_aux(paux, type, destsex)
pAUXU paux;
long type;
long destsex;
void swap_reloc(preloc, count, destsex)
struct reloc *preloc;
long count;
long destsex;
void swap_ranlib(pranlib, count, destsex)
struct ranlib *pranlib;
long count;
long destsex;

DESCRIPTION
To use these routines, the library libmld.a must be loaded.

Gethostsex returns one of two constants BIGENDIAN or LITTLEENDIAN for the sex of the host machine. These constants are in sex.h.

All swap_* routines that swap headers take a pointer to a header structure to change the byte's sex. The destsex argument lets the swap routines decide whether to swap bitfields before or after swapping the words they occur in. If destsex equals the hostsex of the machine you are running on, the flip happens before the swap; otherwise, the flip happens after the swap. Although not all routines swap structures containing bitfields, the destsex is required in the anticipation of future need.

The swap_aux routine takes a pointer to an aux entry and a type, which is a ST_AUX_* constant in cmplrs/stsupport.h. The constant specifies the type of the aux entry to change the sex of. All other swap_* routines are passed a pointer to an array of structures and a count of structures to change the byte sex of. The routines swap_word and swap_half are macros declared in sex.h. Only the include files necessary to describe the structures being swapped need be included.

AUTHOR
Kevin Enderby
NAME
siginterrupt – allow signals to interrupt system calls

SYNOPSIS
siginterrupt(sig, flag);
int sig, flag;

DESCRIPTION
siginterrupt is used to change the system call restart behavior when a system call is interrupted
by the specified signal. If the flag is false (0), then system calls will be restarted if they are
interrupted by the specified signal and no data has been transferred yet. System call restart is
the default behavior on 4.2 BSD.

If the flag is true (1), then restarting of system calls is disabled. If a system call is interrupted
by the specified signal and no data has been transferred, the system call will return -1 with
errno set to EINTR. Interrupted system calls that have started transferring data will return the
amount of data actually transferred. System call interrupt is the signal behavior found on 4.1
BSD and AT&T System V UNIX systems.

Note that the new 4.2 BSD signal handling semantics are not altered in any other way. Most
notably, signal handlers always remain installed until explicitly changed by a subsequent
sigvec(2) call, and the signal mask operates as documented in sigvec(2). Programs may switch
between restartable and interruptible system call operation as often as desired in the execution
of a program.

Issuing a siginterrupt(3) call during the execution of a signal handler will cause the new action
to take place on the next signal to be caught.

NOTES
This library routine uses an extension of the sigvec(2) system call that is not available in
4.2BSD, hence it should not be used if backward compatibility is needed.

RETURN VALUE
A 0 value indicates that the call succeeded. A -1 value indicates that an invalid signal number
has been supplied.

SEE ALSO
sigvec(2), sigblock(2), sigpause(2), sigsetmask(2).
NAME
signal – simplified software signal facilities

SYNOPSIS
#include <signal.h>

(ssignal(sig, func))
int (sfunc);

DESCRIPTION
signal is a simplified interface to the more general sigvec(2) facility.

A signal is generated by some abnormal event, initiated by a user at a terminal (quit, interrupt, stop), by a program error (bus error, etc.), by request of another program (kill), or when a process is stopped because it wishes to access its control terminal while in the background (see tty(4)). Signals are optionally generated when a process resumes after being stopped, when the status of child processes changes, or when input is ready at the control terminal. Most signals cause termination of the receiving process if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the SIGKILL and SIGSTOP signals, the signal call allows signals either to be ignored or to cause an interrupt to a specified location. The following is a list of all signals with names as in the include file <signal.h>:

SIGHUP 1 hangup
SIGINT 2 interrupt
SIGQUIT 3* quit
SIGILL 4* illegal instruction
SIGTRAP 5* trace trap
SIGIOT 6* IOT instruction
SIGEMT 7* EMT instruction
SIGFPE 8* floating point exception
SIGKILL 9 kill (cannot be caught or ignored)
SIGBUS 10* bus error
SIGSEGV 11* segmentation violation
SIGSYS 12* bad argument to system call
SIGPIPE 13 write on a pipe with no one to read it
SIGALRM 14 alarm clock
SIGTERM 15 software termination signal
SIGURG1 16* urgent condition present on socket
SIGSTOP 17† stop (cannot be caught or ignored)
SIGTSTP 18† stop signal generated from keyboard
SIGCONT 19* continue after stop
SIGCHLD 20* child status has changed
SIGTIN 21† background read attempted from control terminal
SIGTTOU 22† background write attempted to control terminal
SIGIO 23* i/o is possible on a descriptor (see fcntl(2))
SIGXCPU 24 cpu time limit exceeded (see setrlimit(2))
SIGXFSZ 25 file size limit exceeded (see setrlimit(2))
SIGVTALRM 26 virtual time alarm (see setitimer(2))
SIGPROF 27 profiling timer alarm (see setitimer(2))
SIGWINCH 28* Window size change
SIGUSR1 30 User defined signal 1
SIGUSR2 31 User defined signal 2
The starred signals in the list above cause a core image if not caught or ignored.

If `func` is SIG_DFL, the default action for signal `sig` is reinstated; this default is termination (with a core image for starred signals) except for signals marked with • or †. Signals marked with • are discarded if the action is SIG_DFL; signals marked with † cause the process to stop. If `func` is SIG_IGN the signal is subsequently ignored and pending instances of the signal are discarded. Otherwise, when the signal occurs further occurrences of the signal are automatically blocked and `func` is called.

A return from the function unblocks the handled signal and continues the process at the point it was interrupted. **Unlike previous signal facilities, the handler `func` remains installed after a signal has been delivered.**

If a caught signal occurs during certain system calls, causing the call to terminate prematurely, the call is automatically restarted. In particular this can occur during a `read` or `write(2)` on a slow device (such as a terminal; but not a file) and during a `wait(2)`.

The value of `signal` is the previous (or initial) value of `func` for the particular signal.

After a `fork(2)` or `vfork(2)` the child inherits all signals. `execve(2)` resets all caught signals to the default action; ignored signals remain ignored.

**RETURN VALUE**

The previous action is returned on a successful call. Otherwise, −1 is returned and `errno` is set to indicate the error.

**ERRORS**

`signal` will fail and no action will take place if one of the following occur:

- [EINVAL] `sig` is not a valid signal number.
- [EINVAL] An attempt is made to ignore or supply a handler for SIGKILL or SIGSTOP.
- [EINVAL] An attempt is made to ignore SIGCONT (by default SIGCONT is ignored).

**SEE ALSO**

`kill(1)`, `ptrace(2)`, `kill(2)`, `sigvec(2)`, `sigblock(2)`, `sigsetmask(2)`, `sigpause(2)`, `sigstack(2)`, `setjmp(3)`, `tty(4)`, `sigreturn(2)`, `emulate_branch(3)`, `fpc(3)`, `cache_flush(2)`

R2010 Floating Point Coprocessor Architecture Engineering Description
R2360 Floating Point Board Product Description

**NOTES (MIPS)**

The handler routine can be declared:

```c
handler(sig, code, scp)
int sig, code;
struct sigcontext *scp;
```

Here `sig` is the signal number. MIPS hardware exceptions are mapped to specific signals as defined by the table below. `Code` is a parameter that is either a constant as given below or zero. `scp` is a pointer to the `sigcontext` structure (defined in `<signal.h>`), that is the context at the time of the signal and is used to restore the context if the signal handler returns.

The following defines the mapping of MIPS hardware exceptions to signals and codes. All of these symbols are defined in either `<signal.h>` or `<mips/cpu.h>`:

<table>
<thead>
<tr>
<th>Hardware exception</th>
<th>Signal</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer overflow</td>
<td>SIGFPE</td>
<td>EXC_OV</td>
</tr>
<tr>
<td>Segmentation violation</td>
<td>SIGSEGV</td>
<td>SEXC_SEGV</td>
</tr>
<tr>
<td>Illegal Instruction</td>
<td>SIGILL</td>
<td>EXC_II</td>
</tr>
</tbody>
</table>
Coprocomputer Unusable
Data Bus Error
Instruction Bus Error
Read Address Error
Write Address Error
User Breakpoint (used by debuggers)
Kernel Breakpoint (used by prom)
Taken Branch Delay Emulation
Not Taken Branch Delay Emulation
User Single Step (used by debuggers)
Overflow Check
Divide by Zero Check
Range Error Check

SIGILL
SIGBUS
SIGBUS
SIGBUS
SIGBUS
SIGTRAP
SIGTRAP
SIGTRAP
SIGTRAP
SIGTRAP
SIGTRAP
SIGTRAP
SIGTRAP
SEXCPU
EXCBDE
EXCBDE
EXCBDE
EXCBDE
BRKUSERBP
BRKKERNELBP
BRK_BDTAKEN
BRK_BDNOTTAKEN
BRK_SSTEPBP
BRKOVERFLOW
BRKDIVZERO
BRK_RANGE

When a signal handler is reached, the program counter in the signal context structure (sc_pc) points at the instruction that caused the exception as modified by the branch delay bit in the cause register. The cause register at the time of the exception is also saved in the sigcontext structure (sc_cause). If the instruction that caused the exception is at a valid user address it can be retrieved with the following code sequence:

```c
if(scp->sc_cause & CAUSE_BD){
    branch_instruction = *((unsigned long *) (scp->sc_pc));
    exception_instruction = *((unsigned long *) (scp->sc_pc + 4));
} else
    exception_instruction = *((unsigned long *) (scp->sc_pc));
```

Where CAUSE_BD is defined in `<mips/cpu.h>`.

The signal handler may fix the cause of the exception and re-execute the instruction, emulate the instruction and then step over it or perform some non-local goto such as a `longjump()` or an `exit()`.

If corrective action is performed in the signal handler and the instruction that caused the exception would then execute without a further exception, the signal handler simply returns and re-executes the instruction (even when the branch delay bit is set).

If execution is to continue after stepping over the instruction that caused the exception the program counter must be advanced. If the branch delay bit is set the program counter is set to the target of the branch else it is incremented by 4. This can be done with the following code sequence:

```c
if(scp->sc_cause & CAUSE_BD)
    emulate_branch(scp, branch_instruction);
else
    scp->sc_pc += 4;
```

`emulate_branch()` modifies the program counter value in the sigcontext structure to the target of the branch instruction. See `emulate_branch(3)` for more details.

For SIGFPE's generated by floating-point instructions (code == 0) the floating-point control and status register at the time of the exception is also saved in the sigcontext structure (sc_fpc csr). This register has the information on which exceptions have occurred. When a signal handler is entered the register contains the value at the time of the exception but with the exceptions bits cleared. On a return from the signal handler the exception bits in the floating-point control and status register are also cleared so that another SIGFPE will not occur (all other bits are restored from sc_fpc csr).
If the floating-point unit is a R2360 (a floating-point board) and a SIGFPE is generated by the floating-point unit (code == 0) and program counter does not point at the instruction that caused the exception. In this case the instruction that caused the exception is in the floating-point instruction exception register. The floating-point instruction exception register at the time of the exception is also saved in the sigcontext structure (sc_fpc_eir). In this case the instruction that caused the exception can be retrieved with the following code sequence:

```c
union fpc_irl fpc_irl;

fpc_irl.fi_word = get_fpc_irl();
if(sig == SIGFPE && code == 0 &&
   fpc_irl.fi_struct.implementation == IMPLEMENTATION_R2360)
   exception_instruction = scp->sc_fpc_eir;
```

The union `fpc_irl`, and the constant IMPLEMENTATION_R2360 are defined in `<mips/fpu.h>`. For the description of the routine `get_fpc_irl()` see `fpc(3)`. All other floating-point implementations are handled in the normal manner with the instruction that caused the exception at the program counter as modified by the `branch delay` bit.

For SIGSEGV and SIGBUS errors the faulting virtual address is saved in `sc_badvaddr` in the signal context structure.

The SIGTRAP's caused by `break` instructions noted in the above table and all other yet to be defined `break` instructions fill the `code` parameter with the first argument to the `break` instruction (bits 25-16 of the instruction).
NAME
    signal – change the action for a signal

SYNOPSIS
    integer function signal(signum, proc, flag)
    integer signum, flag
    external proc

DESCRIPTION
    When a process incurs a signal (see signal(3C)) the default action is usually to clean up and abort. The user may choose to write an alternative signal handling routine. A call to signal is the way this alternate action is specified to the system.

    Signum is the signal number (see signal(3C)). If flag is negative, then proc must be the name of the user signal handling routine. If flag is zero or positive, then proc is ignored and the value of flag is passed to the system as the signal action definition. In particular, this is how previously saved signal actions can be restored. Two possible values for flag have specific meanings: 0 means "use the default action" (See NOTES below), 1 means "ignore this signal".

    A positive returned value is the previous action definition. A value greater than 1 is the address of a routine that was to have been called on occurrence of the given signal. The returned value can be used in subsequent calls to signal in order to restore a previous action definition. A negative returned value is the negation of a system error code. (See perror(3F))

FILES
    /usr/lib/libU77.a

SEE ALSO
    signal(3C), kill(3F), kill(1)

NOTES
    f77 arranges to trap certain signals when a process is started. The only way to restore the default f77 action is to save the returned value from the first call to signal.

    If the user signal handler is called, it will be passed the signal number as an integer argument.
NAME

sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions and their inverses

SYNOPSIS

```c
#include <math.h>

double sin(x)
double x;
float fsin(float x)
float x;
double cos(x)
double x;
float fcos(float x)
float x;
double tan(float x)
double x;
float ftan(float x)
float x;
double asin(x)
double x;
float fasin(float x)
float x;
double acos(x)
double x;
float facos(float x)
float x;
double atan(x)
double x;
float fatan(float x)
float x;
double atan2(y,x)
double y,x;
float fatan2(float y, float x)
float y,x;
```

DESCRIPTION

Sin, cos and tan return trigonometric functions of radian arguments x for double data types. Fsin, fcos and ftan do the same for float data types.

Asin and fasin returns the arc sine in the range $-\pi/2$ to $\pi/2$ for double and float data types respectively.

Acos and facos returns the arc cosine in the range 0 to $\pi$ for double and float data types respectively.

Atan and fatan returns the arc tangent in the range $-\pi/2$ to $\pi/2$ for double and float data types respectively.

Atan2 and fatan2 returns the arctangent of y/x in the range $-\pi$ to $\pi$, using the signs of both arguments to determine the quadrant of the return value for double and float data types respectively.
DIAGNOSTICS
If |x| > 1 then asin(x) and acos(x) will return the default quiet NaN.

NOTES
Atan2 defines atan2(0,0) = 0. The reasons for assigning a value to atan2(0,0) are these:

(1) Programs that test arguments to avoid computing atan2(0,0) must be indifferent to its value. Programs that require it to be invalid are vulnerable to diverse reactions to that invalidity on diverse computer systems.

(2) Atan2 is used mostly to convert from rectangular (x,y) to polar (r,θ) coordinates that must satisfy x = r*cosθ and y = r*sinθ. These equations are satisfied when (x=0,y=0) is mapped to (r=0,θ=0). In general, conversions to polar coordinates should be computed thus:
\[ r := \text{hypot}(x,y); \quad \theta := \text{atan2}(y,x). \]

(3) The foregoing formulas need not be altered to cope in a reasonable way with signed zeros and infinities on a machine, such as MIPS machines, that conforms to IEEE 754; the versions of hypot and atan2 provided for such a machine are designed to handle all cases. That is why atan2(±0,−0) = ±π, for instance. In general the formulas above are equivalent to these:
\[ r := \sqrt{x^2+y^2}; \quad \text{if } r = 0 \text{ then } x := \text{copy sign}(1,x); \]
\[ \text{if } x > 0 \text{ then } \theta := 2\text{atan}(y/(r+x)) \]
\[ \text{else } \theta := 2\text{atan}((r-x)/y); \]
except if r is infinite then atan2 will yield an appropriate multiple of π/4 that would otherwise have to be obtained by taking limits.

ERROR (due to Roundoff etc.) for
Let P stand for the number stored in the computer in place of π = 3.14159 26535 89793 23846 26433 ... . Let "trig" stand for one of "sin", "cos" or "tan". Then the expression "trig(x)" in a program actually produces an approximation to trig(x*π/P), and "atrig(x)" approximates (P/π)*atrig(x). The approximations are close.

In the codes that run on MIPS machines, P differs from π by a fraction of an ulp; the difference matters only if the argument x is huge, and even then the difference is likely to be swamped by the uncertainty in x. Besides, every trigonometric identity that does not involve π explicitly is satisfied equally well regardless of whether P = π. For instance, sin²(x)+cos²(x) = 1 and sin(2x) = 2 sin(x)cos(x) to within a few ulps no matter how big x may be. Therefore the difference between P and π is most unlikely to affect scientific and engineering computations.

SEE ALSO
math(3M), hypot(3M), sqrt(3M)

AUTHOR
Robert P. Corbett, W. Kahan, Stuart I. McDonald, Peter Tang and, for the codes for IEEE 754, Dr. Kwok–Choi Ng.
NAME
   sinh, cosh, tanh - hyperbolic functions

SYNOPSIS
   #include <math.h>
   double sinh(x)
   double x;
   float fsinh(float x)
   float x;
   double cosh(x)
   double x;
   float fcosh(float x)
   float x;
   double tanh(x)
   double x;
   float ftanh(float x)
   float x;

DESCRIPTION
   These functions compute the designated hyperbolic functions for double and float data types.

ERROR (due to Roundoff etc.)
   Below 2.4 ulps; an ulp is one Unit in the Last Place.

DIAGNOSTICS
   Sinh and cosh return +∞ (and sinh may return -∞ for negative x) if the correct value would
   overflow.

SEE ALSO
   math(3M)

AUTHOR
   W. Kahan, Kwok-Choi Ng
NAME
sleep – suspend execution for interval

SYNOPSIS
sleep(seconds)
unsigned seconds;

DESCRIPTION
The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an interval timer and pausing 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 1 second later.

