UNIX Programmer's Reference Manual (PRM)

4.3 Berkeley Software Distribution: Virtual VAX-11 Version

April, 1986

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, California 94720
(PRМ)

4.3 Berkeley Software Distribution
Virtual VAX-11 Version

April, 1986

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, California 94720
### TABLE OF CONTENTS

**2. System Calls**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intro</td>
<td>introduction to system calls and error numbers</td>
</tr>
<tr>
<td>accept</td>
<td>accept a connection on a socket</td>
</tr>
<tr>
<td>access</td>
<td>determine accessibility of file</td>
</tr>
<tr>
<td>acct</td>
<td>turn accounting on or off</td>
</tr>
<tr>
<td>adjtime</td>
<td>correct the time to allow synchronization of the system clock</td>
</tr>
<tr>
<td>bind</td>
<td>bind a name to a socket</td>
</tr>
<tr>
<td>brk</td>
<td>change data segment size</td>
</tr>
<tr>
<td>chdir</td>
<td>change current working directory</td>
</tr>
<tr>
<td>chmod</td>
<td>change mode of file</td>
</tr>
<tr>
<td>chown</td>
<td>change owner and group of a file</td>
</tr>
<tr>
<td>chroot</td>
<td>change root directory</td>
</tr>
<tr>
<td>close</td>
<td>delete a descriptor</td>
</tr>
<tr>
<td>connect</td>
<td>initiate a connection on a socket</td>
</tr>
<tr>
<td>creat</td>
<td>create a new file</td>
</tr>
<tr>
<td>dup</td>
<td>duplicate a descriptor</td>
</tr>
<tr>
<td>execve</td>
<td>execute a file</td>
</tr>
<tr>
<td>exit</td>
<td>terminate a process</td>
</tr>
<tr>
<td>fcntl</td>
<td>file control</td>
</tr>
<tr>
<td>flock</td>
<td>apply or remove an advisory lock on an open file</td>
</tr>
<tr>
<td>fork</td>
<td>create a new process</td>
</tr>
<tr>
<td>fsync</td>
<td>synchronize a file's in-core state with that on disk</td>
</tr>
<tr>
<td>getdirent</td>
<td>gets directory entries in a filesystem independent format</td>
</tr>
<tr>
<td>getdomainname</td>
<td>get/set name of current domain</td>
</tr>
<tr>
<td>getdtablesize</td>
<td>get descriptor table size</td>
</tr>
<tr>
<td>getgid</td>
<td>get group identity</td>
</tr>
<tr>
<td>getgroups</td>
<td>get group access list</td>
</tr>
<tr>
<td>gethostid</td>
<td>get/set unique identifier of current host</td>
</tr>
<tr>
<td>gethostname</td>
<td>get/set name of current host</td>
</tr>
<tr>
<td>getitimer</td>
<td>get/set value of interval timer</td>
</tr>
<tr>
<td>getpagesize</td>
<td>get system page size</td>
</tr>
<tr>
<td>getpeername</td>
<td>get name of connected peer</td>
</tr>
<tr>
<td>getpgid</td>
<td>get process group</td>
</tr>
<tr>
<td>getpid</td>
<td>get process identification</td>
</tr>
<tr>
<td>getpriority</td>
<td>get/set program scheduling priority</td>
</tr>
<tr>
<td>getrlimit</td>
<td>control maximum system resource consumption</td>
</tr>
<tr>
<td>getrusage</td>
<td>get information about resource utilization</td>
</tr>
<tr>
<td>getsockname</td>
<td>get socket name</td>
</tr>
<tr>
<td>getsockopt</td>
<td>get and set options on sockets</td>
</tr>
<tr>
<td>gettimeofday</td>
<td>get/set date and time</td>
</tr>
<tr>
<td>getuid</td>
<td>get user identity</td>
</tr>
<tr>
<td>ioctl</td>
<td>control device</td>
</tr>
<tr>
<td>kill</td>
<td>send signal to a process</td>
</tr>
<tr>
<td>killpg</td>
<td>send signal to a process group</td>
</tr>
<tr>
<td>link</td>
<td>make a hard link to a file</td>
</tr>
<tr>
<td>listen</td>
<td>listen for connections on a socket</td>
</tr>
<tr>
<td>lseek</td>
<td>move read/write pointer</td>
</tr>
<tr>
<td>mkdir</td>
<td>make a directory file</td>
</tr>
<tr>
<td>mknod</td>
<td>make a special file</td>
</tr>
<tr>
<td>mount</td>
<td>mount file system</td>
</tr>
<tr>
<td>nfssvc</td>
<td>NFS daemons</td>
</tr>
<tr>
<td>open</td>
<td>open a file for reading or writing, or create a new file</td>
</tr>
<tr>
<td>pipe</td>
<td>create an interprocess communication channel</td>
</tr>
<tr>
<td>prof</td>
<td>execution time profile</td>
</tr>
</tbody>
</table>

4.3BSD

June 1986
ptrace ................................. process trace
quota ................................. manipulate disk quotas
quotactl .............................. manipulate disk quotas
read .................................. read input
readlink .............................. read value of a symbolic link
reboot ................................ reboot system or halt processor
recv .................................. receive a message from a socket
rename ................................ change the name of a file
rmdir ................................ remove a directory socket
select ................................ synchronous I/O multiplexing
send .................................. send a message from a socket
setgroups ............................. set group access list
setpgrp ............................... set process group
setquota .............................. enable/disable quotas on a file system
setregid ................................ set real and effective group ID
setreuid ................................ set real and effective user ID’s
shutdown .............................. shut down part of a full-duplex connection
sigblock ................................ block signals
sigpause .............................. atomically release blocked signals and wait for interrupt
sigreturn ............................. return from signal
sigsetmask ........................... set current signal mask
sigstack .............................. set and/or get signal stack context
signest ............................... software signal facilities
socket ................................ create an endpoint for communication
socketpair .......................... create a pair of connected sockets
stat ................................... get file status
statfs ................................ get file system statistics
swapon ................................ add a swap device for interleaved paging/swapping
symlink ............................... make symbolic link to a file
sync .................................. update super-block
syscall ............................... indirect system call
truncate .............................. truncate a file to a specified length
umask ................................ set file creation mode mask
unlink ................................. remove directory entry
unmount ............................... remove a file system
utimes ................................ set file times
vfork .................................. spawn new process in a virtual memory efficient way
vhangup ............................... virtually “hangup” the current control terminal
wait ................................... wait for process to terminate
write .................................. write output

3. C Library Subroutines

intro ................................ introduction to C library functions
abort .................................. generate a fault
abs .................................... integer absolute value
alarm .................................. schedule signal after specified time
asinh .................................. inverse hyperbolic functions
assert .................................. program verification
atof .................................. convert ASCII to numbers
bstring ............................... bit and byte string operations
byteorder ............................. convert values between host and network byte order
crypt .................................. DES encryption
cftime ................................ convert date and time to ASCII
cctype ................................ character classification macros
curses ................................. screen functions with “optimal” cursor motion
dbm ...................................... data base subroutines
directory
ecvt
end
erf
ether
execl
exit
exp
fclose
ferror
floor
fopen
fread
frexp
fseek
getc
getdisk
getenv
getfsent
getgrent
gethostbyname
getlogin
getmntent
getnetent
getnetgrent
getopt
getpass
getprotoent
getpw
getpwent
getrpcent
getrpcport
gets
getservent
gettyent
getusershell
getwd
hypot
ieee
inet
infnan
initgroups
insque
j0
lgamma
lib2648
malloc
math
mktemp
monitor
mount
mp
ndbm
nice
nlist
ns