SEE ALSO
setitimer(2), sigpause(2), usleep(3)
NAME
sleep – suspend execution for an interval

SYNOPSIS
subroutine sleep (itime)

DESCRIPTION
Sleep causes the calling process to be suspended for itime seconds. The actual time can be up to 1 second less than itime due to granularity in system timekeeping.

FILES
/usr/lib/libU77.a

SEE ALSO
sleep(3)
NAME

cbrt, sqrt – cube root, square root

SYNOPSIS

#include <math.h>

double cbrt(x)
double x;
double sqrt(x)
double x;
float fsqrt(float x)
float x;

DESCRIPTION

Cbrt(x) returns the cube root of x.

Sqrt(x) and fsqrt(x) returns the square root of x for double and float data types respectively.

DIAGNOSTICS

Sqrt returns the default quiet NaN when x is negative indicating the invalid operation.

ERROR (due to Roundoff etc.)

Cbrt is accurate to within 0.7 ulps.

Sqrt on MIPS machines conforms to IEEE 754 and is correctly rounded in accordance with the rounding mode in force; the error is less than half an ulp in the default mode (round-to-nearest). An ulp is one Unit in the Last Place carried.

SEE ALSO

math(3M)

AUTHOR

W. Kahan
NAME
  standard - VADS standard library

SYNOPSIS
  standard

DESCRIPTION
  standard contains the VADS implementation of package STANDARD containing all predefined identifiers in the Ada RM as well as other predefined library units. The package STANDARD is an imaginary package that is available to every Ada program. The package enables Ada programmers to use predefined types, functions, and operations on those types.
  Additional packages are available as described in the Ada RM.
  The packages in standard include all types, functions, and operations described in the Ada RM Annex C, Predefined Language Environment.

FILES
  /usr/vads5/standard/*

SEE ALSO
  examples, publiclib, verdixlib
NAME
stat, fstat — get file status

SYNOPSIS
    integer function stat (name, statb)
    characters(*) name
    integer statb(12)
    characters(*) name
    integer statb(12)

    integer function fstat (lunit, statb)
    integer statb(12)

DESCRIPTION
These routines return detailed information about a file. Stat returns information about file
ame; fstat returns information about the file associated with fortran logical unit lunit. The
order and meaning of the information returned in array statb is as described for the structure
stat under stat(2). The "spare" values are not included.
The value of either function will be zero if successful; an error code otherwise.

FILES
/usr/lib/libU77.a

SEE ALSO
stat(2), access(3F), perror(3F), time(3F)

BUGS
Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.
NAME

staux - routines that provide scalar interfaces to auxiliaries

SYNOPSIS

#include <syms.h>

long st_auxbtadd(bt)
long bt;

long stauxbtsize(iaux, width)
long iaux;
long width;

long stauxsymadd (isym)
long isym;

long stauxrnxadd (rfd, index)
long rfd;
long index;

long stauxrnxadd (idn)
long idn;

void st_addtq (iaux, tq)
long iaux;
long tq;

long st_tqhigh_aux(iaux)
long iaux;

void st_shfttq (iaux, tq)
int iaux;
int tq;

long st_iaux_copyty (ifd, psym)
long ifd;
pSYM pSym;

void st_changeaux (iaux, aux)
long iaux;
AUXU aux;

void st_changeauxrmdx (iaux, rfd, index)
long iaux;
long rfd;
long index;

DESCRIPTION

Auxiliary entries are unions with a fixed length of four bytes per entry. Much information is packed within the auxiliaries. Rather than have the compiler front-ends handle each type of auxiliary entry directly, the following set of routines provide a high-level scalar interface to the auxiliaries:

st_auxbtadd

  Adds a type information record (TIR) to the auxiliaries. It sets the basic type (bt) to the argument and all other fields to zero. The index to this auxiliary entry is returned.

st_auxbtsize

  Sets the bit in the TIR, pointed to by the iaux argument. This argument says the basic type is a bit field and adds an auxiliary with its width in bits.

st_auxsymadd

   
Adds an index into the symbol table (or any other scalar) to the auxiliaries. It sets the value to the argument that will occupy all four bytes. The index to this auxiliary entry is returned.

`st_auxrndxadd`

Adds a relative index, RNDXR, to the auxiliaries. It sets the rfd and index to their respective arguments. The index to this auxiliary entry is returned.

`st_auxrndxadd_idn`

Works the same as `st_auxrndxadd` except that RNDXR is referenced by an index into the dense number table.

`st_iaux_copyt`

Copies the type from the specified file (ifd) for the specified symbol into the auxiliary table for the current file. It returns the index to the new aux.

`st_shifttq`

Shifts in the specified type qualifier, tq, into the auxiliary entry TIR, which is specified by the ‘iaux’ index into the current file. The current type qualifiers shift up one tq so that the first tq (tq0) is free for the new entry.

`st_addtq`

Adds a type qualifier in the highest or most significant non-tqNil type qualifier.

`st_tqhigh_iaux`

Returns the most significant type qualifier given an index into the files aux table.

`st_changeaux`

Changes the iauxth aux in the current file’s auxiliary table to aux.

`st_changeauxrnx`

Converts the relative index (RNDXR) auxiliary, which is specified by iaux, to the specified arguments.

**AUTHOR** Mark I. Himelstein

**SEE ALSO**

`stdf(3)`

**BUGS**

The interface will added to incrementally, as needed.
NAME
stcu – routines that provide a compilation unit symbol table interface

SYNOPSIS
#include <syms.h>

pCHDRR st_cuinit ()

void st_setchdr (pchdr)
pCHDRR pchdr;
pCHDRR st_currentpchdr()
void st_free()
long st_extadd (iss, value, st, sc, index)
long iss;
long value;
long st;
long sc;
long index;
pEXTR st_pext_iext (iext)
long iext;
pEXTR st_pext_rndx (rndx)
RNDXR rndx;
long st_iextmax()
long st_extstradd (str)
char *str;
char *st_str_extiss (iss)
long iss;
long st_idn_index_fext (index, fext)
long index;
long fext;
long st_idn_rndx (rndx)
RNDXR rndx;
pRNDXR st_pdn_idn (idn)
long idn;
RNDXR st_rndx_idn (idn)
long idn;
void st_setidn (idndest, idnsrc)
long idndest;
long idnsrc;

DESCRIPTION
The stcu routines provide an interface to objects that occur once per object rather than once per file descriptor (for example, external symbols, strings, and dense numbers). The routines provide access to the current chdr (compile time hdr), which represents the symbol table in running processes with pointers to symbol table sections rather than indices and offsets used in the disk file representation.

A new symbol table can be created with st_cuinit. This routine creates and initializes a CHDRR. The CHDRR is the current chdr and is used in all later calls. NOTE: A chdr can also be created with the read routines (see stio(3)). The st_cuinit routine returns a pointer to the new CHDRR record.
st_currentchdr
Returns a pointer the current chdr.

st_setchdr
Sets the current chdr to the pchdr argument and sets the per file structures to reflect a change in symbol tables.

st_free
Frees all constituent structures associated with the current chdr.

st_extadd
Lets you add to the externals table. It returns the index to the new external for future reference and use. The ifd field for the external is filled in by the current file (see stfd(3)).

st_pext_text
and st_pext_rndx
Returns pointers to the external, given a index referencing them. The latter routine requires a relative index where the index field should be the index in external symbols and the rfd field should be the constant ST_EXTIFD. NOTE: The externals contain the same structure as symbols (see the SYMR and EXTR definitions).

st_extmax
Returns the current number of entries in the external symbol table.
The iss field in external symbols (the index into string space) must point into external string space.

st_extstradd
Adds a null-terminated string to the external string space and returns its index.

st_str_extiss
Converts that index into a pointer to the external string.
The dense number table provides a convenience to the code optimizer, generator, and assembler. This table lets them reference symbols from different files and externals with unique densely packed numbers.

st_idn_index_fext
Returns a new dense number table index, given an index into the symbol table of the current file (or if fext is set, the externals table).

st_idn_rndx
Returns a new dense number, but expects a RNDXR to specify both the file index and the symbol index rather than implying the file index from the current file. The RNDXR contains two fields: an index into the externals table and a file index (rsyms can point into the symbol table, as well). The file index is ST_EXTIFD for externals.

st_rndx_idn
Returns a RNDX, given an index into the dense number table.

st_pdn_idn
Returns a pointer to the RNDXR index by the 'idn' argument.

AUTHOR Mark I. Himelstein
SEE ALSO
stfe(3), stfd(3)
NAME
stdf – routines that provide access to per file descriptor section of the symbol table

SYNOPSIS
#include <syms.h>
long st_currentifd ()
long st_ifdmax ()
void st_setfd (ifd)
long ifd;
long st_fdadd (filename)
char *filename;
long st_symadd (iss, value, st, sc, freloc, index)
long iss;
long value;
long st;
long sc;
long freloc;
long index;
long st_auxadd (aux)
AUXU aux;
long st_stradd (cp)
char *cp;
long st_lineadd (line)
long line;
long st_pdadd (isym)
long isym;
long st_ifd_pcfd (pcfdi)
pCFDR pcfdi;
pCFDR st_pcfd_ifd (ifd)
long ifd;
pSYMRF st_psym_ifd_isym (ifd, isym)
long ifd;
long isym;
pAUXU st_paux_ifd_iaux (ifd, iaux)
long ifd;
long iaux;
pAUXU st_paux_iaux (iaux)
long iaux;
char *st_str_iss (iss)
long iss;
char *st_str_ifd_iss (ifd, iss)
long ifd;
long iss;
pPDR st_ppd_ifd_isym (ifd, isym)
long ifd;
long isym;
NAME
stdio – standard buffered input/output package

SYNOPSIS
#include <stdio.h>
FILE *stdin;
FILE *stdout;
FILE *stderr;

DESCRIPTION
The functions described in section 3S constitute a user-level buffering scheme. The in-line
macros getchar and putc(3S) handle characters quickly. The higher level routines gets, fgets,
scanf, fscanf, fread, puts, fputs, printf, fprintf, fwrite all use getchar and putc; they can be freely
intermixed.

A file with associated buffering is called a stream, and is declared to be a pointer to a defined
type FILE. fopen(3S) creates certain descriptive data for a stream and returns a pointer to
designate the stream in all further transactions. There are three normally open streams with
constant pointers declared in the include file and associated with the standard open files:

stdin standard input file
stdout standard output file
stderr standard error file

A constant `pointer’ NULL (0) designates no stream at all.

An integer constant EOF (−1) is returned upon end of file or error by integer functions that
deal with streams.

Any routine that uses the standard input/output package must include the header file
<stdio.h> of pertinent macro definitions. The functions and constants mentioned in sections
labeled 3S are declared in the include file and need no further declaration. The constants,
and the following ‘functions’ are implemented as macros; redeclaration of these names is perils:
getc, getchar, putc, putchar, feof, ferror, fileno.

SEE ALSO
open(2), close(2), read(2), write(2), fread(3S), fseek(3S), fwrite(3S)

DIAGNOSTICS
The value EOF is returned uniformly to indicate that a FILE pointer has not been initialized
with fopen, input (output) has been attempted on an output (input) stream, or a FILE pointer
designates corrupt or otherwise unintelligible FILE data.

For purposes of efficiency, this implementation of the standard library has been changed to
line buffer output to a terminal by default and attempts to do this transparently by flushing the
output whenever a read(2) from the standard input is necessary. This is almost always tran-
sparent, but may cause confusion or malfunctioning of programs which use standard i/o rou-
tines but use read(2) themselves to read from the standard input.

In cases where a large amount of computation is done after printing part of a line on an output
terminal, it is necessary to fflush(3S) the standard output before going off and computing
so that the output will appear.

BUGS
The standard buffered functions do not interact well with certain other library and system
functions, especially vfork and abort.

LIST OF FUNCTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Appears on Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearerr</td>
<td>ferror.3s</td>
<td>stream status inquiries</td>
</tr>
</tbody>
</table>
fclose  fclose.3s  close or flush a stream
fopen   fopen.3s  open a stream
feof    ferror.3s  stream status inquiries
ferror  ferror.3s  stream status inquiries
fflush  fclose.3s  close or flush a stream
fgetc   getc.3s   get character or word from stream
fgets   gets.3s   get a string from a stream
fileno  ferror.3s  stream status inquiries
fopen   fopen.3s  open a stream
fprintf printf.3s  formatted output conversion
fputc   putc.3s   put character or word on a stream
fputs   puts.3s   put a string on a stream
fread   fread.3s  buffered binary input/output
freopen fopen.3s  open a stream
fscanf  scanf.3s  formatted input conversion
fseek   fseek.3s  reposition a stream
ftell   fseek.3s  reposition a stream
fwrite  fread.3s  buffered binary input/output
getc    getc.3s  get character or word from stream
getchar getc.3s  get character or word from stream
gets    gets.3s  get a string from a stream
getw   getc.3s  get character or word from stream
printf  printf.3s  formatted output conversion
putc    putc.3s  put character or word on a stream
putchar putc.3s  put character or word on a stream
puts    puts.3s  put a string on a stream
putw   putc.3s  put character or word on a stream
rewind  fseek.3s  reposition a stream
scanf  scanf.3s  formatted input conversion
setbuf  setbuf.3s  assign buffering to a stream
setbuffer setbuf.3s  assign buffering to a stream
setlinebuf setbuf.3s  assign buffering to a stream
sprintf printf.3s  formatted output conversion
sscanf  scanf.3s  formatted input conversion
ungetc  ungetc.3s  push character back into input stream
NAME

stfd – routines that provide access to per file descriptor section of the symbol table

SYNOPSIS

#include <syms.h>
long st_currentifd ()
long st_ifdmax ()
void st_setfd (ifd)
long ifd;
long st_fdadd (filename)
char *filename;
long st_symadd (iss, value, st, sc, freloc, index)
long iss;
long value;
long st;
long sc;
long freloc;
long index;
long st_auxadd (aux)
AUXU aux;
long st_stradd (cp)
char *cp;
long st_lineadd (line)
long line;
long st_pdaadd (isym)
long isym;
long st_ifd_pcfdr (pcfdr1)
pCFDR pcfdr1;
pCFDR st_pcfdr_ifd (ifd)
long ifd;
pSYMIR st_psym_ifd_isym (ifd, isym)
long ifd;
long isym;
pAUXU st_paux_ifd_iaux (ifd, iaux)
long ifd;
long iaux;
pAUXU st_paux_iaux (iaux)
long iaux;
char *st_str_iss (iss)
long iss;
char *st_str_ifd_iss (ifd, iss)
long ifd;
long iss;
pPDR st_ppd_ifd_isym (ifd, isym)
long ifd;
long isym;
char * st_malloc (ptr, psize, itemsize, baseitems)
char *ptr;
long *size;
long itemsize;
long baseitems;

DESCRIPTION

The stfd routines provide an interface to objects handled on a per file descriptor (or fd) level (for example, local symbols, auxiliaries, local strings, line numbers, optimization entries, procedure descriptor entries, and the file descriptors). These routines constitute a group because they deal with objects corresponding to fields in the FDR structure.

A fd can be activated by reading an existing one into memory or by creating a new one. The compilation unit routines st_readbinary and st_readst read file descriptors and their constituent parts into memory from a symbol table on disk.

St_fdadd adds a file descriptor to the list of file descriptors. The lang field is initialized from a user specified global st_lang that should be set to a constant designated for the language in symconst.h. The fMerge field is initialized from the user specified global st_merge that specifies whether the file is to start with the attribute of being able to be merged with identical files at load time. The fBigendian field is initialized by the gethostsex(3) routine, which determines the permanent byte ordering for the auxiliary and line number entries for this file.

St_fdadd adds the null string to the new files string table that is accessible by the constant issNull (0). It also adds the filename to the string table and sets the rss field. Finally, the current file is set to the newly added file so that later calls operate on that file.

All routines for fd-level objects handle only the current file unless a file index is specified. The current file can also be set with st_seth.

Programs can find the current file by calling st_currentifd, which returns the current index. Programs can find the number of files by calling st_ifdmax. The fd routines only require working with indices to do most things. They allow more in-depth manipulation by allowing users to get the compile time file descriptor (CFDR) that contains memory pointers to the per file tables (rather than indices or offsets used in disk files). Users can retrieve a pointer to the CFDR by calling st_pcfd_ifd with the index to the desired file. The inverse mapping st_ifd_pcfd exists, as well.

Each of fd's constituent parts has an add routine: st_symadd, st_stradd, st_lineadd, st_pdadd, and st_auxadd. The parameters of the add routines correspond to the fields of the added object. The pdadd routine lets users fill in the isym field only. Further information can be added by directly accessing the procedure descriptor entry.

The add routines return an index that can be used to retrieve a pointer to part of the desired object with one of the following routines: st_psym_psym, st_striss, and st_paux_iaux. NOTE: These routines only return objects within the current file. The following routines allow for file specification: st_psym_ifd_isym, st_aux_ifd_iaux, and st_str_ifd_iss.

St_ppd_ifd_isym allows access to procedures through the file index for the file where they occur and the isym field of the entry that points at the local symbol for that procedure.

The return index from st_symadd should be used to get a dense number (see stcu(3)). That number should be the ucode block number for the object the symbol describes.