directory operations
output conversion
last locations in program
error functions
monitor traffic on the Ethernet
execute a file
terminate a process after flushing any pending output
exponential, logarithm, power
close or flush a stream
stream status inquiries
absolute value, floor, ceiling, and round-to-nearest functions
open a stream
buffered binary input/output
split into mantissa and exponent
reposition a stream
get character or word from stream
disk description by its name
value for environment name
get file system descriptor file entry
get group file entry
get network host entry
get login name
get file system descriptor file entry
get network entry
get network group entry
get option letter from argv
read a password
get protocol entry
group file entry
get rpc entry
get RPC port number
get a string from a stream
get service entry
get ttys file entry
get legal user shells
get current working directory pathname
Euclidean distance, complex absolute value
copysign, remainder, exponent manipulations
Internet address manipulation routines
signals invalid floating-point operations on a VAX (temporary)
initialize group access list
insert/remove element from a queue
bessel functions
log gamma function
subroutines for the HP 2648 graphics terminal
memory allocator
introduction to mathematical library functions
make a unique file name
prepare execution profile
keep track of remotely mounted filesystems
multiple precision integer arithmetic
data base subroutines
set program priority
get entries from name list
Xerox NS(tm) address conversion routines
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pause</td>
<td>stop until signal</td>
</tr>
<tr>
<td>perror</td>
<td>system error messages</td>
</tr>
<tr>
<td>plot</td>
<td>graphics interface</td>
</tr>
<tr>
<td>popen</td>
<td>initiate I/O to/from a process</td>
</tr>
<tr>
<td>printf</td>
<td>formatted output conversion</td>
</tr>
<tr>
<td>perror</td>
<td>system signal messages</td>
</tr>
<tr>
<td>putc</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>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; routines for changing generators</td>
</tr>
<tr>
<td>rcmd</td>
<td>routines for returning a stream to a remote command</td>
</tr>
<tr>
<td>regex</td>
<td>regular expression handler</td>
</tr>
<tr>
<td>resolver</td>
<td>resolver routines</td>
</tr>
<tr>
<td>rusers</td>
<td>return information about users on remote machines</td>
</tr>
<tr>
<td>rquota</td>
<td>implement quotas on remote machines</td>
</tr>
<tr>
<td>rstat</td>
<td>get performance data from remote kernel</td>
</tr>
<tr>
<td>rwall</td>
<td>write to specified remote machines</td>
</tr>
<tr>
<td>scandir</td>
<td>scan a directory</td>
</tr>
<tr>
<td>scanf</td>
<td>formatted input conversion</td>
</tr>
<tr>
<td>setbuf</td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>setjmp</td>
<td>non-local goto</td>
</tr>
<tr>
<td>setuid</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>siginterrupt</td>
<td>allow signals to interrupt system calls</td>
</tr>
<tr>
<td>signal</td>
<td>simplified software signal facilities</td>
</tr>
<tr>
<td>sin</td>
<td>trigonometric functions and their inverses</td>
</tr>
<tr>
<td>sinh</td>
<td>hyperbolic functions</td>
</tr>
<tr>
<td>sleep</td>
<td>suspend execution for interval</td>
</tr>
<tr>
<td>spray</td>
<td>scatter data in order to check the network</td>
</tr>
<tr>
<td>sqrt</td>
<td>cube root, square root</td>
</tr>
<tr>
<td>stdio</td>
<td>standard buffered input/output package</td>
</tr>
<tr>
<td>string</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>syslog</td>
<td>control system log</td>
</tr>
<tr>
<td>system</td>
<td>issue a shell command</td>
</tr>
<tr>
<td>termcap</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>ttyname</td>
<td>find name of a terminal</td>
</tr>
<tr>
<td>ualarm</td>
<td>schedule signal after specified time</td>
</tr>
<tr>
<td>ungetc</td>
<td>push character back into input stream</td>
</tr>
<tr>
<td>usleep</td>
<td>suspend execution for interval</td>
</tr>
<tr>
<td>utime</td>
<td>set file times</td>
</tr>
<tr>
<td>valloc</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>ypclnt</td>
<td>yellow pages client interface</td>
</tr>
<tr>
<td>yppasswd</td>
<td>update user password in yellow pages</td>
</tr>
</tbody>
</table>

**3F. Fortran Library**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intro</td>
<td>introduction to FORTRAN library functions</td>
</tr>
<tr>
<td>abort</td>
<td>abnormal termination</td>
</tr>
<tr>
<td>access</td>
<td>determine accessibility of a file</td>
</tr>
</tbody>
</table>
Table of Contents

- alarm
  execute a subroutine after a specified time
- bessel
  of two kinds for integer orders
- bit
  and, or, xor, not, rshift, lshift bitwise functions
- chdir
  change default directory
- chmod
  change mode of a file
- etime
  return elapsed execution time
- exit
  terminate process with status
- fdate
  return date and time in an ASCII string
- fmin
  return extreme values
- flush
  flush output to a logical unit
- fork
  create a copy of this process
- fseek
  reposition a file on a logical unit
- getarg
  return command line arguments
- getc
  get a character from a logical unit
- getcwd
  get pathname of current working directory
- getenv
  get value of environment variables
- getlog
  get user's login name
- getpid
  get process id
- getuid
  get user or group ID of the caller
- hostnm
  get name of current host
- idate
  return date or time in numerical form
- index
  tell about character objects
- ioinit
  change f77 I/O initialization
- kill
  send a signal to a process
- link
  make a link to an existing file
- loc
  return the address of an object
- long
  integer object conversion
- malloc
  memory allocator
- perror
  get system error messages
- plot
  f77 library interface to plot (3X) libraries.
- putc
  write a character to a fortran logical unit
- qsort
  quick sort
- rand
  return random values
- random
  better random number generator
- rename
  rename a file
- signal
  change the action for a signal
- sleep
  suspend execution for an interval
- stat
  get file status
- system
  execute a UNIX command
- time
  return system time
- topen
  f77 tape I/O
- trap
  trap arithmetic errors
- trapov
  trap and repair floating point overflow
- trpfpe
  trap and repair floating point faults
- ttynam
  find name of a terminal port
- unlink
  remove a directory entry
- wait
  wait for a process to terminate

4. Special Files

- intro
  introduction to special files and hardware support
- acc
  ACC LH/DH IMP interface
- ad
  Data Translation A/D converter
- arp
  Address Resolution Protocol
- autoconf
  diagnostics from the autoconfiguration code
- bk
  line discipline for machine-machine communication (obsolete)
- cons
  VAX-11 console interface

4.3BSD

June 1986
Table of Contents

- VAX 8600 console RL02 interface
- DEC IMP-11A LH/DH IMP interface
- phototypesetter interface
- DDN Standard Mode X.25 IMP interface
- DEC DEUNA 10 Mb/s Ethernet interface
- DH-11/DM-11 communications multiplexer
- DLU-11 communications multiplexer
- DEC DMC-11/DMR-11 point-to-point communications device
- DMF-32, terminal multiplexor
- DMZ-32 terminal multiplexor
- DN-11 autocall unit interface
- paging device
- ACC IF-11/HDH IMP interface
- RK6-11/RK06 and RK07 moving head disk
- MASSBUS disk interface
- TM-03/TE-16, TU-45, TU-77 MASSBUS magtape interface
- Network Systems Hyperchannel interface
- Ikonas frame buffer, graphics device interface
- Interlan N1010 10 Mb/s Ethernet interface
- 1822 network interface
- IMP raw socket interface
- Internet protocol family
- Internet Protocol
- Interlan Np100 10 Mb/s Ethernet interface
- KL-11/DL-11W line clock
- software loopback network interface
- line printer
- main memory
- TM78/TU-78 MASSBUS magtape interface
- UNIX magtape interface
- Interlan Np100 10 Mb/s Ethernet interface
- Xerox Network Systems(tm) protocol family
- software network interface encapsulating ns packets in ip packets.
- data sink
- DEC CSS PCL-11 B Network Interface
- Evans and Sutherland Picture System 2 graphics device interface
- pseudo terminal driver
- DEC DEQNA Q-bus 10 Mb/s Ethernet interface
- DEC RX02 floppy disk interface
- Xerox Sequenced Packet Protocol
- line discipline for digitizing devices
- Internet Transmission Control Protocol
- TM-11/TE-10 magtape interface
- DEC TMSCP magtape interface
- TS-11 magtape interface
- general terminal interface
- VAX-11/730 and VAX-11/750 TU58 console cassette interface
- UDA-50 disk controller interface
- Internet User Datagram Protocol
### Table of Contents

- **up**
  - UNIBUS storage module controller/drives
- **ut**
  - UNIBUS TU45 tri-density tape drive interface
- **uu**
  - TU58/DECtape II UNIBUS cassette interface
- **va**
  - Benson-Varian interface
- **vp**
  - Versatec interface
- **vv**
  - Protean proNET 10 Megabit ring