AUTHOR Mark I. Himelstein
SEE ALSO stfe(3), stcu(3).
BUGS

The interface will added to incrementally, as needed.
NAME
stfe - routines that provide a high-level interface to basic functions needed to access and add
to the symbol table

SYNOPSIS
#include <syms.h>
long st_filebegin (filename, lang, merge, glevel)
char *filename;
long lang;
long merge;
long glevel;
long st_endallfiles ()
long st_fileend (idn)
long idn;
long st_blockbegin(iss, value, sc)
long iss;
long value;
long sc;
long st_textblock ()
long st_blockend (size)
long size;
long st_procend (idn)
long idn
long st_procbegin (idn)
long idn;
char *st_str_idn (idn)
long idn;
char *st_sym_idn (idn, value, sc, st, index)
long idn;
long +value;
long +sc;
long +st;
long +index;
long st_abs_ifd_index (ifd, index)
long ifd;
long index;
long st_fglobal_idn (idn)
long idn;
pSYM R st_psym_idn_offset (idn, offset)
long idn;
long offset;
long st_pdaddd_idn (idn)
long idn;

DESCRIPTION
The stfe routines provide a high-level interface to the symbol table based on common needs of
the compiler front-ends.
st_filebegin
should be called upon encountering each cpp directive in the front end. It calls
\texttt{st\_fileadd} to add symbols and will find the appropriate open file or start a new file. It
takes a filename, language constant (see \texttt{symconst.h}), a merge flag (0 or 1) and the \texttt{-g}
level constant (see \texttt{symconst.h}). It returns a dense number pointing to the file symbol
to be used in line number directives.

\texttt{st\_fileend}

Requires the dense number from the corresponding \texttt{st\_filebegin} call for the file in
question. It then generates an end symbol and patches the references so that the
index field of the begin file points to that of one beyond the end file. The end file
points to the begin file.

\texttt{st\_endallfiles}

Is called at the end of execution to close off all files that haven't been ended by previous
calls to \texttt{st\_filebegin}. CPP directives might not reflect the return to the original
source file; therefore, this routine can possibly close many files.

\texttt{st\_blockbegin}

Supports both language blocks (for example, C's left curly brace blocks), beginning
of structures, and unions. If the storage class is \texttt{scText}, it is the former; if it is \texttt{scInfo},
it is one of the latter. The \texttt{iss} (index into string space) specifies the name of the
structure/etc, if any.

If the storage class is \texttt{scText}, we must check the result of \texttt{st\_blockbegin}. It returns a dense
number for outer blocks and a zero for nested blocks. The non-zero block number should be
used in the BGNB ucode. Users of languages without nested blocks that provide variable
declarations can ignore the rest of this paragraph. Nested blocks are two-staged: one stage
happens when we detect the language block and the other stage happens when we know the
block has content. If the block has content (for example, local variables), the front-end must
call \texttt{st\_textblock} to get a non-zero dense number for the block's BGNB ucode. If the block has
no content and \texttt{st\_textblock} is not called, the block's \texttt{st\_blockbegin} and \texttt{st\_blockend} do not
produce block and end symbols.

If it is \texttt{scInfo}, \texttt{st\_blockbegin} creates a begin block symbol in the symbol table and returns a
dense number referencing it. The dense number is necessary to build the auxiliary required to
reference the structure/etc. It goes in the aux after the TIR along with a file index. This dense
number is also noted in a stack of blocks used by \texttt{st\_blockend}.

\texttt{St\_blockbegin} should not be called for language blocks when the front-end is not producing
debugging symbols.

\texttt{St\_blockend} requires that blocks occur in a nested fashion. It retrieves the dense number for
the most recently started block and creates a corresponding end symbol. As in \texttt{fileend}, both
the begin and end symbol index fields point at the other end's symbol. If the symbol ends a
structure/etc., as determined by the storage class of the begin symbol, the size parameter
is assigned to the begin symbol's value field. It's usually the size of the structure or max value of
an enum. We only know it at this point. The dense number of the end symbol is returned so
that the ucode ENDB can be use it. If it is an ignored text block, the dense number is zero
and no ENDB should be generated.

In general, defined external procedures or functions appear in the symbols table and the externals
table. The external table definition must occur first through the use of a \texttt{st\_extadd}. After
that definition, \texttt{st\_procbegin} can be called with a dense number referring to the external sym-
bol for that procedure. It checks to be sure we have a defined procedure (by checking the
storage class). It adds a procedure symbol to the symbol table. The external's index should
point at its auxiliary data type information (or if debugging is off, indexNil). This index is
copied into the regular symbol's index field or a copy of its type is generated (if the external is
in a different file than the regular symbol). Next, we put the index to symbol in the external's index field. The external's dense number is used as a block number in ucodes referencing it and is used to add a procedure when in the \texttt{st\_pdadd\_idn}.

\texttt{st\_procend}

Creates an end symbol and fixes the indices as in \texttt{blockend} and \texttt{fileend}, except that the end procedure reference is kept in the begin procedure's aux rather than in the index field (because the begin procedure has a type as well as an end reference). This must be called with the dense number of the procedure's external symbol as an argument and returns the dense number of the end symbol to be used in the END ucode.

\texttt{st\_str\_idn}

Returns the string associated with symbol or external referenced by the dense number argument. If the symbol was anonymous (for example, there was no symbol) a (char *) -1 is returned.

\texttt{st\_sym\_idn}

Returns the same result as \texttt{st\_str\_idn}, except that the rest of the fields of the symbol specified by the \texttt{idn} are returned in the arguments.

\texttt{st\_fglobal\_idn}

Returns a 1 if the symbol associated with the specified idn is non-static; otherwise, a 0 is returned.

\texttt{st\_abs\_ifd\_index}

Returns the absolute offset for a dense number. If the symbol is global, the global's index is returned. If the symbol occurred in a file, the sum of all symbols in files occurring before that file and the symbol's index within the file is returned.

\texttt{st\_pdadd\_idn}

Adds an entry to the procedure table for the \texttt{st\_proc entry} generated by procbegin. This should be called when the front-end generates code for the procedure in question.

\textbf{AUTHOR} Mark I. Himelstein

\textbf{SEE ALSO}

\texttt{stcu(3), stfd(3)}
NAME
stio – routines that provide a binary read/write interface to the MIPS symbol table

SYNOPSIS
#include <syms.h>

long st_readbinary (filename, how)
char *filename;
char how;

long st_readst (fn, how, filebase, pchdr, flags)
long fn;
char how;
long filebase;
pCHDRR pchdr;
long flags;

void st_writebinary (filename, flags)
char *filename;
long flags;

void st_writest (fn, flags)
long fn;
long flags;

DESCRIPTION
The CHDRR structure (see stcu(3)) represents a symbol table in memory. A new CHDRR
can be created by reading a symbol table in from disk. St_readbinary and st_readst read a
symbol table in from disk.

St_readbinary takes the file name of the symbol table and assumes the symbol table header
HDRR occurs at the beginning of the file. St_readst assumes that its file number references a
file positioned at the beginning of the symbol table header and that the filebase parameter
specifies where the object or symbol table file is based (for example, non-zero for archives).

The second parameter to the read routines can be ‘r’ for read only or ‘a’ for appending to the
symbol table. Existing local symbol, line, procedure, auxiliary, optimization, and local string
tables can not be appended. If they didn’t exist on disk, they can be created. This restriction
stems from the allocation algorithm for those symbol table sections when read in from disk
and follows the standard pattern for building the symbol table.

The symbol table can be read incrementally. If pchdr is zero, st_readst assumes that no sym-
bol table has been read yet; therefore, it reads in the symbol table header and file descriptors.
The flags argument is a bit mask that defines what other tables should be read. St_p* constants
for each table can be ORed. If flags equals ‘-1’, all tables are read. If pchdr is set, the tables
specified by flags are added to the tables that have already been read. The value of pchdr can
be gotten from st_current_pchdr (see stcu(3)).

Line number entries are encoded on disk, and the read routines expand them to longs. See
the MIPS System Programmer Guide.

If the version stamp is out of date, a warning message is issued to stderr. If the magic number
in the HDRR is incorrect, st_error is called. All other errors cause the read routines to read
non-zero; otherwise, a zero is returned.

St_writebinary and st_writest are symmetric to the read routines, excluding the how and pchdr
parameters. The flags parameter is a bit mask that defines what table should be written. St_p*
constants for each table can be ORed. If flags equals ‘-1’, all tables are written.
The write routines write sections of the table in the approved order, as specified in the link editor (ld) specification.

Line numbers are compressed on disk. See the MIPS System Programmer Guide.

The write routines start all sections of the symbol table on four-byte boundaries.

If the write routines encounter an error, st_error is called. After writing the symbol table, further access to the table by other routines is undefined.

AUTHOR Mark I. Himelstein

SEE ALSO

stcu(3), stfe(3), stfd(3).

The MIPS System Programmer Guide.
NAME
stprint - routines to print the symbol table

SYNOPSIS
#include <syms.h>
#include <stdio.h>
char *st_mlang_ascii [];
char *st_mst_ascii [];
char *st_msc_ascii [];
char *st_mbst_ascii [];
char *st_mtb_asci [];
void st_dump (fd, flags)
FILE *fd;
long flags;
void st_printfd (fd, ifd, flags)
FILE *fd;
long ifd;
long flags;

DESCRIPTION
The stprint routines and arrays provide an easy way to print the MIPS symbol table. The print
the symbol table from st_current pchdr().

The arrays map constants to their ASCII equivalents. The constants can be found in
symconst.h and represent languages (lang), symbol types (st), storage classes (sc), basic types
(bt), and type qualifiers (aq).

The st_dump routine prints an ASCII version of the symbol. If fd is NULL, the routine prints
file fd and stdout. The flags can be a mask of a section of symbol table specified by ORing
ST_Ps constants together from cmplrs/stsupport.h. This routine modifies the current file.

st_printfd prints the sections associated with the file specified by the ifd argument. The other
arguments are the same as in st_dump. These arguments modify the current file, as well.

AUTHOR Mark I. Hmelstein

SEE ALSO
stfe(3), stcu(3), sym.h(5), stsupport.h(5)

ERRORS
The interface will be added to incrementally as needed.
NAME
strcat, strncat, stremp, strn cmp, strcpy, strncpy, strlen, index, strchr, rindex, strrchr, strp brk,
strspn, strcspn, strtok – string operations

SYNOPSIS
#include <string.h>
char *strcat (s1, s2)
char *s1, *s2;
char *strncat (s1, s2, n)
char *s1, *s2;
int n;
int strcmp (s1, s2)
char *s1, *s2;
int strncmp (s1, s2, n)
char *s1, *s2;
int n;
char *strcpy (s1, s2)
char *s1, *s2;
char *strncpy (s1, s2, n)
char *s1, *s2;
int n;
int strlen (s)
char *s;
char *index (s, c)
char *s;
int c;
char *strchr (s, c)
char *s;
int c;
char *rindex (s, c)
char *s;
int c;
char *strrchr (s, c)
char *s;
int c;
char *strpbrk (s1, s2)
char *s1, *s2;
int strspn (s1, s2)
char *s1, *s2;
int strlen (s1, s2)
char *s1, *s2;
char *strtok (s1, s2)
char *s1, *s2;

DESCRIPTION
The arguments s1, s2 and s point to strings (arrays of characters terminated by a null character). The functions strcat, strncat, strcpy, and strncpy all alter s1. These functions do not check for overflow of the array pointed to by s1.
`strcat` appends a copy of string `s2` to the end of string `s1`. `strcat` appends at most `n` characters. Each returns a pointer to the null-terminated result.

`strncpy` compares its arguments and returns an integer less than, equal to, or greater than 0, according as `s1` is lexicographically less than, equal to, or greater than `s2`. `strncpy` makes the same comparison but looks at at most `n` characters.

`strcpy` copies string `s2` to `s1`, stopping after the null character has been copied. `strncpy` copies exactly `n` characters, truncating `s2` or adding null characters to `s1` if necessary. The result will not be null-terminated if the length of `s2` is `n` or more. Each function returns `s1`.

`strlen` returns the number of characters in `s`, not including the terminating null character.

`Index (rindex)` returns a pointer to the first (last) occurrence of character `c` in string `s`, or a NULL pointer if `c` does not occur in the string. The null character terminating a string is considered to be part of the string. The routines `strchr` and `strchr` are, respectively, different names for the `index` and `rindex`.

`strpbrk` returns a pointer to the first occurrence in string `s1` of any character from string `s2`, or a NULL pointer if no character from `s2` exists in `s1`.

`strspn` returns the length of the initial segment of string `s1` which consists entirely of characters from (not from) string `s2`.

`strtok` considers the string `s1` to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string `s2`. The first call (with pointer `s1` specified) returns a pointer to the first character of the first token, and will have written a null character into `s1` immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument a NULL pointer) will work through the string `s1` immediately following that token. In this way subsequent calls will work through the string `s1` until no tokens remain. The separator string `s2` may be different from call to call. When no token remains in `s1`, a NULL pointer is returned.

**NOTE**

For user convenience, all these functions are declared in the optional `<string.h>` header file.

**ERRORS**

`strcmp` and `strncmp` use native character comparison, which is signed on PDP-11s and VAX-11s, unsigned on other machines. Thus the sign of the value returned when one of the characters has its high-order bit set is implementation-dependent.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.
NAME
stime, gtty – set and get terminal state (defunct)

SYNOPSIS

#include <sgtty.h>

stime(fd, buf)
int fd;
struct sgttyb *buf;

gtty(fd, buf)
int fd;
struct sgttyb *buf;

DESCRIPTION

This interface is obsoleted by ioctl(2).

stime sets the state of the terminal associated with fd. gtty retrieves the state of the terminal associated with fd. To set the state of a terminal the call must have write permission.

The stime call is actually “ioctl(fd, TIOCSETP, buf)”, while the gtty call is “ioctl(fd, TIOCGETP, buf)”. See ioctl(2) and tty(4) for an explanation.

DIAGNOSTICS

If the call is successful 0 is returned, otherwise -1 is returned and the global variable errno contains the reason for the failure.

SEE ALSO

ioctl(2), tty(4)
NAME
  swab – swap bytes

SYNOPSIS
  swab(from, to, nbytes)
  char *from, *to;

DESCRIPTION
  swab copies nbytes bytes pointed to by from to the position pointed to by to, exchanging adjacent even and odd bytes. It is useful for carrying binary data between PDP11's and other machines. nbytes should be even.
NAME
syslog, openlog, closelog, setlogmask – control system log

SYNOPSIS
#include <syslog.h>
openlog(ident, logopt, facility)
char *ident;
syslog(priority, message, parameters ...)
char *message;
closelog()
setlogmask(maskpri)

DESCRIPTION
syslog arranges to write message onto the system log maintained by syslogd(8). The message is tagged with priority. The message looks like a printf(3) string except that %m is replaced by the current error message (collected from errno). A trailing newline is added if needed. This message will be read by syslogd(8) and written to the system console, log files, or forwarded to syslogd on another host as appropriate.

Priorities are encoded as a facility and a level. The facility describes the part of the system generating the message. The level is selected from an ordered list:

LOG_EMERG A panic condition. This is normally broadcast to all users.
LOG_ALERT A condition that should be corrected immediately, such as a corrupted system database.
LOG_CRIT Critical conditions, e.g., hard device errors.
LOG_ERR Errors.
LOG_WARNING Warning messages.
LOG_NOTICE Conditions that are not error conditions, but should possibly be handled specially.
LOG_INFO Informational messages.
LOG_DEBUG Messages that contain information normally of use only when debugging a program.

If syslog cannot pass the message to syslogd, it will attempt to write the message on /dev/console if the LOG_CONS option is set (see below).

If special processing is needed, openlog can be called to initialize the log file. The parameter ident is a string that is prepended to every message. logopt is a bit field indicating logging options. Current values for logopt are:

LOG_PID log the process id with each message: useful for identifying instantiations of daemons.
LOG_CONS Force writing messages to the console if unable to send it to syslogd. This option is safe to use in daemon processes that have no controlling terminal since syslog will fork before opening the console.
LOG_NDELAY Open the connection to syslogd immediately. Normally the open is delayed until the first message is logged. Useful for programs that need to manage the order in which file descriptors are allocated.
LOG_NOWAIT Don't wait for children forked to log messages on the console. This option should be used by processes that enable notification of child termination via SIGCHLD, as syslog may otherwise block waiting for a child.
whose exit status has already been collected.

The *facility* parameter encodes a default facility to be assigned to all messages that do not have an explicit facility encoded:

- **LOG_KERN**: Messages generated by the kernel. These cannot be generated by any user processes.
- **LOG_USER**: Messages generated by random user processes. This is the default facility identifier if none is specified.
- **LOG_MAIL**: The mail system.
- **LOG_DAEMON**: System daemons, such as `ftpd(8)`, `routed(8)`, etc.
- **LOG_AUTH**: The authorization system: `login(1)`, `su(1)`, `getty(8)`, etc.
- **LOG_LPR**: The line printer spooling system: `lpr(1)`, `lpc(8)`, `lpd(8)`, etc.
- **LOG_LOCAL0**: Reserved for local use. Similarly for **LOG_LOCAL1** through **LOG_LOCAL7**.

*closelog* can be used to close the log file.

*setlogmask* sets the log priority mask to *maskpri* and returns the previous mask. Calls to *syslog* with a priority not set in *maskpri* are rejected. The mask for an individual priority *pri* is calculated by the macro **LOG_MASK(pri)**; the mask for all priorities up to and including *toppri* is given by the macro **LOG_UPTO(toppri)**. The default allows all priorities to be logged.

**EXAMPLES**

```c
syslog(LOG_ALERT, "who: internal error 23");

openlog("ftpd", LOG_PID, LOG_DAEMON);
setlogmask(LOG_UPTO(LOG_ERR));
syslog(LOG_INFO, "Connection from host \%d", CallingHost);

syslog(LOG_INFO|LOG_LOCAL2, "foobar error: \%m");
```

**SEE ALSO**

`logger(1)`, `syslogd(8)`
NAME
   system – issue a shell command

SYNOPSIS
   system(string)
   char *string;

DESCRIPTION
   system causes the string to be given to sh(1) as input as if the string had been typed as a com-
mand at a terminal. The current process waits until the shell has completed, then returns the
exit status of the shell.

SEE ALSO
   popen(3S), execve(2), wait(2)

DIAGNOSTICS
   Exit status 127 indicates the shell couldn't be executed.
NAME
  system – execute a UNIX command

SYNOPSIS
  integer function system (string)
  character*(*) string

DESCRIPTION
  System causes string to be given to your shell as input as if the string had been typed as a com-
mand. If environment variable SHELL is found, its value will be used as the command inter-
preter (shell); otherwise sh(1) is used.

The current process waits until the command terminates. The returned value will be the exit
status of the shell. See wair(2) for an explanation of this value.

FILES
  /usr/lib/libU77.a

SEE ALSO
  exec(2), wait(2), system(3)

BUGS
  String can not be longer than NCARGS–50 characters, as defined in <sys/param.h>.
NAME

tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs – terminal independent operation routines

SYNOPSIS

char PC;
char *BC;
char *UP;
short ospeed;
tgetent(bp, name);
char *bp, *name;
tgetnum(id)
char *id;
tgetflag(id)
char *id;
char *
tgetstr(id, area)
char *id, **area;
char *
tgoto(cm, destcol, destline)
char *cm;
tputs(cp, affcnt, oute)
register char *cp;
int affcnt;
int (*oute)();

DESCRIPTION

These functions extract and use capabilities from the terminal capability data base termcap(5). These are low level routines; see curses(3X) for a higher level package.

tgetent extracts the entry for terminal name into the buffer at bp. Bp should be a character buffer of size 4096 and must be retained through all subsequent calls to tgetnum, tgetflag, and tgetstr. tgetent returns -1 if it cannot open the termcap file, 0 if the terminal name given does not have an entry, and 1 if all goes well. It will look in the environment for a TERMCP variable. If found, and the value does not begin with a slash, and the terminal type name is the same as the environment string TERM, the TERMCP string is used instead of reading the termcap file. If it does begin with a slash, the string is used as a path name rather than /etc/termcap. This can speed up entry into programs that call tgetent, as well as to help debug new terminal descriptions or to make one for your terminal if you can't write the file /etc/termcap.

tgetnum gets the numeric value of capability id, returning -1 if is not given for the terminal. tgetflag returns 1 if the specified capability is present in the terminal's entry, 0 if it is not. tgetstr returns the string value of the capability id, places it in the buffer at area, and advances the area pointer. It decodes the abbreviations for this field described in termcap(5), except for cursor addressing and padding information. tgetstr returns NULL if the capability was not found.

tgoto returns a cursor addressing string decoded from cm to go to column destcol in line destline. It uses the external variables UP (from the up capability) and BC (if bc is given rather than bs) if necessary to avoid placing \n, ^D or ^@ in the returned string. (Programs which call tgoto should be sure to turn off the XTABS bit(s), since tgoto may now output a tab. Note that programs using termcap should in general turn off XTABS anyway since some terminals use control I for other functions, such as nondestructive space.) If a % sequence is given
which is not understood, then \texttt{tgoto} returns "OOPS".

\texttt{tputs} decodes the leading padding information of the string \texttt{cp}; \texttt{affcnt} gives the number of lines affected by the operation, or 1 if this is not applicable, \texttt{outc} is a routine which is called with each character in turn. The external variable \texttt{ospeed} should contain the output speed of the terminal as encoded by \texttt{stty(3)}. The external variable \texttt{PC} should contain a pad character to be used (from the \texttt{pe} capability) if a null (\texttt{\@}) is inappropriate.

\textbf{FILES}
\begin{verbatim}
/usr/lib/libtermcap.a -ltermcap library
/etc/termcap       data base
\end{verbatim}

\textbf{SEE ALSO}
\begin{itemize}
\item ex(1), curses(3X), termcap(5)
\end{itemize}

\textbf{AUTHOR}
William Joy
NAME
time, ftime – get date and time

SYNOPSIS
long time(0)
long time(tloc)
long *tloc;
#include <sys/types.h>
#include <sys/timeb.h>
ftime(tp)
struct timeb *tp;

DESCRIPTION
These interfaces are obsoleted by gettimeofday(2).

time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.
If tloc is nonnull, the return value is also stored in the place to which tloc points.
The ftime entry fills in a structure pointed to by its argument, as defined by <sys/timeb.h>:
/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved. The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 * *
 */ ( Berkely) 6/4/86

/*
 * Structure returned by ft ime system call
 */
struct timeb
{
    time_t time;
    unsigned short millitm;
    short timezone;
    short dstflag;
};
The structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-
precise interval, 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.

SEE ALSO
date(1), gettimeofday(2), settimeofday(2), ctime(3)
NAME
time, ctime, ltime, gmtime – return system time

SYNOPSIS
integer function time()

character*() function ctime (stime)
integer stime

subroutine ltime (stime, tarray)
integer stime, tarray(9)

subroutine gmtime (stime, tarray)
integer stime, tarray(9)

DESCRIPTION
Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds. This is the value of the UNIX system clock.

Ctime converts a system time to a 24 character ASCII string. The format is described under ctime(3). No 'newline' or NULL will be included.

Ltime and gmtime dissect a UNIX time into month, day, etc., either for the local time zone or as GMT. The order and meaning of each element returned in tarray is described under ctime(3).

FILES
/usr/lib/libU77.a

SEE ALSO
ctime(3), itime(3F), idate(3F), fdate(3F)
NAME
		times – get process times

SYNOPSIS
	#include <sys/types.h>
	#include <sys/times.h>

times(buffer)
struct tms *buffer;

DESCRIPTION
This interface is obsoleted by getrusage(2).

times returns time-accounting information for the current process and for the terminated child
processes of the current process. All times are in 1/HZ seconds, where HZ is 60.

This is the structure returned by times:

*/
* Copyright (c) 1982, 1986 Regents of the University of California.
* All rights reserved. The Berkeley software License Agreement
* specifies the terms and conditions for redistribution.
*
* @(#)times.h 7.1 (Berkeley) 6/4/86
*/

/*
 * Structure returned by times()
 */
struct tms {

time_t tms_utime;    / * user time */
time_t tms_stime;    / * system time */
time_t tms_cutime;   / * user time, children */
time_t tms_cstime;   / * system time, children */
}

The children times are the sum of the children’s process times and their children’s times.

SEE ALSO

time(1), getrusage(2), wait3(2), time(3)
NAME
timezone – supply timezone string

SYNOPSIS
cchar *timezone(zone, dst)

DESCRIPTION
NOTE: This routine is supplied for use with programs that need it. Recent changes in the
Daylight Savings Time rules may make the value returned incorrect at times. Programs that
need the current timezone should be changed to get it from the array tzname, as described in
cftime(3).

timezone returns the name of the time zone associated with its first argument, which is measured
in minutes westward from Greenwich. If the second argument is 0, the standard name is
used, otherwise the Daylight Saving version.

If the required name does not appear in a table built into the routine, the difference from
GMT is produced; e.g., in Afghanistan timezone(-((60*4+30), 0) is appropriate because it is 4:30
ahead of GMT and the string GMT+4:30 is produced.

SEE ALSO
gftimeofday(2), ctime(3)
NAME

ttynamex, isatty, ttyslot – find name of a terminal

SYNOPSIS

char *ttynamex(filedes)
isatty(filedes)
ttyslot()

DESCRIPTION

ttynamex returns a pointer to the null-terminated path name of the terminal device associated
with file descriptor filedes (this is a system file descriptor and has nothing to do with the stan-
dard I/O FILE typedef).
isaty returns 1 if filedes is associated with a terminal device, 0 otherwise.
ttyslot returns the number of the entry in the ttys(5) file for the control terminal of the current
process.

FILES

/dev/*
/etc/ ttys

SEE ALSO

ioctl(2), ttys(5)

DIAGNOSTICS

ttynamex returns a null pointer (0) if filedes does not describe a terminal device in directory
‘/dev’.
ttyslot returns 0 if ‘/etc/ ttys’ is inaccessible or if it cannot determine the control terminal.

ERRORS

The return value points to static data whose content is overwritten by each call.
NAME
tynam, isatty — find name of a terminal port

SYNOPSIS
characters(*) function tynam (lunit)

logical function isatty (lunit)

DESCRIPTION
Tynam returns a blank padded path name of the terminal device associated with logical unit lunit.
Isatty returns .true. if lunit is associated with a terminal device, .false. otherwise.

FILES
/dev/*
/usr/lib/libU77.a

DIAGNOSTICS
Tynam returns an empty string (all blanks) if lunit is not associated with a terminal device in directory ‘/dev’.
NAME
ualarm – schedule signal after specified time

SYNOPSIS
unsigned ualarm(value, interval)
unsigned value;
unsigned interval;

DESCRIPTION
This is a simplified interface to setitimer(2).

ualarm causes signal SIGALRM, see signal(3C), to be sent to the invoking process in a
number of microseconds given by the value argument. Unless caught or ignored, the signal
terminates the process.

If the interval argument is non-zero, the SIGALRM signal will be sent to the process every
interval microseconds after the timer expires (e.g. after value microseconds have passed).
Because of scheduling delays, resumption of execution of when the signal is caught may be
delayed an arbitrary amount. The longest specifiable delay time (on the vax) is 2147483647
microseconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO
getitimer(2), setitimer(2), sigpause(2), sigvec(2), signal(3C), sleep(3), alarm(3), usleep(3)
NAME
handle_unaligned_traps, print_unaligned_summary – gather statistics on unaligned references

SYNOPSIS
void handle_unaligned_traps();
void print_unaligned_summary();
long unaligned_load_word(addr)
    char *addr;
long unaligned_load_half(addr)
    char *addr;
long unaligned_load_uhalf(addr)
    char *addr;
float unaligned_load_float(addr)
    char *addr;
double unaligned_load_double(addr)
    char *addr;
void unaligned_store_word(addr, value)
    char *addr;
    long value;
void unaligned_store_half(addr, value)
    char *addr;
    long value;
void unaligned_store_float(addr, float value)
    char *addr;
    float value;
void unaligned_store_double(addr, value)
    char *addr;
    double value;

DESCRIPTION
The first two routines implement a facility for finding unaligned references. The MIPS hardware traps load and store operations where the address is not a multiple of the number of bytes loaded or stored. Usually this trap indicates incorrect program operation and so by default the kernel converts this trap into a SIGBUS signal to the process, typically causing a core dump for debugging.

Older programs developed on systems with lax alignment constraints sometimes make occasional misaligned references in course of correct operation. The best way to port such programs to MIPS hardware is to correct the program by aligning the data.
A call to handle_unaligned_traps installs a SIGBUS handler that fixes unaligned memory references and keeps a record of the types, counts, and instruction addresses of these traps. A call to print_unaligned_summary prints the accumulated information. The following is an example of the output produced by print_unaligned_summary:

The listing is written to standard error and describes the type and number of unaligned references, followed by a list of every address that contains an unaligned reference. To convert the addresses into a dbx(I) script and run the script, pipe the output (both standard output and standard error) through the following command. The output from dbx will be the name of the function and line number of the misalignment.

```
sed -n -e 's;^ # [0-9a-f]+/i/,+$/i;1;p' | dbx prog
```

This information can be used to decide the best way to correct the problem. If not all of the data can be aligned, or not all of the identified program locations that reference unaligned data can be changed, the sysmips(2) [MIPS_FIXADE] system call may be appropriate.

The other routines load or store their indicated data type at the address specified. The address need not meet the normal alignment constraints.

There exist fortran entry points for these routines so they may be called directly from fortran with the names documented here.

**DIAGNOSTICS**

If these routines try to load or store to an address that is outside the program’s address space a SIGSEGV signal will be generated from inside these routines. If the program did not use these routines and the address was unaligned then the program would generate a SIGBUS signal. This is because the check for alignment is done before the address is checked to be in the program’s address space.

**SEE ALSO**

dbx(1), sysmips(2) [MIPS_FIXADE], signal(2), sigset(2).
NAME

handle_unaligned_traps, print_unaligned_summary – gather statistics on unaligned references

SYNOPSIS

void handle_unaligned_traps()
void print_unaligned_summary()
long unaligned_load_word(addr)
char *addr;
long unaligned_load_half(addr)
char *addr;
long unaligned_load_uhalf(addr)
char *addr;
float unaligned_load_float(addr)
char *addr;
double unaligned_load_double(addr)
char *addr;
void unaligned_store_word(addr, value)
char *addr;
long value;
void unaligned_store_half(addr, value)
char *addr;
long value;
void unaligned_store_float(addr, float value)
char *addr;
float value;
void unaligned_store_double(addr, value)
char *addr;
double value;

DESCRIPTION

The first two routines implement a facility for finding unaligned references. The MIPS hardware traps load and store operations where the address is not a multiple of the number of bytes loaded or stored. Usually this trap indicates incorrect program operation and so by default the kernel converts this trap into a SIGBUS signal to the process, typically causing a core dump for debugging.

Older programs developed on systems with lax alignment constraints sometimes make occasional misaligned references in course of correct operation. The best way to port such programs to MIPS hardware is to correct the program by aligning the data.

A call to handle_unaligned_traps installs a SIGBUS handler that fixes unaligned memory references and keeps a record of the types, counts, and instruction addresses of these traps. A call to print_unaligned_summary prints the accumulated information. The following is an example of the output produced by print_unaligned_summary:
The listing is written to standard error and describes the type and number of unaligned references, followed by a list of every address that contains an unaligned reference. To convert the addresses into a `dbx(1)` script and run the script, pipe the output (both standard output and standard error) through the following command. The output from `dbx` will be the name of the function and line number of the misalignment.

```
    sed -n -e 's;^# [0-9a-f]*/i);*##$;1;p' | dbx prog
```

This information can be used to decide the best way to correct the problem. If not all of the data can be aligned, or not all of the identified program locations that reference unaligned data can be changed, the `fixade(2)` `sysmips(2)` [MIPS_FIXADE] system call may be appropriate.

The other routines load or store their indicated data type at the address specified. The address need not meet the normal alignment constraints.

There exist fortran entry points for these routines so they may be called directly from fortran with the names documented here.

**DIAGNOSTICS**

If these routines try to load or store to an address that is outside the program's address space a SIGSEGV signal will be generated from inside these routines. If the program did not use these routines and the address was unaligned then the program would generate a SIGBUS signal. This is because the check for alignment is done before the address is checked to be in the program's address space.

**SEE ALSO**

`dbx(1), fixade(2), sigvec(2). sysmips(2)` [MIPS_FIXADE], `signal(2), sigset(2).`
NAME
ungetc - push character back into input stream

SYNOPSIS
#include <stdio.h>
ungetc(c, stream)
FILE *stream;

DESCRIPTION
ungetc pushes the character c back on an input stream. That character will be returned by the
next getc call on that stream. ungetc returns c.

One character of pushback is guaranteed provided something has been read from the stream
and the stream is actually buffered. Attempts to push EOF are rejected.

fseek(3S) erases all memory of pushed back characters.

SEE ALSO
getc(3S), setbuf(3S), fseek(3S)

DIAGNOSTICS
ungetc returns EOF if it can't push a character back.
NAME
  unlink – remove a directory entry

SYNOPSIS
  integer function unlink (name)
  characters(* name

DESCRIPTION
  Unlink causes the directory entry specified by pathname name to be removed. If this was the
  last link to the file, the contents of the file are lost. The returned value will be zero if successful; a system error code otherwise.

FILES
  /usr/lib/libU77.a

SEE ALSO
  unlink(2), link(3F), filsys(5), perror(3F)

BUGS
  Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.
NAME
usleep – suspend execution for interval

SYNOPSIS
usleep(useconds)
unsigned useconds;

DESCRIPTION
The current process is suspended from execution for the number of microseconds specified by the argument. The actual suspension time may be an arbitrary amount longer because of other activity in the system or because of the time spent in processing the call.

The routine is implemented by setting an interval timer and pausing 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.

This routine is implemented using setitimer(2); it requires eight system calls each time it is invoked. A similar but less compatible function can be obtained with a single select(2); it would not restart after signals, but would not interfere with other uses of setitimer.

SEE ALSO
setitimer(2), getitimer(2), sigpause(2), ualarm(3), sleep(3), alarm(3)
NAME
  utime – set file times

SYNOPSIS
  #include <sys/types.h>
  utime(file, timep)
  char *file;
  time_t timep[2];

DESCRIPTION
  This interface is obsoleted by utimes(2).
  The utime call uses the ‘accessed’ and ‘updated’ times in that order from the timep vector to set the corresponding recorded times for file.
  The caller must be the owner of the file or the super-user. The ‘inode-changed’ time of the file is set to the current time.

SEE ALSO
  utimes(2), stat(2)
NAME
valloc – aligned memory allocator

SYNOPSIS
char *valloc(size)
unsigned size;

DESCRIPTION
Valloc is obsoleted by the current version of malloc, which aligns page-sized and larger allocations.

valloc allocates size bytes aligned on a page boundary. It is implemented by calling malloc(3) with a slightly larger request, saving the true beginning of the block allocated, and returning a properly aligned pointer.

DIAGNOSTICS
valloc returns a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block.

ERRORS
vfree isn't implemented.
NAME

varargs – variable argument list

SYNOPSIS

#include <varargs.h>

function(va_list)
    va_dcl
    va_list pvar;
    va_start(pvar);
    f = va_arg(pvar, type);
    va_end(pvar);

DESCRIPTION

This set of macros provides a means of writing portable procedures that accept variable argument lists. Routines having variable argument lists (such as printf(3)) that do not use varargs are inherently nonportable, since different machines use different argument passing conventions.

va_list is used in a function header to declare a variable argument list.

va_dcl is a declaration for va_list. Note that there is no semicolon after va_dcl.

va_list is a type which can be used for the variable pvar, which is used to traverse the list. One such variable must always be declared.

va_start(pvar) is called to initialize pvar to the beginning of the list.

va_arg(pvar, type) will return the next argument in the list pointed to by pvar. type is the type to which the expected argument will be converted when passed as an argument. In standard C, arguments that are char or short should be accessed as int, unsigned char or unsigned 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.

va_end(pvar) is used to finish up.

Multiple traversals, each bracketed by va_start ... va_end, are possible.