#### 5. File Formats

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-devices</td>
<td>UUCP device description file</td>
</tr>
<tr>
<td>L-dialcodes</td>
<td>UUCP phone number index file</td>
</tr>
<tr>
<td>L.aliases</td>
<td>UUCP hostname alias file</td>
</tr>
<tr>
<td>L.cmds</td>
<td>UUCP remote command permissions file</td>
</tr>
<tr>
<td>L.sys</td>
<td>UUCP remote host description file</td>
</tr>
<tr>
<td>USERFILE</td>
<td>assembler and link editor output</td>
</tr>
<tr>
<td>a.out</td>
<td>execution accounting file</td>
</tr>
<tr>
<td>aliases</td>
<td>aliases file for sendmail</td>
</tr>
<tr>
<td>ar</td>
<td>archive (library) file format</td>
</tr>
<tr>
<td>core</td>
<td>format of memory image file</td>
</tr>
<tr>
<td>dbx</td>
<td>dbx symbol table information</td>
</tr>
<tr>
<td>dir</td>
<td>format of directories</td>
</tr>
<tr>
<td>disktab</td>
<td>disk description file</td>
</tr>
<tr>
<td>dump</td>
<td>incremental dump format</td>
</tr>
<tr>
<td>exports</td>
<td>NFS file systems being exported</td>
</tr>
<tr>
<td>fs</td>
<td>format of file system volume</td>
</tr>
<tr>
<td>fstab</td>
<td>static information about filesystems</td>
</tr>
<tr>
<td>gettytab</td>
<td>terminal configuration data base</td>
</tr>
<tr>
<td>group</td>
<td>group file</td>
</tr>
<tr>
<td>hosts</td>
<td>host name data base</td>
</tr>
<tr>
<td>hosts.equiv</td>
<td>list of trusted hosts</td>
</tr>
<tr>
<td>map3270</td>
<td>database for mapping ascii keystrokes into IBM 3270 keys</td>
</tr>
<tr>
<td>mtab</td>
<td>mounted file system table</td>
</tr>
<tr>
<td>netgroup</td>
<td>list of network groups</td>
</tr>
<tr>
<td>networks</td>
<td>network name data base</td>
</tr>
<tr>
<td>passwd</td>
<td>password file</td>
</tr>
<tr>
<td>phones</td>
<td>remote host phone number data base</td>
</tr>
<tr>
<td>plot</td>
<td>graphics interface</td>
</tr>
<tr>
<td>printcap</td>
<td>printer capability data base</td>
</tr>
<tr>
<td>protocols</td>
<td>protocol name data base</td>
</tr>
<tr>
<td>remote</td>
<td>remote host description file</td>
</tr>
<tr>
<td>resolver</td>
<td>resolver configuration file</td>
</tr>
<tr>
<td>rmtab</td>
<td>remotely mounted file system table</td>
</tr>
<tr>
<td>services</td>
<td>service name data base</td>
</tr>
<tr>
<td>stab</td>
<td>symbol table types</td>
</tr>
<tr>
<td>tar</td>
<td>tape archive file format</td>
</tr>
<tr>
<td>termcap</td>
<td>terminal capability data base</td>
</tr>
<tr>
<td>tp</td>
<td>DEC/mag tape formats</td>
</tr>
<tr>
<td>ttys</td>
<td>terminal initialization data</td>
</tr>
<tr>
<td>types</td>
<td>primitive system data types</td>
</tr>
<tr>
<td>utmp</td>
<td>login records</td>
</tr>
<tr>
<td>uuencode</td>
<td>format of an encoded uuencode file</td>
</tr>
<tr>
<td>vfont</td>
<td>font formats for the Benson-Varian or Versatec</td>
</tr>
<tr>
<td>vgrindefs</td>
<td>vgrind's language definition data base</td>
</tr>
<tr>
<td>ypfiles</td>
<td>the yellowpages database and directory structure</td>
</tr>
</tbody>
</table>
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 system calls overload the meanings of these error numbers, and that the meanings must be interpreted 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>.

0 Error 0
   Unused.

1 EPERM Not owner
   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.

2 ENOENT No such file or directory
   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.

3 ESRCH No such process
   The process or process group whose number was given does not exist, or any such process is already dead.

4 EINTR Interrupted system call
   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.

5 EIO I/O error
   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.

6 ENXIO No such device or address
   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.

7 E2BIG Arg list too long
   An argument list longer than 20480 bytes (or the current limit, NCARGS in <sys/param.h>) is presented to execve.

8 ENOEXEC Exec format error
   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)).

9 EBADF Bad file number
   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).
10 ECHILD No children
   "Wait" and the process has no living or unwaited-for children.

11 EAGAIN No more processes
   In a "fork," the system's process table is full or the user is not allowed to create any
   more processes.

12 ENOMEM Not enough memory
   During an "execve" or "break," a program asks for more core or swap space than the sys-
   tem 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
   condition; the maximum size of the text, data, and stack segments is a system
   parameter. Soft limits may be increased to their corresponding hard limits.

13 EACCES Permission denied
   An attempt was made to access a file in a way forbidden by the protection system.

14 EFAULT Bad address
   The system encountered a hardware fault in attempting to access the arguments of a
   system call.

15 ENOTBLK Block device required
   A plain file was mentioned where a block device was required, e.g., in "mount".

16 EBUSY Device busy
   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.

17 EEXIST File exists
   An existing file was mentioned in an inappropriate context, e.g., "link".

18 EXDEV Cross-device link
   A hard link to a file on another device was attempted.

19 ENODEV No such device
   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.

20 ENOTDIR Not a directory
   A non-directory was specified where a directory is required, for example, in a path
   name or as an argument to "chdir".

21 EISDIR Is a directory
   An attempt to write on a directory.

22 EINVAL Invalid argument
   Some invalid argument: dismounting a non-mounted device, mentioning an unkn-
   own signal in "signal," or some other argument inappropriate for the call. Also set
   by math functions, (see "math(3)").

23 ENFILE File table overflow
   The system's table of open files is full, and temporarily no more "opens" can be
   accepted.

24 EMFILE Too many open files
   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.
25 ENOTTY Inappropriate ioctl for device
   The file mentioned in an ioctl is not a terminal or one of the devices to which this
call applies.
26 ETXTBSY Text file busy
   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.
27 EFBIG File too large
   The size of a file exceeded the maximum (about $2^{31}$ bytes).
28 ENOSPC No space left on device
   A write to an ordinary file, the creation of a directory or symbolic link, or the crea-
tion 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.
29 ESPIPE Illegal seek
   An lseek was issued to a socket or pipe. This error may also be issued for other
non-seekable devices.
30 EROFS Read-only file system
   An attempt to modify a file or directory was made on a device mounted read-only.
31 EMLINK Too many links
   An attempt to make more than 32767 hard links to a file.
32 EPIPE Broken pipe
   A write on a pipe or socket for which there is no process to read the data. This con-
dition normally generates a signal; the error is returned if the signal is caught or
ignored.
33 EDOM Argument too large
   The argument of a function in the math package (3M) is out of the domain of the
function.
34 ERANGE Result too large
   The value of a function in the math package (3M) is unrepresentable within
machine precision.
35 EWOULDBLOCK Operation would block
   An operation that would cause a process to block was attempted on an object in
non-blocking mode (see fcntl(2)).
36 EINPROGRESS Operation now in progress
   An operation that takes a long time to complete (such as a connect(2)) was
attempted on a non-blocking object (see fcntl(2)).
37 EALREADY Operation already in progress
   An operation was attempted on a non-blocking object that already had an operation
in progress.
38 ENOTSOCK Socket operation on non-socket
   Self-explanatory.
39 EDESTADDRREQ Destination address required
   A required address was omitted from an operation on a socket.
40 EMSGSIZE Message too long
   A message sent on a socket was larger than the internal message buffer or some
other network limit.
**EPROTOTYPE**  Protocol wrong type for socket  
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.

**ENOPROTOOPT**  Option not supported by protocol  
A bad option or level was specified in a `getsockopt(2)` or `setsockopt(2)` call.

**EPROTONOSUPPORT**  Protocol not supported  
The protocol has not been configured into the system or no implementation for it exists.

**ESOCKTNOSUPPORT**  Socket type not supported  
The support for the socket type has not been configured into the system or no implementation for it exists.

**EOPNOTSUPP**  Operation not supported on socket  
For example, trying to `accept` a connection on a datagram socket.

**EPFNOSUPPORT**  Protocol family not supported  
The protocol family has not been configured into the system or no implementation for it exists.

**EAFNOSUPPORT**  Address family not supported by protocol family  
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.

**EADDRINUSE**  Address already in use  
Only one usage of each address is normally permitted.

**EADDRNOTAVAIL**  Can’t assign requested address  
Normally results from an attempt to create a socket with an address not on this machine.