EXAMPLE

#include <varargs.h>

execv(va_list)
    va_dcl
{
    va_list ap;
    char *file;
    char *args[100];
    int argno = 0;

    va_start(ap);
    file = va_arg(ap, char *);
    while (args[argno++] = va_arg(ap, char *))
    ;
    va_end(ap);
    return execv(file, args);
ERRORS

It is up to the calling routine to determine how many arguments there are, since it is not possible to determine this from the stack frame. For example, *exec* passes a 0 to signal the end of the list. *printf* can tell how many arguments are supposed to be there by the format.

The macros *va_start* and *va_end* may be arbitrarily complex; for example, *va_start* might contain an opening brace, which is closed by a matching brace in *va_end*. Thus, they should only be used where they could be placed within a single complex statement.
NAME
verdixlib – MIPS-supported Ada library packages

SYNOPSIS
verdixlib

DESCRIPTION
verdixlib contains the packages MATH, COMPLEX_ARITH, ORDERING, COMMAND_LINE, and UNIX_CALLS. MATH uses the UNIX C mathematics library to provide most standard mathematical functions and many constants. COMPLEX_ARITH defines the private type type COMPLEX and provides arithmetic functions for complex numbers. ORDERING includes sorting packages (QUICKSORT, HEAPSORT, and INSERTIONSORT) and a permuting package (PERMUTE).

COMMAND_LINE lets the user access the command line arguments and environments variables of an Ada program. UNIX_CALLS provides an interface to commonly used UNIX system calls.

TYPES AND FUNCTIONS
private type COMPLEX in COMPLEX_ARITH

FILES
/usr/vads5/verdixlib/*

SEE ALSO
MATH fully describes the MATH and COMPLEX_ARITH packages. Other libraries of Ada programs are standard, publiclib, and examples.
NAME
vlimit – control maximum system resource consumption

SYNOPSIS
#include <sys/vlimit.h>
vlimit(resource, value)

DESCRIPTION
This facility is superseded by getrlimit(2).

Limits the consumption by the current process and each process it creates to not individually exceed value on the specified resource. If value is specified as -1, then the current limit is returned and the limit is unchanged. The resources which are currently controllable are:

LIM_NORaise A pseudo-limit; if set non-zero then the limits may not be raised. Only the super-user may remove the noraise restriction.

LIM_CPU the maximum number of cpu-seconds to be used by each process

LIMFSIZE the largest single file which can be created

LIM_DATA the maximum growth of the data+stack region via sbrk(2) beyond the end of the program text

LIM_STACK the maximum size of the automatically-extended stack region

LIM_CORE the size of the largest core dump that will be created.

LIM_MAXRSS a soft limit for the amount of physical memory (in bytes) to be given to the program. If memory is tight, the system will prefer to take memory from processes which are exceeding their declared LIM_MAXRSS.

Because this information is stored in the per-process information this system call must be executed directly by the shell if it is to affect all future processes created by the shell; limit is thus a built-in command to csh(1).

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way; a break call fails if the data space limit is reached, or the process is killed when the stack limit is reached (since the stack cannot be extended, there is no way to send a signal).

A file i/o operation which would create a file which is too large will cause a signal SIGXFSZ to be generated, this normally terminates the process, but may be caught. When the cpu time limit is exceeded, a signal SIGXCPU is sent to the offending process; to allow it time to process the signal it is given 5 seconds grace by raising the cpu time limit.

SEE ALSO
csh(1)

ERRORS
LIM_NORaise no longer exists.
NAME
vtimes - get information about resource utilization

SYNOPSIS
#include <sys/vtimes.h>
vtimes(par_vm, ch_vm)
struct vtimes *par_vm, *ch_vm;

DESCRIPTION
This facility is superseded by getrusage(2).

vtimes 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 0, in which case
only the information for the pointers which are non-zero is returned.

After the call, each buffer contains information as defined by the contents of the include file
/usr/include/sys/vtimes.h:

struct vtimes {
    int vm_utime;           /* user time (+HZ) */
    int vm_stime;           /* system time (+HZ) */
    /* divide next two by utime+stime to get averages */
    unsigned vm_idrss;      /* integral of d+s rss */
    unsigned vm_ixrss;      /* integral of text rss */
    int vm_maxrss;          /* maximum rss */
    int vm_majflt;          /* major page faults */
    int vm_minflt;          /* minor page faults */
    int vm_nswap;           /* number of swaps */
    int vm_inblk;           /* block reads */
    int vm_oublk;           /* block writes */
};

The vm_utime and vm_stime fields give the user and system time respectively in 60ths of a
second (or 50ths if that is the frequency of wall current in your locality.) The vm_idrss and
vm_ixrss measure memory usage. They are computed by integrating the number of memory
pages in use each over cpu time. They are reported as though computed discretely, adding the
current memory usage (in 512 byte pages) each time the clock ticks. If a process used 5 core
pages over 1 cpu-second for its data and stack, then vm_idrss would have the value 5*60,
where vm_utime+vm_stime would be the 60. vm_idrss integrates data and stack segment usage,
while vm_ixrss integrates text segment usage. vm_maxrss reports the maximum instantaneous
sum of the text+data+stack core-resident page count.

The vm_majflt field gives the number of page faults which resulted in disk activity; the
vm_minflt field gives the number of page faults incurred in simulation of reference bits;
vm_nswap is the number of swaps which occurred. The number of file system input/output
events are reported in vm_inblk and vm_oublk These numbers 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.

SEE ALSO
    time(2), wait3(2), getrusage(2)
NAME
wait – wait for a process to terminate

SYNOPSIS
integer function wait (status)
integer status

DESCRIPTION
Wait causes its caller to be suspended until a signal is received or one of its child processes terminates. If any child has terminated since the last wait, return is immediate; if there are no children, return is immediate with an error code.

If the returned value is positive, it is the process ID of the child and status is its termination status (see wait(2)). If the returned value is negative, it is the negation of a system error code.

FILES
/usr/lib/libU77.a

SEE ALSO
wait(2), signal(3F), kill(3F), perror(3F)
NAME
xdr – library routines for external data representation

DESCRIPTION
These routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Data for remote procedure calls are transmitted using these routines.

FUNCTIONS
xdr_array() translate arrays to/from external representation
xdr_bool() translate Booleans to/from external representation
xdr_bytes() translate counted byte strings to/from external representation
xdr_destroy() destroy XDR stream and free associated memory
xdr_double() translate double precision to/from external representation
xdr_enum() translate enumerations to/from external representation
xdr_float() translate floating point to/from external representation
xdr_getpos() return current position in XDR stream
xdr_inline() invoke the in-line routines associated with XDR stream
xdr_int() translate integers to/from external representation
xdr_long() translate long integers to/from external representation
xdr_opaque() translate fixed-size opaque data to/from external representation
xdr_reference() chase pointers within structures
xdr_setpos() change current position in XDR stream
xdr_short() translate short integers to/from external representation
xdr_string() translate null-terminated strings to/from external representation
xdr_u_int() translate unsigned integers to/from external representation
xdr_u_long() translate unsigned long integers to/from external representation
xdr_u_short() translate unsigned short integers to/from external representation
xdr_union() translate discriminated unions to/from external representation
xdr_void() always return one (1)
xdr_wrapstring() package RPC routine for XDR routine, or vice-versa
xdrmem_create() initialize an XDR stream
xdrrec_create() initialize an XDR stream with record boundaries
xdrrec_endofrecord() mark XDR record stream with an end-of-record
xdrrec_eof() mark XDR record stream with an end-of-file
xdrrec_skiprecord() skip remaining record in XDR record stream
xdrstdio_create() initialize an XDR stream as standard I/O FILE stream

SEE ALSO
NAME

ypclnt, yp_get_default_domain, yp_bind, yp_unbind, yp_match, yp_first, yp_next, yp_all,
yp_order, yp_master, yperr_string, yp prot err — yellow pages client interface

SYNOPSIS

#include <rpcsvc/ypclnt.h>

yp binds (indomain);
char *indomain;

void yp unbind (indomain)
char *indomain;

 yp get default domain (outdomain);
char * * outdomain;

yp_match (indomain, inmap, inkey, inkeylen, outval, outval len)
char *indomain;
char *inmap;
char *inkey;
int inkeylen;
char * * outval;
int * outval len;

yp_first (indomain, inmap, outkey, outkeylen, outval, outval len)
char *indomain;
char *inmap;
char *outkey;
int * outkeylen;
char * * outval;
int * outval len;

yp_next (indomain, inmap, inkey, inkeylen, outkey, outkeylen, outval, outval len);
char *indomain;
char *inmap;
char *inkey;
int inkeylen;
char * * outkey;
int * outkeylen;
char * * outval;
int * outval len;

yp_all (indomain, inmap, incallback);
char *indomain;
char *inmap;
struct ypall_callback incallback;

yp_order (indomain, inmap, outorder);
char *indomain;
char *inmap;
int * outorder;

yp_master (indomain, inmap, outname);
char *indomain;
char *inmap;
char * * outname;
char *yperr_string (incode)
int incode;

ypprot_err(incode)
unsigned int incode;

DESCRIPTION

This package of functions provides an interface to the yellow pages (YP) network lookup service. The package can be loaded from the standard library, /lib/libc.a. Refer to yfiles(5) and ypserv(8) for an overview of the yellow pages, including the definitions of map and domain, and a description of the various servers, databases, and commands that comprise the YP.

All input parameters names begin with in. Output parameters begin with out. Output parameters of type char ** should be addresses of uninitialized character pointers. Memory is allocated by the YP client package using malloc(3), and may be freed if the user code has no continuing need for it. For each outkey and outval, two extra bytes of memory are allocated at the end that contain NEWLINE and NULL, respectively, but these two bytes are not reflected in outkeylen or outvallen. indomain and inmap strings must be non-null and null-terminated. String parameters which are accompanied by a count parameter may not be null, but may point to null strings, with the count parameter indicating this. Counted strings need not be null-terminated.

All functions in this package of type int return 0 if they succeed, and a failure code (YPERR_xxxx) otherwise. Failure codes are described under DIAGNOSTICS below.

The YP lookup calls require a map name and a domain name, at minimum. It is assumed that the client process knows the name of the map of interest. Client processes should fetch the node’s default domain by calling yp_get_default_domain(), and use the returned outdomain as the indomain parameter to successive YP calls.

To use the YP services, the client process must be “bound” to a YP server that serves the appropriate domain using yp_bind. Binding need not be done explicitly by user code; this is done automatically whenever a YP lookup function is called. yp_bind can be called directly for processes that make use of a backup strategy (e.g., a local file) in cases when YP services are not available.

Each binding allocates (uses up) one client process socket descriptor; each bound domain costs one socket descriptor. However, multiple requests to the same domain use that same descriptor. yp_unbind() is available at the client interface for processes that explicitly manage their socket descriptors while accessing multiple domains. The call to yp_unbind() make the domain unbound, and free all per-process and per-node resources used to bind it.

If an RPC failure results upon use of a binding, that domain will be unbound automatically. At that point, the ypclnt layer will retry forever or until the operation succeeds, provided that ypbind is running, and either

a) the client process can’t bind a server for the proper domain, or

b) RPC requests to the server fail.

If an error is not RPC-related, or if ypbind is not running, or if a bound ypserv process returns any answer (success or failure), the ypclnt layer will return control to the user code, either with an error code, or a success code and any results.

yp_match returns the value associated with a passed key. This key must be exact; no pattern matching is available.

yp_first returns the first key-value pair from the named map in the named domain.
yp_next() returns the next key-value pair in a named map. The inkey parameter should be the outkey returned from an initial call to yp_first() (to get the second key-value pair) or the one returned from the nth call to yp_next() (to get the nth + second key-value pair).

The concept of first (and, for that matter, of next) is particular to the structure of the YP map being processed; there is no relation in retrieval order to either the lexical order within any original (non-YP) data base, or to any obvious numerical sorting order on the keys, values, or key-value pairs. The only ordering guarantee made is that if the yp_first() function is called on a particular map, and then the yp_next() function is repeatedly called on the same map at the same server until the call fails with a reason of YPERR_NOMORE, every entry in the data base will be seen exactly once. Further, if the same sequence of operations is performed on the same map at the same server, the entries will be seen in the same order.

Under conditions of heavy server load or server failure, it is possible for the domain to become unbound, then bound once again (perhaps to a different server) while a client is running. This can cause a break in one of the enumeration rules; specific entries may be seen twice by the client, or not at all. This approach protects the client from error messages that would otherwise be returned in the midst of the enumeration. The next paragraph describes a better solution to enumerating all entries in a map.

yp_all provides a way to transfer an entire map from server to client in a single request using TCP (rather than UDP as with other functions in this package). The entire transaction take place as a single RPC request and response. You can use yp_all just like any other YP procedure, identify the map in the normal manner, and supply the name of a function which will be called to process each key-value pair within the map. You return from the call to yp_all only when the transaction is completed (successfully or unsuccessfully), or your “foreach” function decides that it doesn’t want to see any more key-value pairs.

The third parameter to yp_all is

struct ypall_callback *incallback {
    int (*foreach)();
    char *data;
};

The function foreach is called

foreach(instatus, inkey, inkeylen, inval, invallen, indata);
int instatus;
char *inkey;
int inkeylen;
char *inval;
int invallen;
char *indata;

The instatus parameter will hold one of the return status values defined in <rpcsvc/yp_prot.h> – either YP_TRUE or an error code. (See ypprot_err, below, for a function which converts a YP protocol error code to a ypclnt layer error code.)

The key and value parameters are somewhat different than defined in the synopsis section above. First, the memory pointed to by the inkey and inval parameters is private to the yp_all function, and is overwritten with the arrival of each new key-value pair. It is the responsibility of the foreach function to do something useful with the contents of that memory, but it does not own the memory itself. Key and value objects presented to the foreach function look exactly as they do in the server’s map – if they were not newline-terminated or null-terminated in the map, they won’t be here either.
The *indata* parameter is the contents of the *incallback-*->*data* element passed to *yp_all*. The *data* element of the callback structure may be used to share state information between the *foreach* function and the mainline code. Its use is optional, and no part of the YP client package inspects its contents – cast it to something useful, or ignore it as you see fit.

The *foreach* function is a Boolean. It should return zero to indicate that it wants to be called again for further received key-value pairs, or non-zero to stop the flow of key-value pairs. If *foreach* returns a non-zero value, it is not called again; the functional value of *yp_all* is then 0.

*yp_order* returns the order number for a map.

*yp_master* returns the machine name of the master YP server for a map.

*yperr_string* returns a pointer to an error message string that is null-terminated but contains no period or newline.

*ypprot_err* takes a YP protocol error code as input, and returns a ypclnt layer error code, which may be used in turn as an input to *yperr_string*.

**FILES**

/usr/include/rpcssvc/ypclnt.h
/usr/include/rpcssvc/yp_prot.h

**SEE ALSO**

ypfiles(5), ypserv(8),

**DIAGNOSTICS**

All integer functions return 0 if the requested operation is successful, or one of the following errors if the operation fails.

```
#define YPERR_BADARGS 1 /* args to function are bad */
#define YPERR_RPC 2 /* RPC failure - domain has been unbound */
#define YPERR_DOMAIN 3 /* can't bind to server on this domain */
#define YPERR_MAP 4 /* no such map in server's domain */
#define YPERR_KEY 5 /* no such key in map */
#define YPERR_YPERR 6 /* internal yp server or client error */
#define YPERR_RESRC 7 /* resource allocation failure */
#define YPERR_NOMORE 8 /* no more records in map database */
#define YPERR_PMAP 9 /* can't communicate with portmapper */
#define YPERR_YPBIND 10 /* can't communicate with ypbind */
#define YPERR_YPSERV 11 /* can't communicate with ypserv */
#define YPERR_NODOM 12 /* local domain name not set */
```
NAME
ypassword – update user password in yellow pages

SYNOPSIS
#include <rpcsvc/ypassword.h>
ypassword(oldpass, newpw)
    char *oldpass
    struct passwd *newpw;

DESCRIPTION
If oldpass is indeed the old user password, this routine replaces the password entry with newpw. It returns 0 if successful.

RPC INFO
program number:
YPASSWORDPROG
xdr routines:
    xdr_password(xdrs, yp)
        XDR *xdrs;
        struct yppassword *yp;
    xdr_yppassword(xdrs, pw)
        XDR *xdrs;
        struct passwd *pw;

procs:
YPASSWORDPROC_UPDATE
    Takes struct yppassword as argument, returns integer.
    Same behavior as yppassword() wrapper.
    Uses UNIX authentication.

versions:
YPASSWORDVERS_ORIGIN
structures:
    struct yppassword {
        char *oldpass; /* old (unencrypted) password */
        struct passwd newpw; /* new pw structure */
    }

SEE ALSO
ypassword(1), yppassword(8C)
NAME
acct – execution accounting file

SYNOPSIS
#include <sys/acct.h>

DESCRIPTION
The acct(2) system call arranges for entries to be made in an accounting file for each process that terminates. The accounting file is a sequence of entries whose layout, as defined by the include file is:

/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved. The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 * @(#)acct.h 7.1 (Berkeley) 6/4/86
 */

/*
 * Accounting structures;
 * these use a comp_t type which is a 3 bits base 8
 * exponent, 13 bit fraction “floating point” number.
 * Units are 1/AHZ seconds.
 */
typedef u_short comp_t;

struct acct {
    char    ac_comm[10];    /* Accounting command name */
    comp_t  ac_utime;       /* Accounting user time */
    comp_t  ac_stime;       /* Accounting system time */
    comp_t  ac_etime;       /* Accounting elapsed time */
    time_t  ac_btime;       /* Beginning time */
    uid_t   ac_uid;         /* Accounting user ID */
    gid_t   ac_gid;         /* Accounting group ID */
    short   ac_mem;         /* average memory usage */
    comp_t  ac_io;          /* number of disk IO blocks */
    dev_t   ac_tty;         /* control typewriter */
    char    ac_flag;        /* Accounting flag */
};

#define AFORK 0001        /* has executed fork, but no exec */
#define ASU 0002           /* used super-user privileges */
#define ACOMPAT 0004       /* used compatibility mode */
#define ACORE 0010         /* dumped core */
#define AXSIG 0020         /* killed by a signal */

/*
 * 1/AHZ is the granularity of the data encoded in the various
 * comp_t fields. This is not necessarily equal to hz.
 */
#define AHZ 64
#ifdef KERNEL
struct acct acctbuf;
struct vnode *acctp;
#endif

If the process was created by an execve(2), the first 10 characters of the filename appear in ac_comm. The accounting flag contains bits indicating whether execve(2) was ever accomplished, and whether the process ever had super-user privileges.