**ENETDOWN**  Network is down  
A socket operation encountered a dead network.

**ENETUNREACH**  Network is unreachable  
A socket operation was attempted to an unreachable network.

**ENETRESET**  Network dropped connection on reset  
The host you were connected to crashed and rebooted.

**ECONNABORTED**  Software caused connection abort  
A connection abort was caused internal to your host machine.

**ECONNRESET**  Connection reset by peer  
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.

**ENOBUFS**  No buffer space available  
An operation on a socket or pipe was not performed because the system lacked sufficient buffer space or because a queue was full.

**EISCONN**  Socket is already connected  
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.

**ENOTCONN**  Socket is not 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.
58 ESHUTDOWN Can't send after socket shutdown
   A request to send data was disallowed because the socket had already been shut
down with a previous shutdown(2) call.

59 unused

60 ETIMEDOUT Connection timed out
   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 communica-
tion protocol.)

61 ECONNREFUSED Connection refused
   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.

62 ELOOP Too many levels of symbolic links
   A path name lookup involved more than 8 symbolic links.

63 ENAMETOOLONG File name too long
   A component of a path name exceeded 255 (MAXNAMELEN) characters, or an
entire path name exceeded 1023 (MAXPATHLEN-1) characters.

64 EHOSTDOWN Host is down
   A socket operation failed because the destination host was down.

65 EHOSTUNREACH Host is unreachable
   A socket operation was attempted to an unreachable host.

66 ENOTEMPTY Directory not empty
   A directory with entries other than "." and "...", was supplied to a remove directory
or rename call.

69 EDQUOT Disc quota exceeded
   A write to an ordinary file, the creation of a directory or symbolic link, or the crea-
tion 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.

70 ESTALE Stale NFS file handle
   A client referenced a an open file, when the file has been deleted.

71 EREMOTE Too many levels of remote in path
   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 pro-
cess 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 ID
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, and Access Groups
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 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
accept - accept a connection on a socket

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

int accept(int s, struct sockaddr *addr, socklen_t *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 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:

- [EBADF] 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)
This page intentionally left almost blank.
NAME
access – determine accessibility of file

SYNOPSIS
#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 directories leading to the file can be searched and the file exists.

The real user ID and the group access list (including the real group ID) 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.
[EACCESS] Search permission is denied for a component of the path prefix.
[ELOOP] Too many symbolic links were encountered in translating the pathname.
[EROFS] Write access is requested for a file on a read-only file system.
[ETXTBSY] Write access is requested for a pure procedure (shared text) file that is being executed.
[EACCESS] 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] An I/O error occurred while reading from or writing to the file system.

SEE ALSO
chmod(2), stat(2)

4th Berkeley Distribution May 22, 1986
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 exercised 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)

BUGS
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 4.3BSD, 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 inadequate 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.
[ENAMETOOLONG] 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 pathname.
[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)

BUGS
Setting the break may fail due to a temporary lack of swap space. It is not possible to distinguish this from a failure caused by exceeding the maximum size of the data segment without consulting getrlimit.
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 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 directory does not exist.
[ELOOP] Too many symbolic links were encountered in translating the pathname.
[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 system.

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, defined 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
- 00007 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.
- [ENAME_TOO_LOLONG] 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.
The named file resides on a read-only file system.

Path points outside the process’s allocated address space.

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

Fchmod will fail if:

The descriptor is not valid.

Fd refers to a socket, not to a file.

The file resides on a read-only file system.

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 path name.
[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
   chroot – change root directory

SYNOPSIS
   chroot(dirname)
   char *dirname;

DESCRIPTION
   Dirname is the address of the pathname of a directory, terminated by a null byte. Chroot
   causes this directory to become the root directory, the starting point for path names beginning
   with “/”.

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

   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 an error.

ERRORS
   Chroot will fail and the root directory will be unchanged if one or more of the following are
   true:
   [ENOTDIR] A component of the path name 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.
   [EACCES] Search permission is denied for any component of the path name.
   [ELOOP] Too many symbolic links were encountered in translating the pathname.
   [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.

SEE ALSO
   chdir(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 completed
                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).
Creal 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
Creal 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 issu-
ing 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 charac-
  ter 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:

    process ID          see getpid(2)
    parent process ID   see getppid(2)
    process group ID    see getgrgid(2)
    access groups       see getgroups(2)
    working directory   see chdir(2)
    root directory      see chroot(2)
    control terminal    see tty(4)
    resource usages     see getrusage(2)
    interval timers     see getitimer(2)
    resource limits     see getrlimit(2)
    file mode mask      see umask(2)
    signal mask         see sigvec(2), sigmask(2)

  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 standard 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 provides for control over descriptors. The argument fd is a 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.
Same object references as the original descriptor.
New descriptor shares the same file pointer if the object was a file.
Same access mode (read, write or read/write).
Same file status flags (i.e., both file descriptors share the same file status flags).
The close-on-exec flag associated with the new file descriptor is set to remain open across execv(2) system calls.

F_GETFD Get the close-on-exec flag associated with the file 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, as described below.

F_SETFL Set descriptor status flags.

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 flags for the F_GETFL and F_SETFL flags are as follows:

FNDELAY Non-blocking I/O; if no data is available to a read call, or if a write operation would block, the call returns -1 with the error EWOULDBLOCK.

FAPPEND Force each write to append at the end of file; corresponds to the O_APPEND flag of open(2).

FASYNC Enable the SIGIO signal to be sent to the process group when I/O is possible, e.g., upon availability of data to be read.

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

F_DUPFD A new file descriptor.
F_GETFD Value of flag (only the low-order bit is defined).
F_GETFL Value of flags.
F_GETOWN Value of file descriptor owner.
other Value other than -1.
Otherwise, a value of -1 is returned and \texttt{errno} is set to indicate the error.

**ERRORS**

\texttt{fcntl} will fail if one or more of the following are true:

- \textbf{[EBADF]} \textit{Fildes} is not a valid open file descriptor.
- \textbf{[EMFILE]} \textit{Cmd} is \texttt{F\_DUPFD} and the maximum allowed number of file descriptors are currently open.
- \textbf{[EINVAL]} \textit{Cmd} is \texttt{F\_DUPFD} and \textit{arg} is negative or greater than the maximum allowable number (see \texttt{getdtablesize(2)}).
- \textbf{[ESRCH]} \textit{Cmd} is \texttt{F\_SETOWN} and the process ID given as argument is not in use.

**SEE ALSO**

close(2), execve(2), getdtablesize(2), open(2), sigve(2)

**BUGS**

The asynchronous I/O facilities of \texttt{FNDELAY} and \texttt{FASYNC} are currently available only for \texttt{tty} and socket operations.
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 happen; 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:
[EWOULDBLOCK] The file is locked and the LOCK_NB option was specified.
[EBADF] The argument fd is an invalid descriptor.
(EIFVAL] 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 refer­
ence the same underlying objects, so that, for instance, file pointers in file objects are
shared between the child and the parent, so that an isseek(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).

RETURN VALUE
Upon successful completion, fork returns a value of 0 to the child process and returns the pro­
cess 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
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_namlen;
  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_namlen 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.

SEE ALSO
open(2), lseek(2)

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

[EBADF]     fd is not a valid file descriptor open for reading.
[EFAULT]    Either buf or basep point outside the allocated address space.
[EINTR]     A read from a slow device was interrupted before any data arrived by the delivery of a signal.
[EIO] An I/O error occurred while reading from or writing to the file system.
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
common 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 setdomainname.

BUGS
Domain names are limited to 255 characters.
This page intentionally left almost blank.
NAME
getdtablesize – get descriptor table size

SYNOPSIS
nfds = getdtablesize()
int nfds;

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 getdtablesize 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)

BUGS
The gidset array should be of type gid_t, but remains integer for compatibility with earlier systems.
GETHOSTID(2) UNIX Programmer's Manual GETHOSTID(2)

NAME
gethostid, sethostid — get/set unique identifier of current host

SYNOPSIS
hostid = gethostid()
long hostid;

sethostid(hostid)
long hostid;

DESCRIPTION
Sethostid 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)

BUGS
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
sethostname. 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.

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 process 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 running on behalf of the process. It is designed to be used by interpreters in statistically profiling the execution of interpreted programs. Each time the ITIMER_PROF timer expires, the SIGPROF signal is delivered. Because this signal may interrupt in-progress system calls, programs using this timer must be prepared to restart interrupted system calls.

NOTES
Three macros for manipulating time values are defined in <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
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 TIOCGPGRP 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.