SEE ALSO
acct(2), execve(2), sa(8)
NAME
cshrc – startup file for csh command

SYNOPSIS
$HOME/.cshrc

DESCRIPTION
When csh(1) is executed without the option -f, it reads commands from the file
$HOME/.cshrc. If the shell is a login shell (this can be done by logging in, executing the
login(1) command, or executing su(1) with the - option), the file $HOME/.login is executed
after the .cshrc file.

This file should do the following:

Set the path variable (this must be here if rsh(1c) or rcp(1c) is to work properly. If
these are not required, the path may be set in the file $HOME/.login).

Set up aliases (interactive shells only).

Set up internal csh variables for things like line editing, filename completion, history,
etc (interactive shells only).

Set the prompt to be used when the shell is invoked as a non-login shell (interactive
shells only).

In general, the format of the file is as follows (items in {} should be replaced by appropriate
commands and/or pathnames):

    set path= ( . \
          {personal bins} \ 
          {local/project bins} \ 
          $path \ 
          /usr/new \ 
          /usr/new/mh \ 
    )
    set cdpath= ( {path for use with cd command} )
    if ($?prompt) then
        {set prompt}
        {set variables and aliases for interactive shells}
    else
        {set variables and aliases for non-interactive shells}
    endif
    {set variables and aliases for all shells}

There is almost never a reason to execute any commands in this file other than those for set-
ing up variables and aliases. Special care should be taken to avoid executing commands like
biff(1), sysline(1), or tset(1), especially in non-interactive shells.

Environment variables can be set in $HOME/.login at login time, since they are passed to all
subshells. In fact, setting environment variables in .cshrc can cause unexpected results.

An example of a useful .cshrc file is:

    #!/bin/csh -f
    # .cshrc for root

    set path = (/usr/ucb /bin /usr/bin /etc )
# Things for interactive shells
if ($?prompt) then
    alias j jobs -1
    alias h history
    alias z suspend
    set history=100
else
    # nothing for non-interactive shells
endif

SEE ALSO
csh(1), su(1), login(5), profile(5)
NAME
printcap – printer capability database

SYNOPSIS
/etc/printcap

DESCRIPTION
printcap is a simplified version of the termcap(5) database used to describe line printers. The
spooling system accesses the printcap file every time it is used, allowing dynamic addition and
deletion of printers. Each entry in the database is used to describe one printer. This database
can not be substituted for, as is possible for termcap, because it can allow accounting to be
bypassed.

The default printer is normally lp, though the environment variable PRINTER can be used to
override this. Each spooling utility supports an option, -Pprinter, to allow explicit naming of
a destination printer.

Refer to the 4.2BSD Line Printer Spooler Manual for a complete discussion on how setup the
database for a given printer.

CAPABILITIES
Refer to termcap for a description of the file layout.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>af</td>
<td>str</td>
<td>NULL</td>
<td>name of accounting file</td>
</tr>
<tr>
<td>br</td>
<td>num</td>
<td>none</td>
<td>if lp is a tty, set the baud rate (ioctl call)</td>
</tr>
<tr>
<td>cf</td>
<td>str</td>
<td>NULL</td>
<td>cifplot data filter</td>
</tr>
<tr>
<td>df</td>
<td>str</td>
<td>NULL</td>
<td>tex data filter (DVI format)</td>
</tr>
<tr>
<td>fc</td>
<td>num</td>
<td>0</td>
<td>if lp is a tty, clear flag bits (sgtty.h)</td>
</tr>
<tr>
<td>ff</td>
<td>str</td>
<td>&quot;f&quot;</td>
<td>string to send for a form feed</td>
</tr>
<tr>
<td>fo</td>
<td>bool</td>
<td>false</td>
<td>print a form feed when device is opened</td>
</tr>
<tr>
<td>fs</td>
<td>num</td>
<td>0</td>
<td>like ‘fc’ but set bits</td>
</tr>
<tr>
<td>gf</td>
<td>str</td>
<td>NULL</td>
<td>graph data filter (plot (3X) format)</td>
</tr>
<tr>
<td>ic</td>
<td>bool</td>
<td>false</td>
<td>driver supports (non standard) ioctl to indent printout</td>
</tr>
<tr>
<td>if</td>
<td>str</td>
<td>NULL</td>
<td>name of text filter which does accounting</td>
</tr>
<tr>
<td>lf</td>
<td>str</td>
<td>&quot;/dev/console&quot;</td>
<td>error logging file name</td>
</tr>
<tr>
<td>lo</td>
<td>str</td>
<td>&quot;lock&quot;</td>
<td>name of lock file</td>
</tr>
<tr>
<td>lp</td>
<td>str</td>
<td>&quot;/dev/lp&quot;</td>
<td>device name to open for output</td>
</tr>
<tr>
<td>mx</td>
<td>num</td>
<td>1000</td>
<td>maximum file size (in BUFSIZ blocks), zero = unlimited</td>
</tr>
<tr>
<td>nd</td>
<td>str</td>
<td>NULL</td>
<td>next directory for list of queues (unimplemented)</td>
</tr>
<tr>
<td>nf</td>
<td>str</td>
<td>NULL</td>
<td>ditroff data filter (device independent troff)</td>
</tr>
<tr>
<td>of</td>
<td>str</td>
<td>NULL</td>
<td>name of output filtering program</td>
</tr>
<tr>
<td>pl</td>
<td>num</td>
<td>66</td>
<td>page length (in lines)</td>
</tr>
<tr>
<td>pw</td>
<td>num</td>
<td>132</td>
<td>page width (in characters)</td>
</tr>
<tr>
<td>px</td>
<td>num</td>
<td>0</td>
<td>page width in pixels (horizontal)</td>
</tr>
<tr>
<td>py</td>
<td>num</td>
<td>0</td>
<td>page length in pixels (vertical)</td>
</tr>
<tr>
<td>rf</td>
<td>str</td>
<td>NULL</td>
<td>filter for printing FORTRAN style text files</td>
</tr>
<tr>
<td>rm</td>
<td>str</td>
<td>NULL</td>
<td>machine name for remote printer</td>
</tr>
<tr>
<td>rp</td>
<td>str</td>
<td>&quot;lp&quot;</td>
<td>remote printer name argument</td>
</tr>
<tr>
<td>rs</td>
<td>bool</td>
<td>false</td>
<td>restrict remote users to those with local accounts</td>
</tr>
<tr>
<td>rw</td>
<td>bool</td>
<td>false</td>
<td>open the printer device for reading and writing</td>
</tr>
<tr>
<td>sb</td>
<td>bool</td>
<td>false</td>
<td>short banner (one line only)</td>
</tr>
<tr>
<td>sc</td>
<td>bool</td>
<td>false</td>
<td>suppress multiple copies</td>
</tr>
<tr>
<td>sd</td>
<td>str</td>
<td>&quot;/usr/spool/lpd&quot;</td>
<td>spool directory</td>
</tr>
<tr>
<td>sf</td>
<td>bool</td>
<td>false</td>
<td>suppress form feeds</td>
</tr>
<tr>
<td>sh</td>
<td>bool</td>
<td>false</td>
<td>suppress printing of burst page header</td>
</tr>
</tbody>
</table>
Error messages sent to the console have a carriage return and a line feed appended to them, rather than just a line feed.

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

**EXAMPLE**

```
# This is a sample of printcap entries used by various printers/plotters
#
# DecWriter over a tty line.
lp|lp|arpa|ucbarpa|A-180 DecWriter III:
    :br#1200:fs#06320:tr=of=/usr/lib/lpf:lf=/usr/adm/lpd-errs:
# typical remote printer entry
ucbvax|vax|x|ucbvax line printer:
    :lp=rm=ucbvax:sd=/usr/spool/vaxlpd:lf=/usr/adm/lpd-errs:
varian|a|Benson Varian:
    :lp=/dev/va0:sd=/usr/spool/vad:mx#2000:pl#58:px#2112:py#1700:tr=
        :gf=/usr/lib/vplotf:df=/usr/local/dvif:
        :vf=/usr/lib/vpldmp:lf=/usr/adm/lpd-errs:
versatec|vp|Versatec plotter:
    :lp=/dev/vp0:sd=/usr/spool/vpd:sb:sf:mx#0:pw#106:pl#86:px#7040:py#2400:
        :gf=/usr/lib/vplotf:vf=/usr/lib/vpldmp:lf=/usr/adm/lpd-errs:
        :tr=
```

**SEE ALSO**

termcap(5), lpc(8), lpd(8), pac(8), lpr(1), lpq(1), lprm(1)

*4.2BSD Line Printer Spooler Manual*
NAME
termcap - terminal capability data base

SYNOPSIS
/etc/termcap

DESCRIPTION
termcap is a data base describing terminals, used, e.g., by vi(1) and curses(3X). Terminals are described in termcap by giving a set of capabilities that they have and by describing how operations are performed. Padding requirements and initialization sequences are included in termcap.

Entries in termcap consist of a number of ‘:’-separated fields. The first entry for each terminal gives the names that are known for the terminal, separated by ‘|’ characters. The first name is always two characters long and is used by older systems which store the terminal type in a 16-bit word in a system-wide data base. The second name given is the most common abbreviation for the terminal, the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the first and last should be in lower case and contain no blanks; the last name may well contain upper case and blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, thus “hp2621”. This name should not contain hyphens. Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Therefore, a “vt100” in 132-column mode would be “vt100-w”. The following suffixes should be used where possible:

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-w</td>
<td>Wide mode (more than 80 columns)</td>
<td>vt100-w</td>
</tr>
<tr>
<td>-am</td>
<td>With automatic margins (usually default)</td>
<td>vt100-am</td>
</tr>
<tr>
<td>-nam</td>
<td>Without automatic margins</td>
<td>vt100-nam</td>
</tr>
<tr>
<td>-n</td>
<td>Number of lines on the screen</td>
<td>aaa-60</td>
</tr>
<tr>
<td>-na</td>
<td>No arrow keys (leave them in local)</td>
<td>concept100-na</td>
</tr>
<tr>
<td>-np</td>
<td>Number of pages of memory</td>
<td>concept100-4p</td>
</tr>
<tr>
<td>-rv</td>
<td>Reverse video</td>
<td>concept100-rv</td>
</tr>
</tbody>
</table>

CAPABILITIES
The characters in the Notes field in the table have the following meanings (more than one may apply to a capability):

N  indicates numeric parameter(s)
P  indicates that padding may be specified
*  indicates that padding may be based on the number of lines affected
o  indicates capability is obsolete