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

SEE ALSO
    gethostid(2)
NAME
gotprioritiy, 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
process 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 favorable 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)
GETRLIMIT(2) UNIX Programmer's Manual GETRLIMIT(2)

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:
RLIMIT_CPU the maximum amount of cpu time (in seconds) to be used by each process.
RLIMITFSIZE the largest size, in bytes, of any single file that may be created.
RLIMIT_DATA 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.
RLIMIT_STACK 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.
RLIMIT_CORE the largest size, in bytes, of a core file that may be created.
RLIMIT_RSS 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.

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,

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 (0x7fffffff).

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)

BUGS
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;

DESCRIPTION
Getrusage returns information describing the resources utilized by the current process, or all
its terminated child processes. 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; /* involuntary context switches */
    int ru_nivcsw; /* voluntary 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 kilobytes).
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 kilobytes • 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 kilobytes • 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 kilobytes • seconds-of-execution).
GETRUSAGE(2) UNIX Programmer's Manual GETRUSAGE(2)

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
gettimeofday(2), wait(2)

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

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

DESCRIPTION
Getsockname 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:
[EBADFD] The argument s is not a valid descriptor.
[ENOTSOCK] The argument s is a file, not a socket.
[ENOBUS] 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)

BUGS
Names bound to sockets in the UNIX domain are inaccessible; getsockname 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_DEBUG</td>
<td>toggle recording of debugging information</td>
</tr>
<tr>
<td>SO_REUSEADDR</td>
<td>toggle local address reuse</td>
</tr>
<tr>
<td>SO_KEEPALIVE</td>
<td>toggle keep connections alive</td>
</tr>
<tr>
<td>SO_DONTRoute</td>
<td>toggle routing bypass for outgoing messages</td>
</tr>
<tr>
<td>SO_LINGER</td>
<td>linger on close if data present</td>
</tr>
<tr>
<td>SO_BROADCAST</td>
<td>toggle permission to transmit broadcast messages</td>
</tr>
<tr>
<td>SO_OOBINLINE</td>
<td>toggle reception of out-of-band data in band</td>
</tr>
<tr>
<td>SO_SNDBUF</td>
<td>set buffer size for output</td>
</tr>
<tr>
<td>SO_RCVBUF</td>
<td>set buffer size for input</td>
</tr>
<tr>
<td>SO_TYPE</td>
<td>get the type of the socket (get only)</td>
</tr>
<tr>
<td>SO_ERROR</td>
<td>get and clear error on the socket (get only)</td>
</tr>
</tbody>
</table>
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.
[ENOTSOCK] The argument s is a file, not a socket.
[ENOPROTOOPT] 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)

BUGS
Several of the socket options should be handled at lower levels of the system.
NAME
gettimeofday, settimeofday — get/set date and time

SYNOPSIS
#include <sys/time.h>
gettimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;
settimeofday(tp, tzp)
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 gettimeofday 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
ioctl – 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
c 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:

[EBADF]  D is not a valid descriptor.
[ENOTTY]  D is not associated with a character special device.
[ENOTTY]  The specified request does not apply to the kind of object that the descriptor
          d references.
[EINVAL]  Request or argp is not valid.

SEE ALSO
execve(2), fcntl(2), mt(4), tty(4), intro(4N)
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 sig-
naled.

SEE ALSO

getpid(2), getpgrp(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 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 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
  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 path­
                 names.
  [ENOENT]      The file named by name1 does not exist.
  [EEXIST]      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)

BUGS
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 */

int 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.
[ESPIPE]  Fildes is associated with a pipe or a socket.
[EINVAL]  Whence is not a proper value.

SEE ALSO
dup(2), open(2)

BUGS
This document’s use of whence is incorrect English, but maintained for historical reasons.
MKDIR(2)

NAME
mkdir – make a directory file

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

DESCRIPTION
mkdir 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
mkdir 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 containing 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 con­
taining 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.

4.2 Berkeley Distribution August 26, 1985
[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)
MKNOD(2)  UNIX Programmer’s Manual  MKNOD(2)

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.
  [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 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 containing 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 system 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.
  [EEXIST] The named file exists.
[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.

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 */
    fhandle_t *fh; /* File handle to be mounted */
    int flags; /* flags */
    int wsize; /* write size in bytes */
    int rsize; /* read size in bytes */
    int timeo; /* initial timeout in .1 secs */
    int retrans; /* times to retry send */
};

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

ERRORS
Mount will fail when one of the following occurs:
[EPERM] The caller is not the super-user.
[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).
[EBUSY] Dir is not a directory, or another process currently holds a reference to it.

Sun Microsystems Rel 3.0 19 August 1985
[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.
[ENOTDIR] A component of the path prefix in \textit{special or name} is not a directory.
[EPERM] The pathname of \textit{special or name} contains a character with the high-order bit set.
[ENAMETOOLONG] The pathname of \textit{special or name} was too long.
[ENOENT] \textit{Special or name} does not exist.
[EACCES] Search permission is denied for a component of the path prefix of \textit{special or name}.
[EFAULT] \textit{Special or name} points outside the process's allocated address space.
[ELOOP] Too many symbolic links were encountered in translating the pathname of \textit{special or name}.
[EIO] An I/O error occurred while reading from or writing to the file system.

\textbf{SEE ALSO}
unmount(2), mount(8)

\textbf{BUGS}
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.

BUGS
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_NDELAY  do not block on open
- O_APPEND  append on each write
- O_CREAT   create file if it does not exist
- 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 component of the path name is a symbolic link, the open will fail even if the symbolic link points to a non-existent name. If the O_NDELAY 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 process 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 process. Getdtablesize(2) returns the current system limit.

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

- [EINVAL]  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 writing.

[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), getdtablesize(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)

BUGS
Should more than 4096 bytes be necessary in any pipe among a loop of processes, deadlock will occur.
This page intentionally left almost blank.
NAME
prof - 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 <sys/signal.h>
#include <sys/ptrace.h>

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

DESCRIPTION
Ptrace provides a means by which a parent process may control the execution of a child 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, which must be a child (no more distant descendant) of the tracing 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. Then the traced process enters a stopped state and its parent is notified via wait(2). When the child is in the stopped state, its core image can be examined and modified using ptrace. If desired, another ptrace request can then cause the child either to terminate or to continue, possibly ignoring the signal.

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

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

PT_READ_I, PT_READ_D
The word in the child process’s address space at addr is returned. If I and D space are separated (e.g. historically on a pdp-11), request PT_READ_I indicates I space, PT_READ_D D space. Addr must be even on some machines. The child must be stopped. The input data is ignored.

PT_READ_U
The word of the system’s per-process data area corresponding to addr is returned. Addr must be even on some machines and less than 512. This space contains the registers and other information about the process; its layout corresponds to the user structure in the system.

PT_WRITE_I, PT_WRITE_D
The given data is written at the word in the process’s address space corresponding to addr, which must be even on some machines. No useful value is returned. If I and D space are separated, request PT_WRITE_I indicates I space, PT_WRITE_D D space. Attempts to write in pure procedure fail if another process is executing the same file.

PT_WRITE_U
The process’s system data is written, as it is read with request PT_READ_U. 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.

PT_CONTINUE
The data argument is taken as a signal number and the child’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.
PT_KILL
The traced process terminates.

PT_STEP
Execution continues as in request PT_CONTINUE; however, as soon as possible after execution of at least one instruction, execution stops again. The signal number from the stop is SIGTRAP. (On the VAX-11 the T-bit is used and just one instruction is executed.) This is part of the mechanism for implementing breakpoints.

As indicated, these calls (except for request PT_TRACE_ME) 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.

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.

On a VAX-11, “word” also means a 32-bit integer, but the “even” restriction does not apply.

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
[EIO] The request code is invalid.
[ESRCH] The specified process does not exist.
[EIO] The given signal number is invalid.
[EIO] 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 that 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 PT_TRACE_ME call should be able to specify signals that 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;
char *addr;

DESCRIPTION
N.B.: This call is not implemented in the NFS version of the system. The quota call manipulates disk quotas for file systems that 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 that 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 dqusage structure (defined in <sys/quota.h>). This call is restricted to the super-user.

Q_SYNC
Update the on-disc copy of quota usages. Arg is a major-minor device indicating the file system to be sync'ed. If the arg parameter is specified as NODEV, all file systems that have disc quotas will be sync'ed. The uid 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 user is under quota, his warning count is reset to MAX_*_WARN (defined in <sys/quota.h>). If the arg parameter is specified as NODEV, all file systems that 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, otherwise the value —1 is returned and the global variable errno indicates the reason for the failure.
ERRORS

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

* [EINV AL] The kernel has not been compiled with the QUOTA option.
* [EINV AL] *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.
* [ENODEV] 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 some way 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 quotacheck(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 restricted 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.

[EUSERS] 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
  iovcnt 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.
  [EINTR]  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] `iowcnt` was less than or equal to 0, or greater than 16.

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

[EINVAL] The sum of the `iow_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 interface permits only RB_HALT or RB_AUTOBOOT 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 procedures. When none of these options (e.g. RB_AUTOBOOT) 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_HALT
    the processor is simply halted; no reboot takes place. RB_HALT 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 consistency 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>

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

ce = recvfrom(s, buf, len, flags, from, fromlen)
int cc, s;
char *buf;
int len, flags;
struct sockaddr *from;
int *fromlen;

cc = recvmsg(s, msg, flags)
int cc, s;
struct msghdr msg[];
int 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 recvmsg 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 EWOULDBLOCK.

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_iov_len;  /* # elements in msg_iov */
    caddr_t msg_control;  /* access rights sent/received */
    int msg_control_len;
};

Here msg_name and msg_name_len 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_iov_len 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 \textit{msg\_accrights}, which has length \textit{msg\_accrightslen}. Access rights are currently limited to file descriptors, which each occupy the size of an \textit{int}.

**RETURN VALUE**

These calls return the number of bytes received, or \textbf{-1} if an error occurred.

**ERRORS**

The calls fail if:

- [\textbf{EBADF}] The argument \textit{s} is an invalid descriptor.
- [\textbf{ENOTSOCK}] The argument \textit{s} is not a socket.
- [\textbf{EWOULDBLOCK}] The socket is marked non-blocking and the receive operation would block.
- [\textbf{EINTR}] The receive was interrupted by delivery of a signal before any data was available for the receive.
- [\textbf{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 Flto 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.

[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 dev-
ices (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
  rmdir – remove a directory file

SYNOPSIS
  rmdir(path)
  char *path;

DESCRIPTION
  Rmdir 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.
  [ELOOP] 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 link to be
  removed.
  [EPERM] 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>
nfound = select(nfds, readfds, writefds, exceptfds, timeout)
int nfound, 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 nfound.

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) initializes 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.

[EINTR] 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 modi­
fying 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 message is given by len. If the message is too long to pass atomically through the underlying protocol, 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
getgroups(2), initgroups(3X)

BUGS
The gidset array should be of type gid_t, but remains integer for compatibility with earlier systems.
NAME
setpgroup - set process group

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

DESCRIPTION
Setpgroup 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
Setpgroup returns when the operation was successful. If the request failed, -1 is returned and
the global variable errno indicates the reason.

ERRORS
Setpgroup 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 descendent of the calling process.

SEE ALSO
getpgroup(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:
[ENOTDIR] A component of either path prefix is not a directory.
[EINVAL] Either path name contains a character with the high-order bit set.
[EINVAL] The kernel has not been compiled with the QUOTA option.
[ENAMETOOLONG] A component of either path name exceeded 255 characters, or the entire length of either path name exceeded 1023 characters.
[ENODEV] Special does not exist.
[ENOENT] File does not exist.
[ELOOP] Too many symbolic links were encountered in translating either path name.
[EPERM] The caller is not the super-user.
[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).
[EROFS] File resides on a read-only file system.
[EACCES] Search permission is denied for a component of either path prefix.
[EACCES] File resides on a file system different from special.
[EACCES] File is not a plain file.
[EIO] An I/O error occurred while reading from or writing to the file containing the quotas.
[EFAULT] Special or path points outside the process's allocated address space.

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>

struct sigcootext {
    int sc_onstack;
    int sc_mask;
    int sc_sp;
    int sc_fp;
    int sc_ap;
    int sc_pc;
    int sc_ps;
};
sigreturn(scp);
struct sigcootext *scp;

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 stack pointer, frame pointer, argument pointer, and processor status longword are restored from the context. Execution resumes at the specified pc. 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 one of the following occurs.

[EFAULT] Scp points to memory that is not a valid part of the process address space.
[EINVAL] The process status longword is invalid or would improperly raise the privilege level of the process.

SEE ALSO

sigvec(2), setjmp(3)
NAME
    sigsetmask - 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 process 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 process 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
    SIGIOF   6*  IOT instruction
    SIGEMT   7*  EMT instruction
SIGVEC(2) UNIX Programmer's Manual SIGVEC(2)

SIGFPE  8  floating point exception
SIGKILL  9  kill (cannot be caught, blocked, 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
SIGURG  16  urgent condition present on socket
SIGSTOP  17†  stop (cannot be caught, blocked, or ignored)
SIGTSTP  18†  stop signal generated from keyboard
SIGCONT  19  continue after stop (cannot be blocked)
SIGCHLD  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 (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.

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 signals marked with • or †. Signals marked with • are discarded if the action is SIG_DFL; signals marked with † cause the process to stop. If sv_handler is SIG_IGN the signal is subsequently 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 device (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.

Execve(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 SIGKILL, 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 is set to indicated the reason.

ERRORS

Sigvec will fail and no new signal handler will be installed if one of the following occurs:
Either `vee` or `ovec` points to memory that is not a valid part of the process address space.

`SIGILL` is not a valid signal number.

An attempt is made to ignore or supply a handler for `SIGKILL` or `SIGSTOP`.

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)`

NOTES (VAX-11)

The handler routine can be declared:

```c
handler(sig, code, scp)
int sig, code;
struct sigcontext *scp;
```

Here `sig` is the signal number, into which the hardware faults and traps are mapped as defined below. `Code` is a parameter that is either a constant as given below or, for compatibility mode faults, the code provided by the hardware (Compatibility mode faults are distinguished from the other `SIGILL` traps by having PSL_CM set in the psl). `Scp` is a pointer to the `sigcontext` structure (defined in `<signal.h>`), used to restore the context from before the signal.

The following defines the mapping of hardware traps to signals and codes. All of these symbols are defined in `<signal.h>`:

<table>
<thead>
<tr>
<th>Hardware condition</th>
<th>Signal</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic traps:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer overflow</td>
<td>SIGFPE</td>
<td>FPE_INTOVF_TRAP</td>
</tr>
<tr>
<td>Integer division by zero</td>
<td>SIGFPE</td>
<td>FPE_INTDIV_TRAP</td>
</tr>
<tr>
<td>Floating overflow trap</td>
<td>SIGFPE</td>
<td>FPE_FTOVF_TRAP</td>
</tr>
<tr>
<td>Floating/decimal division by zero</td>
<td>SIGFPE</td>
<td>FPE_FLTDIV_TRAP</td>
</tr>
<tr>
<td>Floating underflow trap</td>
<td>SIGFPE</td>
<td>FPE_FLTUND_TRAP</td>
</tr>
<tr>
<td>Decimal overflow trap</td>
<td>SIGFPE</td>
<td>FPE_DECOFV_TRAP</td>
</tr>
<tr>
<td>Subscript-range</td>
<td>SIGFPE</td>
<td>FPE_SUBRNG_TRAP</td>
</tr>
<tr>
<td>Floating overflow fault</td>
<td>SIGFPE</td>
<td>FPE_FTOVF_FAULT</td>
</tr>
<tr>
<td>Floating divide by zero fault</td>
<td>SIGFPE</td>
<td>FPE_FLTDIV_FAULT</td>
</tr>
<tr>
<td>Floating underflow fault</td>
<td>SIGFPE</td>
<td>FPE_FLTDUND_FAULT</td>
</tr>
<tr>
<td>Length access control</td>
<td>SIGSEGV</td>
<td></td>
</tr>
<tr>
<td>Protection violation</td>
<td>SIGBUS</td>
<td></td>
</tr>
<tr>
<td>Reserved instruction</td>
<td>SIGILL</td>
<td>ILL_RESAD_FAULT</td>
</tr>
<tr>
<td>Customer-reserved instr.</td>
<td>SIGEMT</td>
<td></td>
</tr>
<tr>
<td>Reserved operand</td>
<td>SIGILL</td>
<td>ILL_PRIVIN_FAULT</td>
</tr>
<tr>
<td>Reserved addressing</td>
<td>SIGILL</td>
<td>ILL_RESOP_FAULT</td>
</tr>
<tr>
<td>Trace pending</td>
<td>SIGTRAP</td>
<td></td>
</tr>
<tr>
<td>Bpt instruction</td>
<td>SIGTRAP</td>
<td></td>
</tr>
<tr>
<td>Compatibility-mode</td>
<td>SIGILL</td>
<td>hardware supplied code</td>
</tr>
<tr>
<td>Chme</td>
<td>SIGSEGV</td>
<td></td>
</tr>
<tr>
<td>Chms</td>
<td>SIGSEGV</td>
<td></td>
</tr>
<tr>
<td>Chmu</td>
<td>SIGSEGV</td>
<td></td>
</tr>
</tbody>
</table>

BUGS

This manual page is still confusing.
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 process sends 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.