“Obsolete” capabilities have no terminfo equivalents, since they were considered useless, or are subsumed by other capabilities. New software should not rely on them at all.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Notes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!1</td>
<td>str</td>
<td></td>
<td>Sent by shifted save key</td>
</tr>
<tr>
<td>!2</td>
<td>str</td>
<td></td>
<td>Sent by shifted suspend key</td>
</tr>
<tr>
<td>!3</td>
<td>str</td>
<td></td>
<td>Sent by shifted undo key</td>
</tr>
<tr>
<td>#1</td>
<td>str</td>
<td></td>
<td>Sent by shifted help key</td>
</tr>
<tr>
<td>#2</td>
<td>str</td>
<td></td>
<td>Sent by shifted home key</td>
</tr>
<tr>
<td>#3</td>
<td>str</td>
<td></td>
<td>Sent by shifted input key</td>
</tr>
<tr>
<td>#4</td>
<td>str</td>
<td></td>
<td>Sent by shifted left-arrow key</td>
</tr>
<tr>
<td>%0</td>
<td>str</td>
<td>Sent by redo key</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>%1</td>
<td>str</td>
<td>Sent by help key</td>
<td></td>
</tr>
<tr>
<td>%2</td>
<td>str</td>
<td>Sent by mark key</td>
<td></td>
</tr>
<tr>
<td>%3</td>
<td>str</td>
<td>Sent by message key</td>
<td></td>
</tr>
<tr>
<td>%4</td>
<td>str</td>
<td>Sent by move key</td>
<td></td>
</tr>
<tr>
<td>%5</td>
<td>str</td>
<td>Sent by next-object key</td>
<td></td>
</tr>
<tr>
<td>%6</td>
<td>str</td>
<td>Sent by open key</td>
<td></td>
</tr>
<tr>
<td>%7</td>
<td>str</td>
<td>Sent by options key</td>
<td></td>
</tr>
<tr>
<td>%8</td>
<td>str</td>
<td>Sent by previous-object key</td>
<td></td>
</tr>
<tr>
<td>%9</td>
<td>str</td>
<td>Sent by print or copy key</td>
<td></td>
</tr>
<tr>
<td>%a</td>
<td>str</td>
<td>Sent by shifted message key</td>
<td></td>
</tr>
<tr>
<td>%b</td>
<td>str</td>
<td>Sent by shifted move key</td>
<td></td>
</tr>
<tr>
<td>%c</td>
<td>str</td>
<td>Sent by shifted next-object key</td>
<td></td>
</tr>
<tr>
<td>%d</td>
<td>str</td>
<td>Sent by shifted options key</td>
<td></td>
</tr>
<tr>
<td>%e</td>
<td>str</td>
<td>Sent by shifted previous-object key</td>
<td></td>
</tr>
<tr>
<td>%f</td>
<td>str</td>
<td>Sent by shifted print or copy key</td>
<td></td>
</tr>
<tr>
<td>%g</td>
<td>str</td>
<td>Sent by shifted redo key</td>
<td></td>
</tr>
<tr>
<td>%h</td>
<td>str</td>
<td>Sent by shifted replace key</td>
<td></td>
</tr>
<tr>
<td>%i</td>
<td>str</td>
<td>Sent by shifted right-arrow key</td>
<td></td>
</tr>
<tr>
<td>%j</td>
<td>str</td>
<td>Sent by shifted resume key</td>
<td></td>
</tr>
<tr>
<td>&amp;0</td>
<td>str</td>
<td>Sent by shifted cancel key</td>
<td></td>
</tr>
<tr>
<td>&amp;1</td>
<td>str</td>
<td>Sent by ref(ERENCE) key</td>
<td></td>
</tr>
<tr>
<td>&amp;2</td>
<td>str</td>
<td>Sent by refresh key</td>
<td></td>
</tr>
<tr>
<td>&amp;3</td>
<td>str</td>
<td>Sent by replace key</td>
<td></td>
</tr>
<tr>
<td>&amp;4</td>
<td>str</td>
<td>Sent by restart key</td>
<td></td>
</tr>
<tr>
<td>&amp;5</td>
<td>str</td>
<td>Sent by resume key</td>
<td></td>
</tr>
<tr>
<td>&amp;6</td>
<td>str</td>
<td>Sent by save key</td>
<td></td>
</tr>
<tr>
<td>&amp;7</td>
<td>str</td>
<td>Sent by suspend key</td>
<td></td>
</tr>
<tr>
<td>&amp;8</td>
<td>str</td>
<td>Sent by undo key</td>
<td></td>
</tr>
<tr>
<td>&amp;9</td>
<td>str</td>
<td>Sent by shifted beg(inning) key</td>
<td></td>
</tr>
<tr>
<td>\a</td>
<td>str</td>
<td>Sent by shifted find key</td>
<td></td>
</tr>
<tr>
<td>\b</td>
<td>str</td>
<td>Sent by shifted cmd (command) key</td>
<td></td>
</tr>
<tr>
<td>\c</td>
<td>str</td>
<td>Sent by shifted copy key</td>
<td></td>
</tr>
<tr>
<td>\d</td>
<td>str</td>
<td>Sent by shifted create key</td>
<td></td>
</tr>
<tr>
<td>\e</td>
<td>str</td>
<td>Sent by shifted delete-char key</td>
<td></td>
</tr>
<tr>
<td>\f</td>
<td>str</td>
<td>Sent by shifted delete-line key</td>
<td></td>
</tr>
<tr>
<td>\g</td>
<td>str</td>
<td>Sent by select key</td>
<td></td>
</tr>
<tr>
<td>\h</td>
<td>str</td>
<td>Sent by shifted end key</td>
<td></td>
</tr>
<tr>
<td>\i</td>
<td>str</td>
<td>Sent by shifted clear-line key</td>
<td></td>
</tr>
<tr>
<td>\j</td>
<td>str</td>
<td>Sent by shifted exit key</td>
<td></td>
</tr>
<tr>
<td>\l</td>
<td>bool</td>
<td>Printer won't echo on screen</td>
<td></td>
</tr>
<tr>
<td>\m</td>
<td>str</td>
<td>Sent by find key</td>
<td></td>
</tr>
<tr>
<td>\n</td>
<td>str</td>
<td>Sent by beg(inning) key</td>
<td></td>
</tr>
<tr>
<td>\o</td>
<td>str</td>
<td>Sent by cancel key</td>
<td></td>
</tr>
<tr>
<td>\p</td>
<td>str</td>
<td>Sent by close key</td>
<td></td>
</tr>
<tr>
<td>\q</td>
<td>str</td>
<td>Sent by cmd (command) key</td>
<td></td>
</tr>
<tr>
<td>\r</td>
<td>str</td>
<td>Sent by copy key</td>
<td></td>
</tr>
<tr>
<td>\s</td>
<td>str</td>
<td>Sent by create key</td>
<td></td>
</tr>
<tr>
<td>\t</td>
<td>str</td>
<td>Sent by end key</td>
<td></td>
</tr>
<tr>
<td>\u</td>
<td>str</td>
<td>Sent by enter/send key (unreliable)</td>
<td></td>
</tr>
<tr>
<td>\v</td>
<td>str</td>
<td>Sent by exit key</td>
<td></td>
</tr>
<tr>
<td>\x</td>
<td>str</td>
<td>Graphic character set pairs aAbBeC – def=VT100</td>
<td></td>
</tr>
</tbody>
</table>
ae  str  (P)  End alternate character set
AL  str  (NP)  Add n new blank lines
al  str  (P)  Add new blank line
am  bool  Terminal has automatic margins
as  str  (P)  Start alternate character set
bc  str  (o)  Backspace if not \H
bl  str  (P)  Audible signal (bell)
bs  bool  (o)  Terminal can backspace with \H
bt  str  (P)  Back tab
bw  bool  le (backspace) wraps from column 0 to last column
cb  str  (P)  Clear to beginning of line, inclusive
CC  str  Terminal settable command character in prototype
cd  str  (P)  Clear to end of display
ce  str  (P)  Clear to end of line
ch  str  (NP)  Set cursor column (horizontal position)
cl  str  (P)  Clear screen and home cursor
CM  str  (NP)  Memory-relative cursor motion to row m, column n
cm  str  (NP)  Screen-relative cursor motion to row m, column n
cn  num  Number of columns in a line
cr  str  (P)  Carriage return
cs  str  (NP)  Change scrolling region to lines m thru n (VT100)
cr  str  (P)  Clear all tab stops
cv  str  (NP)  Set cursor row (vertical position)
da  bool  Display may be retained above the screen
dB  num  (o)  Milliseconds of bs delay needed (default 0)
db  bool  Display may be retained below the screen
DC  str  (NP)  Delete n characters
dC  num  (o)  Milliseconds of er delay needed (default 0)
dc  str  (P)  Delete character
dF  num  (o)  Milliseconds of ff delay needed (default 0)
DL  str  (NP)  Delete n lines
dl  str  (P)  Delete line
dm  str  Enter delete mode
DN  num  (o)  Milliseconds of nl delay needed (default 0)
DO  str  (NP)  Move cursor down n lines
do  str  Down one line
ds  str  Disable status line
dT  num  (o)  Milliseconds of horizontal tab delay needed (default 0)
dV  num  (o)  Milliseconds of vertical tab delay needed (default 0)
eA  str  (P)  Enable graphic character set
eC  str  (NP)  Erase n characters
ed  str  End delete mode
ei  str  End insert mode
eo  bool  Can erase overstrides with a blank
EP  bool  (o)  Even parity
es  bool  Escape can be used on the status line
F1-F9  str  Sent by function keys 11-19
FA-FZ  str  Sent by function keys 20-45
Fa-Fr  str  Sent by function keys 46-63
ff  str  (P)  Hardcopy terminal page eject
fs  str  Return from status line
gn  bool  Generic line type (e.g. dialup, switch)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC</td>
<td>bool</td>
<td>Cursor is hard to see</td>
</tr>
<tr>
<td>hc</td>
<td>bool</td>
<td>Hardcopy terminal</td>
</tr>
<tr>
<td>HD</td>
<td>bool (o)</td>
<td>Half-duplex</td>
</tr>
<tr>
<td>hd</td>
<td>str</td>
<td>Half-line down (forward 1/2 linefeed)</td>
</tr>
<tr>
<td>ho</td>
<td>str  (P)</td>
<td>Home cursor</td>
</tr>
<tr>
<td>hs</td>
<td>bool</td>
<td>Has extra “status line”</td>
</tr>
<tr>
<td>hu</td>
<td>str</td>
<td>Half-line up (reverse 1/2 linefeed)</td>
</tr>
<tr>
<td>hz</td>
<td>bool</td>
<td>Cannot print ‘s (Hazeltine)</td>
</tr>
<tr>
<td>i2</td>
<td></td>
<td>Initialize status line</td>
</tr>
<tr>
<td>i1,i3</td>
<td>str</td>
<td>Terminal initialization strings (terminfo only)</td>
</tr>
<tr>
<td>IC</td>
<td>str  (NP)</td>
<td>Insert n blank characters</td>
</tr>
<tr>
<td>ic</td>
<td>str  (P)</td>
<td>Insert character</td>
</tr>
<tr>
<td>if</td>
<td>str</td>
<td>Name of file containing initialization string</td>
</tr>
<tr>
<td>im</td>
<td>str</td>
<td>Enter insert mode</td>
</tr>
<tr>
<td>in</td>
<td>bool</td>
<td>Insert mode distinguishes nulls</td>
</tr>
<tr>
<td>iP</td>
<td>str</td>
<td>Pathname of program for initialization (terminfo only)</td>
</tr>
<tr>
<td>ip</td>
<td>str  (P)</td>
<td>Insert pad after character inserted</td>
</tr>
<tr>
<td>is</td>
<td>str</td>
<td>Terminal initialization string</td>
</tr>
<tr>
<td>it</td>
<td>num</td>
<td>Tabs initially every n positions</td>
</tr>
<tr>
<td>k;</td>
<td>str</td>
<td>Sent by function key 10</td>
</tr>
<tr>
<td>K1</td>
<td>str</td>
<td>Sent by keypad upper left</td>
</tr>
<tr>
<td>K2</td>
<td>str</td>
<td>Sent by keypad center</td>
</tr>
<tr>
<td>K3</td>
<td>str</td>
<td>Sent by keypad upper right</td>
</tr>
<tr>
<td>K4</td>
<td>str</td>
<td>Sent by keypad lower left</td>
</tr>
<tr>
<td>K5</td>
<td>str</td>
<td>Sent by keypad lower right</td>
</tr>
<tr>
<td>k0-k9</td>
<td>str</td>
<td>Sent by function keys 0-9</td>
</tr>
<tr>
<td>kA</td>
<td>str</td>
<td>Sent by insert-line key</td>
</tr>
<tr>
<td>ka</td>
<td>str</td>
<td>Sent by clear-all-tabs key</td>
</tr>
<tr>
<td>kB</td>
<td>str</td>
<td>Sent by back-tab key</td>
</tr>
<tr>
<td>kb</td>
<td>str</td>
<td>Sent by backspace key</td>
</tr>
<tr>
<td>kC</td>
<td>str</td>
<td>Sent by clear-screen or erase key</td>
</tr>
<tr>
<td>kD</td>
<td>str</td>
<td>Sent by delete-character key</td>
</tr>
<tr>
<td>kd</td>
<td>str</td>
<td>Sent by down-arrow key</td>
</tr>
<tr>
<td>kE</td>
<td>str</td>
<td>Sent by clear-to-end-of-line key</td>
</tr>
<tr>
<td>ke</td>
<td>str</td>
<td>Out of “keypad transmit” mode</td>
</tr>
<tr>
<td>kF</td>
<td>str</td>
<td>Sent by scroll-forward/down key</td>
</tr>
<tr>
<td>kH</td>
<td>str</td>
<td>Sent by home-down key</td>
</tr>
<tr>
<td>kh</td>
<td>str</td>
<td>Sent by home key</td>
</tr>
<tr>
<td>kI</td>
<td>str</td>
<td>Sent by insert-character or enter-insert-mode key</td>
</tr>
<tr>
<td>kL</td>
<td>str</td>
<td>Sent by delete-line key</td>
</tr>
<tr>
<td>kl</td>
<td>str</td>
<td>Sent by left-arrow key</td>
</tr>
<tr>
<td>kM</td>
<td>str</td>
<td>Sent by insert key while in insert mode</td>
</tr>
<tr>
<td>km</td>
<td>bool</td>
<td>Has a “meta” key (shift, sets parity bit)</td>
</tr>
<tr>
<td>kN</td>
<td>str</td>
<td>Sent by next-page key</td>
</tr>
<tr>
<td>kn</td>
<td>num (o)</td>
<td>Number of function (k0–k9) keys (default 0)</td>
</tr>
<tr>
<td>ko</td>
<td>str  (o)</td>
<td>termcap entries for other non-function keys</td>
</tr>
<tr>
<td>kP</td>
<td>str</td>
<td>Sent by previous-page key</td>
</tr>
<tr>
<td>kR</td>
<td>str</td>
<td>Sent by scroll-backward/up key</td>
</tr>
<tr>
<td>kr</td>
<td>str</td>
<td>Sent by right-arrow key</td>
</tr>
<tr>
<td>kS</td>
<td>str</td>
<td>Sent by clear-to-end-of-screen key</td>
</tr>
<tr>
<td>ks</td>
<td>str</td>
<td>Put terminal in “keypad transmit” mode</td>
</tr>
<tr>
<td>kT</td>
<td>str</td>
<td>Sent by set-tab key</td>
</tr>
</tbody>
</table>
kt  str  Sent by clear-tab key
ku  str  Sent by up-arrow key
la  str  Label on function key 10 if not f10
l0-l9 str  Labels on function keys 0-9 if not f0-f9
LC bool (o)  Lower-case only
LE str (NP)  Move cursor left n positions
le str (P)  Move cursor left one position
LF str (P)  Turn off soft labels
lh num  Number of rows in each label
li num  Number of lines on screen or page
ll str  Last line, first column
lm num  Lines of memory if > ll (0 means varies)
LO str (P)  Turn on soft labels
lw num  Number of columns in each label
ma str (o)  Arrow key map (used by vi version 2 only)
mb str  Turn on blinking attribute
MC str (P)  Clear left and right soft margins
md str  Turn on bold (extra bright) attribute
me str  Turn off all attributes
mh str  Turn on half-bright attribute
mi bool  Safe to move while in insert mode
mk str  Turn on blank attribute (characters invisible)
ML str (P)  Set soft left margin
ml str (o)  Memory lock on above cursor
mm str  Turn on “meta mode” (8th bit)
mo str  Turn off “meta mode”
mp str  Turn on protected attribute
MR str (P)  Set soft right margin
mr str  Turn on reverse-video attribute
ms bool  Safe to move in standout modes
mu str (o)  Memory unlock (turn off memory lock)
nc bool (o)  No correctly-working cr (Datamedia 2500, Hazeltine 2000)
nd str  Non-destructive space (cursor right)
NL bool (o) \n is newline, not line feed
NI num  Number of labels on screen (start at 1)
nl str (o)  Newline character if not \n
NP bool  Pad character doesn’t exist
NR bool  ti does not reverse te
ns bool (o)  Terminal is a CRT but doesn’t scroll
nw str (P)  Newline (behaves like cr followed by do)
nx bool  Padding won’t work, xoff/xon required
OP bool (o)  Odd parity
os bool  Terminal overstrikes
pb num  Lowest baud where delays are required
pc str  Pad character (default NUL)
pf str  Turn off the printer
pk str  Program function key n to type string s (terminfo only)
pl str  Program function key n to execute string s (terminfo only)
pn str (NP)  Program label n to show string s (terminfo only)
pO str (N)  Turn on the printer for n bytes
po str  Turn on the printer
ps str  Print contents of the screen
pt  bool  (o)  Has hardware tabs (may need to be set with is)
px  str   Program function key n to transmit string s (terminfo only)
r1,r3 str   Reset terminal completely to sane modes (terminfo only)
RA str   (P)  Turn off automatic margins
rc str   (P)  Restore cursor to position of last sc
RF str   Send next input character (for pty's)
rf str   Name of file containing reset string
RI str   (NP)  Move cursor right n positions
rP str   (P)  Like ip but when in replace mode
rp str   (NP+) Repeat character c n times
rs str   Reset terminal completely to sane modes
RX str   (P)  Turn off xoff/xon handshaking
SA str   (P)  Turn on automatic margins
sa str   (NP)  Define the video attributes (9 parameters)
sc str   (P)  Save cursor position
se str   End standout mode
SF str   (NP+) Scroll forward n lines
sf str   (P)  Scroll text up
sg num   Number of garbage chars left by so or se (default 0)
so str   Begin standout mode
SR str   (NP+) Scroll backward n lines
sr str   (P)  Scroll text down
st str   Set a tab in all rows, current column
SX str   (P)  Turn on xoff/xon handshaking
ta str   (P)  Tab to next 8-position hardware tab stop
tc str   Entry of similar terminal – must be last
te str   String to end programs that use termcap
ti str   String to begin programs that use termcap
ts str   (N)  Go to status line, column n
UC bool  (o)  Upper-case only
uc str   Underscore one character and move past it
ue str   End underscore mode
ug num   Number of garbage chars left by us or ue (default 0)
ul bool  Underline character overstrikes
UP str   (NP+) Move cursor up n lines
up str   Upline (cursor up)
us str   Start underscore mode
vb str   Visible bell (must not move cursor)
ve str   Make cursor appear normal (undo vs/vi)
vi str   Make cursor invisible
vs str   Make cursor very visible
vt num   Virtual terminal number (not supported on all systems)
wi str   (N)  Set current window to lines i thru j, columns m thru n
ws num   Number of columns in status line
xb bool  Beehive (f1=ESC, f2="C")
XF str   X-off character (default DC3)
XN str   X-on character (default DC1)
xn bool  Newline ignored after 80 cols (Concept)
xo bool  Terminal uses xoff/xon handshaking
xr bool  (o)  Return acts like ee cr nl (Delta Data)
xs bool  Standout not erased by overwriting (Hewlett-Packard)
xr bool  Tabs destructive, magic so char (Teleray 1061)
A Sample Entry