[EACCESS]
   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.
[EOPNOSUPPORT] 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 refer­
ences.

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_mtime; /* file last modify time */
    int    st_ctime; /* file last status change time */
    int    st_blocks; /* actual number of blocks allocated */
    long   st_blksize; /* optimal blocksize for file system i/o ops */
    long   st_blocks; /* actual number of blocks allocated */
};

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.
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 0120000 /* 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 *Istat* 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_sparel`, `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 pathname 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 {
    struct {
        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 */
        long     f_fileid; /* file system id */
        long     f_spare[7]; /* spare for later */
    } f_statfs;

};

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 is not a directory.
[EPERM] The pathname contains a character with the high-order bit set.
[ENAMEETOOLONG] The pathname was too long.
[ENOENT] The named file does not exist.
[EACCES] Search permission is denied for a component of the path prefix.
[EFAULT] Buf or name points to an invalid address.
[ELOOP] Too many symbolic links were encountered in translating the pathname.
[EIO] An I/O error occurred while reading from or writing to the file system.
Fstatfs fails 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.
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(namel, name2)
    char *namel, *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.
[ENAMETOOLOONG] 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.
[EXEXIST] 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 system containing the directory.
[ENOSPC] The new symbolic link cannot be created because there 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>
syscall(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 – 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:

[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] 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
  corresponding bits in the file mode (see *chmod(2)*). This clearing allows each user to restrict
  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
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 containing
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 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.
[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.
[ENOTDIR] A component of the path prefix is not a directory.
[EPERM] The pathname contains a character with the high-order bit set.
[ENAMETOOLONG] The pathname was too long.
[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 pathname.
[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

Ut ime 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)
This page intentionally left almost blank.
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 child's 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 ioctl's 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 con­
trol 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;

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. 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