The following entry, which describes the Concept-100, is among the more complex entries in the termcap file as of this writing.

```
xx bool (o) Tektronix 4025 insert-line
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be included for readability (here between the last field on a line and the first field on the next). Comments may be included on lines beginning with ‘#’.

Types of Capabilities

Capabilities in termcap are of three types: Boolean capabilities, which indicate particular features that the terminal has; numeric capabilities, giving the size of the display or the size of other attributes; and string capabilities, which give character sequences that can be used to perform particular terminal operations. All capabilities have two-letter codes. For instance, the fact that the Concept has automatic margins (i.e., an automatic return and linefeed when the end of a line is reached) is indicated by the Boolean capability am. Hence the description of the Concept includes am.

Numeric capabilities are followed by the character ‘#’ then the value. In the example above co, which indicates the number of columns the display has, gives the value ‘80’ for the Concept.

Finally, string-valued capabilities, such as ce (clear-to-end-of-line sequence) are given by the two-letter code, an ‘s’, then a string ending at the next following ‘?’. A delay in milliseconds may appear after the ‘s’ in such a capability, which causes padding characters to be supplied by tputs after the remainder of the string is sent to provide this delay. The delay can be either a number, e.g. ‘20’, or a number followed by an ‘s’, i.e., ‘3s’. An ‘s’ indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-line padding required. (In the case of insert-character, the factor is still the number of lines affected; this is always 1 unless the terminal has in and the software uses it.) When an ‘s’ is specified, it is sometimes useful to give a delay of the form ‘3.5’ to specify a delay per line to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string-valued capabilities for easy encoding of control characters there. \E maps to an esc character, \X maps to a control-X for any appropriate X, and the sequences \n \r \t \b \v map to linefeed, return, tab, backspace, and formfeed, respectively. Finally, characters may be given as three octal digits after a \, and the characters ‘ ‘ \ and \ are given as ‘ ‘ and ‘ ‘. If it is necessary to place a : in a capability it must be escaped in octal as \072. If it is necessary to place a nul character in a string capability it must be encoded as \200. (The routines that deal with termcap use C strings and strip the high bits of the output very late, so that a \200 comes out as a \000 would.)
Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the first er and ta in the example above.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in termcap and to build up a description gradually, using partial descriptions with vi to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the termcap file to describe it or bugs in vi. To easily test a new terminal description you can set the environment variable TERM to the absolute pathname of a file containing the description you are working on and programs will look there rather than in /etc/termcap. TERM can also be set to the termcap entry itself to avoid reading the file when starting up a program.

To get the padding for insert-line right (if the terminal manufacturer did not document it), a severe test is to use vi to edit /etc/passwd at 9600 baud, delete roughly 16 lines from the middle of the screen, then hit the ‘u’ key several times quickly. If the display messes up, more padding is usually needed. A similar test can be used for insert-character.

Basic Capabilities

The number of columns on each line of the display is given by the co numeric capability. If the display is a CRT, then the number of lines on the screen is given by the li capability. If the display wraps around to the beginning of the next line when the cursor reaches the right margin, then it should have the am capability. If the terminal can clear its screen, the code to do this is given by the cl string capability. If the terminal overstrikes (rather than clearing the position when a character is overwritten), it should have the os capability. If the terminal is a printing terminal, with no soft copy unit, give it both he and os. (os applies to storage scope terminals, such as the Tektronix 4010 series, as well as to hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as cr. (Normally this will be carriage-return, ‘M.’) If there is a code to produce an audible signal (bell, beep, etc.), give this as bl.

If there is a code (such as backspace) to move the cursor one position to the left, that capability should be given as le. Similarly, codes to move to the right, up, and down should be given as nd, up, and do, respectively. These local cursor motions should not alter the text they pass over; for example, you would not normally use “nd= ” unless the terminal has the os capability, because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in termcap have undefined behavior at the left and top edges of a CRT display. Programs should never attempt to backspace around the left edge, unless bw is given, and never attempt to go up off the top using local cursor motions.

In order to scroll text up, a program goes to the bottom left corner of the screen and sends the sf (index) string. To scroll text down, a program goes to the top left corner of the screen and sends the sr (reverse index) string. The strings sf and sr have undefined behavior when not on their respective corners of the screen. Parameterized versions of the scrolling sequences are SF and SR, which have the same semantics as sf and sr except that they take one parameter and scroll that many lines. They also have undefined behavior except at the appropriate corner of the screen.

The am capability tells whether the cursor sticks at the right edge of the screen when text is output there, but this does not necessarily apply to nd from the last column. Leftward local motion is defined from the left edge only when bw is given; then an le from the left edge will move to the right edge of the previous row. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch-selectable automatic margins, the termcap description usually assumes that this feature is on, i.e., am. If the terminal has a
command that moves to the first column of the next line, that command can be given as \texttt{nw} (newline). It is permissible for this to clear the remainder of the current line, so if the terminal has no correctly-working \texttt{CR} and \texttt{LF} it may still be possible to craft a working \texttt{nw} out of one or both of them.

These capabilities suffice to describe hardcopy and "glass-tty" terminals. Thus the Teletype model 33 is described as

\begin{verbatim}
T3 |tty33 |33 |tty |Teletype model 33:\n :bl=\texttt{G}:co#72:cr=\texttt{M}:do=\texttt{J}:hc=\texttt{os}:
\end{verbatim}

and the Lear Siegler ADM–3 is described as

\begin{verbatim}
I3 |adm3 |3 |LSI ADM-3:\n :am:bl=\texttt{G}:cl="Z.co#80:cr=\texttt{M}:do=\texttt{J}:le=\texttt{H}:li#24:sf=\texttt{J}:
\end{verbatim}

Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with \texttt{printf} (3S)-like escapes \texttt{%x} in it, while other characters are passed through unchanged. For example, to address the cursor the \texttt{em} capability is given, using two parameters: the row and column to move to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory. If the terminal has memory-relative cursor addressing, that can be indicated by an analogous \texttt{CM} capability.)

The \texttt{%} encodings have the following meanings:

\begin{verbatim}
\%%     output '\n'
\%d     output value as in \texttt{printf} \%d
\%2     output value as in \texttt{printf} \%2d
\%3     output value as in \texttt{printf} \%3d
\%c     output value as in \texttt{printf} \%c
\%+x    add \texttt{x} to value, then do \%.
\%>xy   if value > \texttt{x} then add \texttt{y}, no output
\%r     reverse order of two parameters, no output
\%i     increment by one, no output
\%n     exclusive-or all parameters with 0140 (Datamedia 2500)
\%B     BCD (16*(value/10)) + (value\%10), no output
\%D     Reverse coding (value – 2*(value\%16)), no output (Delta Data)
\end{verbatim}

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent "\texttt{\&E\&a12c03Y}" padded for 6 milliseconds. Note that the order of the row and column coordinates is reversed here and that the row and column are sent as two-digit integers. Thus its \texttt{em} capability is "\texttt{cm=6\&\&r\%2c\%2Y}".

The Microterm ACT-IV needs the current row and column sent simply encoded in binary preceded by a \texttt{\textbackslash T}, "\texttt{T\%\%}". Terminals that use "\texttt{\.}" need to be able to backspace the cursor (\texttt{le}) and to move the cursor up one line on the screen (\texttt{up}). This is necessary because it is not always safe to transmit \texttt{\textbackslash n}, \texttt{\textbackslash D}, and \texttt{\textbackslash r}, as the system may change or discard them. (Programs using \texttt{termcap} must set terminal modes so that tabs are not expanded, so \textbackslash \texttt{t} is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the Lear Siegler ADM–3a, which offsets row and column by a blank character, thus "\texttt{em=\textbackslash E=+- +%+}".

Row or column absolute cursor addressing can be given as single parameter capabilities \texttt{ch} (horizontal position absolute) and \texttt{cv} (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to \texttt{em}. If there are parameterized local motions (\textit{e.g.,} move \texttt{n} positions to the right) these can be given as \texttt{DO}, \texttt{LE}, \texttt{RI}, and \texttt{UP} with a single parameter indicating how
many positions to move. These are primarily useful if the terminal does not have \texttt{em}, such as the Tektronix 4025.

**Cursor Motions**

If the terminal has a fast way to home the cursor (to the very upper left corner of the screen), this can be given as \texttt{ho}. Similarly, a fast way of getting to the lower left-hand corner can be given as \texttt{ll}; this may involve going up with \texttt{up} from the home position, but a program should never do this itself (unless \texttt{ll} does), because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as cursor address \((0,0)\): to the top left corner of the screen, not of memory. (Therefore, the \texttt{\textbackslash AEH} sequence on Hewlett-Packard terminals cannot be used for \texttt{ho}.)

**Area Clears**

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as \texttt{ce}. If the terminal can clear from the current position to the end of the display, this should be given as \texttt{ed}. \texttt{ed} must only be invoked from the first column of a line. (Therefore, it can be simulated by a request to delete a large number of lines, if a true \texttt{ed} is not available.)

**Insert/Delete Line**

If the terminal can open a new blank line before the line containing the cursor, this should be given as \texttt{ai}; this must be invoked only from the first position of a line. The cursor must then appear at the left of the newly blank line. If the terminal can delete the line that the cursor is on, this should be given as \texttt{dl}; this must only be used from the first position on the line to be deleted. Versions of \texttt{ai} and \texttt{dl} which take a single parameter and insert or delete that many lines can be given as \texttt{AI} and \texttt{DL}. If the terminal has a settable scrolling region (like the VT100), the command to set this can be described with the \texttt{es} capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command -- the \texttt{sc} and \texttt{re} (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using \texttt{sr} or \texttt{sf} on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory which all commands affect, it should be given as the parameterized string \texttt{wi}. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order. (This \texttt{terminfo} capability is described for completeness. It is unlikely that any \texttt{termcap}-using program will support it.)

If the terminal can retain display memory above the screen, then the \texttt{da} capability should be given; if display memory can be retained below, then \texttt{db} should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with \texttt{sr} may bring down non-blank lines.

**Insert/Delete Character**

There are two basic kinds of intelligent terminals with respect to insert/delete character that can be described using \texttt{termcap}. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept-100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen then typing text separated by cursor motions. Type \texttt{"abc \ def"} using local cursor motions (not spaces) between the \texttt{"abc"} and the \texttt{"def"}. Then position the cursor before the \texttt{"abc"} and put the
terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the “abc” shifts over to the “def” which then move together around the end of the current line and onto the next as you insert, then you have the second type of terminal and should give the capability in, which stands for “insert null”. While these are two logically separate attributes (one line vs. multi-line insert mode, and special treatment of untyped spaces), we have seen no terminals whose insert mode cannot be described with the single attribute.

termcap can describe both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Give as im the sequence to get into insert mode. Give as ei the sequence to leave insert mode. Now give as ic any sequence that needs to be sent just before each character to be inserted. Most terminals with a true insert mode will not give ic; terminals that use a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to ic. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds in ip (a string option). Any other sequence that may need to be sent after insertion of a single character can also be given in ip. If your terminal needs to be placed into an ‘insert mode’ and needs a special code preceding each inserted character, then both im/ei and ic can be given, and both will be used. The IC capability, with one parameter n, will repeat the effects of ic n times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode, you can give the capability mi to speed up inserting in this case. Omitting mi will affect only speed. Some terminals (notably Datamediа's) must not have mi because of the way their insert mode works.

Finally, you can specify dc to delete a single character, DC with one parameter n to delete n characters, and delete mode by giving dm and ed to enter and exit delete mode (which is any mode the terminal needs to be placed in for dc to work).

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as standout mode, representing a good high-contrast, easy-on-the-eyes format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as so and se, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces or garbage characters on the screen, as the TVI 912 and Teleray 1061 do, then sg should be given to tell how many characters are left.

Codes to begin underlining and end underlining can be given as us and ue, respectively. Underline mode change garbage is specified by ug, similar to sg. If the terminal has a code to underline the current character and move the cursor one position to the right, such as the Microterm Mime, this can be given as uc.

Other capabilities to enter various highlighting modes include mb (blinking), md (bold or extra bright), mh (dim or half-bright), mk (blanking or invisible text), mp (protected), mr (reverse video), me (turn off all attribute modes), as (enter alternate character set mode), and ae (exit alternate character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of mode, this should be given as sa (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attributes is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim,
bold, blank, protect, and alternate character set. Not all modes need be supported by sa, only those for which corresponding attribute commands exist. (It is unlikely that a termcap-using program will support this capability, which is defined for compatibility with terminfo.)

Terminals with the “magic cookie” glitches (sg and ug), rather than maintaining extra attribute bits for each character cell, instead deposit special “cookies”, or “garbage characters”, when they receive mode-setting sequences, which affect the display algorithm.

Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or when the cursor is addressed. Programs using standout mode should exit standout mode on such terminals before moving the cursor or sending a newline. On terminals where this is not a problem, the ms capability should be present to say that this overhead is unnecessary.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), this can be given as vb; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to change, for example, a non-blinking underline into an easier-to-find block or blinking underline), give this sequence as vs. If there is a way to make the cursor completely invisible, give that as vi. The capability ve, which undoes the effects of both of these modes, should also be given.

If your terminal correctly displays underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability ul. If overstrikes are erasable with a blank, this should be indicated by giving eo.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local mode (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as ks and ke. Otherwise the keypad is assumed to always transmit. The codes sent by the left-arrow, right-arrow, up-arrow, down-arrow, and home keys can be given as kl, kr, ku, kd, and kh, respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as k0, k1, k9. If these keys have labels other than the default f0 through f9, the labels can be given as l0, l1, l9. The codes transmitted by certain other special keys can be given: kH (home down), kb (backspace), ka (clear all tabs), kt (clear the tab stop in this column), kC (clear screen or erase), kD (delete character), kL (delete line), kM (exit insert mode), kE (clear to end of line), kS (clear to end of screen), kI (insert character or enter insert mode), kA (insert line), kN (next page), kP (previous page), kF (scroll forward/down), kR (scroll backward/up), and kT (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, then the other five keys can be given as k1, k2, k3, k4, and k5. These keys are useful when the effects of a 3 by 3 directional pad are needed. The obsolete ko capability formerly used to describe “other” function keys has been completely supplanted by the above capabilities.

The ma entry is also used to indicate arrow keys on terminals that have single-character arrow keys. It is obsolete but still in use in version 2 of vi which must be run on some minicomputers due to memory limitations. This field is redundant with kl, kr, ku, kd, and kh. It consists of groups of two characters. In each group, the first character is what an arrow key sends, and the second character is the corresponding vi command. These commands are h for kl, j for kd, k for ku, l for kr, and H for kh. For example, the Mime would have “ma="H" "kJ" "Zk" "Xl"” indicating arrow keys left ('H), down ('K), up ('Z), and right ('X). (There is no home key on the Mime.)
Tabs and Initialization

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as \texttt{ti} and \texttt{te}. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory-relative cursor addressing and not screen-relative cursor addressing, a screen-sized window must be fixed into the display for cursor addressing to work properly. This is also used for the Tektronix 4025, where \texttt{ti} sets the command character to be the one used by \texttt{termcap}.

Other capabilities include \texttt{ls}, an initialization string for the terminal, and \texttt{if}, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the \texttt{termcap} description. They are normally sent to the terminal by the \texttt{tset} program each time the user logs in. They will be printed in the following order: \texttt{is}; setting tabs using \texttt{et} and \texttt{st}; and finally \texttt{if}. (\texttt{Terminfo} uses \texttt{ib} before \texttt{is} and runs the program \texttt{ip} and prints \texttt{i3} after the other initializations.) A pair of sequences that does a harder reset from a totally unknown state can be analogously given as \texttt{rs} and \texttt{if}. These strings are output by the \texttt{reset} program, which is used when the terminal gets into a wedged state. (\texttt{Terminfo} uses \texttt{ri} before \texttt{rs} and \texttt{r3} after.) Commands are normally placed in \texttt{rs} and \texttt{rf} only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the VT100 into 80-column mode would normally be part of \texttt{is}, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80-column mode.

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as \texttt{ta} (usually \texttt{1}). A “backtab” command which moves leftward to the previous tab stop can be given as \texttt{bt}. By convention, if the terminal driver modes indicate that tab stops are being expanded by the computer rather than being sent to the terminal, programs should not use \texttt{ta} or \texttt{bt} even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs that are initially set every \texttt{n} positions when the terminal is powered up, then the numeric parameter \texttt{it} is given, showing the number of positions between tab stops. This is normally used by the \texttt{tset} command to determine whether to set the driver mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the \texttt{termcap} description can assume that they are properly set.

If there are commands to set and clear tab stops, they can be given as \texttt{ct} (clear all tab stops) and \texttt{st} (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in \texttt{is} or \texttt{if}.

Delays

Certain capabilities control padding in the terminal driver. These are primarily needed by hardcopy terminals and are used by the \texttt{tset} program to set terminal driver modes appropriately. Delays embedded in the capabilities \texttt{cr}, \texttt{sf}, \texttt{le}, \texttt{ff}, and \texttt{ta} will cause the appropriate delay bits to be set in the terminal driver. If \texttt{pb} (padding baud rate) is given, these values can be ignored at baud rates below the value of \texttt{pb}. For 4.2BSD \texttt{tset}, the delays are given as numeric capabilities \texttt{dC}, \texttt{dN}, \texttt{dB}, \texttt{dF}, and \texttt{dT} instead.

Miscellaneous

If the terminal requires other than a \texttt{NUL} (zero) character as a pad, this can be given as \texttt{pe}. Only the first character of the \texttt{pe} string is used.

If the terminal has commands to save and restore the position of the cursor, give them as \texttt{se} and \texttt{re}.
If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, then the capability \texttt{hs} should be given. Special strings to go to a position in the status line and to return from the status line can be given as \texttt{ts} and \texttt{fs}. (\texttt{fs} must leave the cursor position in the same place that it was before \texttt{ts}. If necessary, the \texttt{sc} and \texttt{re} strings can be included in \texttt{ts} and \texttt{fs} to get this effect.) The capability \texttt{ts} takes one parameter, which is the column number of the status line to which the cursor is to be moved. If escape sequences and other special commands such as \texttt{tab} work while in the status line, the flag \texttt{es} can be given. A string that turns off the status line (or otherwise erases its contents) should be given as \texttt{ds}. The status line is normally assumed to be the same width as the rest of the screen, \textit{i.e.}, \texttt{co}. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded), then its width in columns can be indicated with the numeric parameter \texttt{ws}.

If the terminal can move up or down half a line, this can be indicated with \texttt{hu} (half-line up) and \texttt{hd} (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as \texttt{ff} (usually \texttt{L}).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters), this can be indicated with the parameterized string \texttt{rp}. The first parameter is the character to be repeated and the second is the number of times to repeat it. (This is a \textit{terminfo} feature that is unlikely to be supported by a program that uses \texttt{termcap}.)

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with \texttt{CC}. A prototype command character is chosen which is used in all capabilities. This character is given in the \texttt{CC} capability to identify it. The following convention is supported on some UNIX systems: The environment is to be searched for a \texttt{CC} variable, and if found, all occurrences of the prototype character are replaced by the character in the environment variable. This use of the \texttt{CC} environment variable is a very bad idea, as it conflicts with \texttt{make}(1).

Terminal descriptions that do not represent a specific kind of known terminal, such as \texttt{switch}, \texttt{dialup}, \texttt{patch}, and \texttt{network}, should include the \texttt{gn} (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to \texttt{virtual} terminal descriptions for which the escape sequences are known.)

If the terminal uses \texttt{xoff}/\texttt{xon} (\texttt{DC3}/\texttt{DC1}) handshaking for flow control, give \texttt{xo}. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, then this fact can be indicated with \texttt{km}. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as \texttt{mm} and \texttt{mo}.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with \texttt{lm}. An explicit value of 0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the UNIX system \textit{virtual} terminal protocol, the terminal number can be given as \texttt{vt}.

Media copy strings which control an auxiliary printer connected to the terminal can be given as \texttt{ps}: print the contents of the screen; \texttt{pf}: turn off the printer; and \texttt{po}: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation \texttt{pO} takes one parameter and leaves the printer on for as many characters as the value of the
parameter, then turns the printer off. The parameter should not exceed 255. All text, including `pf`, is transparently passed to the printer while `po` is in effect.

Strings to program function keys can be given as `pk`, `pl`, and `px`. Each of these strings takes two parameters: the function key number to program (from 0 to 9) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The differences among the capabilities are that `pk` causes pressing the given key to be the same as the user typing the given string; `pl` causes the string to be executed by the terminal in local mode; and `px` causes the string to be transmitted to the computer. Unfortunately, due to lack of a definition for string parameters in `termcap`, only `terminfo` supports these capabilities.

**Glitches and Braindamage**

Hazeltine terminals, which do not allow """" characters to be displayed, should indicate `hz`.

The `nc` capability, now obsolete, formerly indicated Datamedia terminals, which echo `\r `n for carriage return then ignore a following linefeed.

Terminals that ignore a linefeed immediately after an `am` wrap, such as the Concept, should indicate `xn`.

If `ce` is required to get rid of standout (instead of merely writing normal text on top of it), `xs` should be given.

Teleray terminals `w`ler `labs turn all characters moved over to blanks, should indicate `xt` (destructive tabs). This glitch is also taken to mean that it is not possible to position the cursor on top of a “magic cookie”, and that to erase standout mode it is necessary to use delete and insert line.

The Beehive Superbee, which is unable to correctly transmit the `esc` or `C` characters, has `xb`, indicating that the “f1” key is used for `esc` and “f2” for `C`. (Only certain Superbees have this problem, depending on the ROM.)

Other specific terminal problems may be corrected by adding more capabilities of the form `xx`.

**Similar Terminals**

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability `tc` can be given with the name of the similar terminal. This capability must be `last`, and the combined length of the entries must not exceed 1024. The capabilities given before `tc` override those in the terminal type invoked by `tc`. A capability can be canceled by placing `xx@` to the left of the `tc` invocation, where `xx` is the capability. For example, the entry

```
hn |2621–nl:ks@:ke@:tc=2621:
```

defines a “2621–nl” that does not have the `ks` or `ke` capabilities, hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

**AUTHOR**

William Joy

Mark Horton added underlining and keypad support

**FILES**

`/etc/termcap` file containing terminal descriptions

**SEE ALSO**

`ex(1)`, `tset(1)`, `vi(1)`, `curses(3X)`, `printf(3S)`, `term(5)`.
CAVEATS AND BUGS

Note: termcap was replaced by terminfo in UNIX System V Release 2.0. The transition will be relatively painless if capabilities flagged as “obsolete” are avoided.

Vi allows only 256 characters for string capabilities, and the routines in termlib(3) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

Not all programs support all entries.