4th Berkeley Distribution June 30, 1985
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

```c
struct iovec {
    char *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 following 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.

4th Berkeley Distribution May 14, 1986
[EFBIG] 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] iovec_t 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 fcntl(2), lseek(2), open(2), pipe(2), select(2)
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 call the infnan routine. By default this routine returns the VAX reserved floating point value which causes the process to get a floating point exception (see sigvec(2)). Programs that wish to take other action should define their own version of infnan (see infnan(3M) for details). 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 "—Im" option.
(3R) These functions constitute the RPC service library, librpcsvc. In order to get the link editor to load this library, use the —lrpcsvc option of cc. Declarations for these functions may be obtained from various include files <rpcsvc/*h>.
(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
/lib/libc.a the C library
/usr/lib/libm.a the math library
/usr/lib/libc_p.a the C library compiled for profiling
/usr/lib/libm_p.a the math library compiled for profiling

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>
</tbody>
</table>

4th Berkeley Distribution       June 30, 1986
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Manual Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>integer absolute value</td>
<td>abs.3</td>
</tr>
<tr>
<td>acos</td>
<td>inverse trigonometric function</td>
<td>sin.3m</td>
</tr>
<tr>
<td>acosh</td>
<td>inverse hyperbolic function</td>
<td>sinh.3m</td>
</tr>
<tr>
<td>alarm</td>
<td>schedule signal after specified time</td>
<td>alarm.3c</td>
</tr>
<tr>
<td>alloca</td>
<td>memory allocator</td>
<td>malloc.3</td>
</tr>
<tr>
<td>arc</td>
<td>graphics interface</td>
<td>plot.3x</td>
</tr>
<tr>
<td>asctime</td>
<td>convert date and time to ASCII</td>
<td>ctime.3</td>
</tr>
<tr>
<td>asin</td>
<td>inverse trigonometric function</td>
<td>sin.3m</td>
</tr>
<tr>
<td>asinh</td>
<td>inverse hyperbolic function</td>
<td>sinh.3m</td>
</tr>
<tr>
<td>assert</td>
<td>program verification</td>
<td>assert.3x</td>
</tr>
<tr>
<td>atan</td>
<td>inverse trigonometric function</td>
<td>sin.3m</td>
</tr>
<tr>
<td>atanh</td>
<td>inverse hyperbolic function</td>
<td>sinh.3m</td>
</tr>
<tr>
<td>atof</td>
<td>convert ASCII to numbers</td>
<td>atof.3</td>
</tr>
<tr>
<td>atoi</td>
<td>convert ASCII to numbers</td>
<td>atof.3</td>
</tr>
<tr>
<td>atol</td>
<td>convert ASCII to numbers</td>
<td>atof.3</td>
</tr>
<tr>
<td>bcmp</td>
<td>bit and byte string operations</td>
<td>bstring.3</td>
</tr>
<tr>
<td>bcopy</td>
<td>bit and byte string operations</td>
<td>bstring.3</td>
</tr>
<tr>
<td>bzero</td>
<td>bit and byte string operations</td>
<td>bstring.3</td>
</tr>
<tr>
<td>cabs</td>
<td>complex absolute value</td>
<td>hypot.3m</td>
</tr>
<tr>
<td>calloc</td>
<td>memory allocator</td>
<td>malloc.3</td>
</tr>
<tr>
<td>cbart</td>
<td>cube root</td>
<td>sqrt.3m</td>
</tr>
<tr>
<td>ceil</td>
<td>integer no less than</td>
<td>floor.3m</td>
</tr>
<tr>
<td>circle</td>
<td>graphics interface</td>
<td>plot.3x</td>
</tr>
<tr>
<td>clearerr</td>
<td>stream status inquiries</td>
<td>ferror.3s</td>
</tr>
<tr>
<td>closedir</td>
<td>directory operations</td>
<td>directory.3</td>
</tr>
<tr>
<td>closelog</td>
<td>control system log</td>
<td>syslog.3</td>
</tr>
<tr>
<td>closepl</td>
<td>graphics interface</td>
<td>plot.3x</td>
</tr>
<tr>
<td>cont</td>
<td>graphics interface</td>
<td>plot.3x</td>
</tr>
<tr>
<td>copiesign</td>
<td>copy sign bit</td>
<td>ieee.3m</td>
</tr>
<tr>
<td>cos</td>
<td>trigonometric function</td>
<td>sin.3m</td>
</tr>
<tr>
<td>coss</td>
<td>hyperbolic function</td>
<td>sinh.3m</td>
</tr>
<tr>
<td>crypt</td>
<td>DES encryption</td>
<td>crypt.3</td>
</tr>
<tr>
<td>ctime</td>
<td>convert date and time to ASCII</td>
<td>ctime.3</td>
</tr>
<tr>
<td>curses</td>
<td>screen functions with &quot;optimal&quot; cursor motion</td>
<td>curses.3x</td>
</tr>
<tr>
<td>dbminit</td>
<td>data base subroutines</td>
<td>dbm.3x</td>
</tr>
<tr>
<td>delete</td>
<td>data base subroutines</td>
<td>dbm.3x</td>
</tr>
<tr>
<td>drem</td>
<td>remainder</td>
<td>ieee.3m</td>
</tr>
<tr>
<td>ecvt</td>
<td>output conversion</td>
<td>ecvt.3</td>
</tr>
<tr>
<td>edata</td>
<td>last locations in program</td>
<td>end.3</td>
</tr>
<tr>
<td>encrypt</td>
<td>DES encryption</td>
<td>crypt.3</td>
</tr>
<tr>
<td>end</td>
<td>last locations in program</td>
<td>end.3</td>
</tr>
<tr>
<td>endfsent</td>
<td>get file system descriptor file entry</td>
<td>getfsent.3x</td>
</tr>
<tr>
<td>endgrent</td>
<td>get group file entry</td>
<td>getgrent.3</td>
</tr>
<tr>
<td>endhostent</td>
<td>get network host entry</td>
<td>gethostbyname.3n</td>
</tr>
<tr>
<td>endnetent</td>
<td>get network entry</td>
<td>getnetent.3n</td>
</tr>
<tr>
<td>endprotoent</td>
<td>get protocol entry</td>
<td>getprotoent.3n</td>
</tr>
<tr>
<td>endpwent</td>
<td>get password file entry</td>
<td>getpwent.3</td>
</tr>
<tr>
<td>endservent</td>
<td>get service entry</td>
<td>getservent.3n</td>
</tr>
<tr>
<td>environ</td>
<td>execute a file</td>
<td>exec1.3</td>
</tr>
<tr>
<td>erase</td>
<td>graphics interface</td>
<td>plot.3x</td>
</tr>
<tr>
<td>erf</td>
<td>error function</td>
<td>erf.3m</td>
</tr>
<tr>
<td>erfcc</td>
<td>complementary error function</td>
<td>erf.3m</td>
</tr>
</tbody>
</table>
etext  end.3  last locations in program
ether  ether.3r  monitor traffic on the Ethernet
exec  exec.3  execute a file
exece  exec.3  execute a file
execl  exec.3  execute a file
execlp  exec.3  execute a file
exec  exec.3  execute a file
execv  exec.3  execute a file
execvp  exec.3  execute a file
exit  exit.3  terminate a process after flushing any pending output
exp  exp.3m  exponential
expm1  exp.3m  \( \exp(x) - 1 \)
fabs  floor.3m  absolute value
fclose  fclose.3s  close or flush a stream
fcvt  fcvt.3  output conversion
feof  ferror.3s  stream status inquiries
ferror  ferror.3s  stream status inquiries
fetch  dbm.3x  data base subroutines
flush  fclose.3s  close or flush a stream
fs  bstring.3  bit and byte string operations
fgetc  getc.3s  get character or word from stream
fgets  gets.3s  get a string from a stream
fileno  ferror.3s  stream status inquiries
firstkey  dbm.3x  data base subroutines
floor  floor.3m  integer no greater than
fopen  fopen.3s  open a stream
fprintf  printf.3s  formatted output conversion
fputc  putc.3s  put character or word on a stream
fgets  puts.3s  put a string on a stream
fread  fread.3s  buffered binary input/output
free  malloc.3  memory allocator
frexp  frexp.3  split into mantissa and exponent
fscanf  scanf.3s  formatted input conversion
fseek  fseek.3s  reposition a stream
ftell  fseek.3s  reposition a stream
ftime  time.3c  get date and time
fwrite  fread.3s  buffered binary input/output
gcvt  fcvt.3  output conversion
getc  getc.3s  get character or word from stream
getchar  getc.3s  get character or word from stream
getdiskbyname  getdisk.3x  get disk description by its name
getenv  getenv.3  value for environment name
getfsent  getfsent.3x  get file system descriptor file entry
getfsfile  getfsent.3x  get file system descriptor file entry
getfsspec  getfsent.3x  get file system descriptor file entry
getfstype  getfsent.3x  get file system descriptor file entry
getgrent  getgrent.3  get group file entry
getgrgid  getgrent.3  get group file entry
getgrnam  getgrent.3  get group file entry
gethostbyaddr  gethostbyname.3n  get network host entry
gethostbyname  gethostbyname.3n  get network host entry
gethostent  gethostbyname.3n  get network host entry

4th Berkeley Distribution  June 30, 1986
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getlogin</td>
<td>get login name</td>
</tr>
<tr>
<td>getnetbyaddr</td>
<td>get network entry</td>
</tr>
<tr>
<td>getnetbyname</td>
<td>get network entry</td>
</tr>
<tr>
<td>getnetent</td>
<td>get network entry</td>
</tr>
<tr>
<td>getpass</td>
<td>read a password</td>
</tr>
<tr>
<td>getprotobyname</td>
<td>get protocol entry</td>
</tr>
<tr>
<td>getprotobynumber</td>
<td>get protocol entry</td>
</tr>
<tr>
<td>getprototent</td>
<td>get protocol entry</td>
</tr>
<tr>
<td>getpw</td>
<td>get name from uid</td>
</tr>
<tr>
<td>getpwent</td>
<td>get password file entry</td>
</tr>
<tr>
<td>getpwnam</td>
<td>get password file entry</td>
</tr>
<tr>
<td>getpwuid</td>
<td>get password file entry</td>
</tr>
<tr>
<td>getrpcport</td>
<td>get RPC port number</td>
</tr>
<tr>
<td>gets</td>
<td>get a string from a stream</td>
</tr>
<tr>
<td>getservbyname</td>
<td>get service entry</td>
</tr>
<tr>
<td>getservbyport</td>
<td>get service entry</td>
</tr>
<tr>
<td>getservent</td>
<td>get service entry</td>
</tr>
<tr>
<td>getw</td>
<td>get character or word from stream</td>
</tr>
<tr>
<td>getwd</td>
<td>get current working directory pathname</td>
</tr>
<tr>
<td>gmtime</td>
<td>convert date and time to ASCII</td>
</tr>
<tr>
<td>gtty</td>
<td>set and get terminal state (defunct)</td>
</tr>
<tr>
<td>havedisk</td>
<td>determine if remote machine has disk</td>
</tr>
<tr>
<td>htonl</td>
<td>convert values between host and network byte ord</td>
</tr>
<tr>
<td>htons</td>
<td>Euclidean distance</td>
</tr>
<tr>
<td>hypot</td>
<td>convert values between host and network byte ord</td>
</tr>
<tr>
<td>index</td>
<td>string operations</td>
</tr>
<tr>
<td>inet_addr</td>
<td>Internet address manipulation routines</td>
</tr>
<tr>
<td>inet_inaof</td>
<td>Internet address manipulation routines</td>
</tr>
<tr>
<td>inet_makeaddr</td>
<td>Internet address manipulation routines</td>
</tr>
<tr>
<td>inet_netof</td>
<td>Internet address manipulation routines</td>
</tr>
<tr>
<td>inet_network</td>
<td>Internet address manipulation routines</td>
</tr>
<tr>
<td>infnan</td>
<td>signals exceptions</td>
</tr>
<tr>
<td>initgroups</td>
<td>initialize group access list</td>
</tr>
<tr>
<td>initstate</td>
<td>better random number generator</td>
</tr>
<tr>
<td>insque</td>
<td>insert/remove element from a queue</td>
</tr>
<tr>
<td>isalnum</td>
<td>character classification macros</td>
</tr>
<tr>
<td>isalpha</td>
<td>character classification macros</td>
</tr>
<tr>
<td>isscii</td>
<td>character classification macros</td>
</tr>
<tr>
<td>isatty</td>
<td>find name of a terminal</td>
</tr>
<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>isprint</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>
</tbody>
</table>
INTRO (3)  UNIX Programmer's Manual  INTRO (3)

line  plot.3x  graphics interface
linemod  plot.3x  graphics interface
localtime  ctime.3  convert date and time to ASCII
log  exp.3m  natural logarithm
log10  exp.3m  logarithm to base 10
log1p  exp.3m  \log(1+x)
logb  ieee.3m  exponent extraction
longjmp  setjmp.3  non-local goto
malloc  malloc.3  memory allocator
mktemp  mktemp.3  make a unique file name
modf  frexp.3  split into mantissa and exponent
moncontrol  monitor.3  prepare execution profile
monitor  monitor.3  prepare execution profile
mont startup  monitor.3  prepare execution profile
mount  mount.3r  keep track of remotely mounted filesystems
move  plot.3x  graphics interface
nextkey  dbm.3x  data base subroutines
nice  nice.3c  set program priority
nlist  nlist.3  get entries from name list
ntohl  byteorder.3n  convert values between host and network byte order
ntohs  byteorder.3n  convert values between host and network byte order
opendir  directory.3  directory operations
openlog  syslog.3  control system log
openpl  plot.3x  graphics interface
pause  pause.3c  stop until signal
pclose  popen.3  initiate I/O to/from a process
perror  perror.3  system error messages
plot  plot.3x  graphics interface
popen  popen.3  initiate I/O to/from a process
pow  exp.3m  exponential \( x^{\text{**y}} \)
printf  printf.3s  formatted output conversion
psignal  psignal.3  system signal messages
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
qsort  qsort.3  quicker sort
rand  rand.3c  random number generator
random  random.3  better random number generator
rcmd  rcmd.3x  routines for returning a stream to a remote command
re_comp  regex.3  regular expression handler
re_exec  regex.3  regular expression handler
readdir  directory.3  directory operations
realloc  malloc.3  memory allocator
remque  insque.3  insert/remove element from a queue
rewind  fseek.3s  reposition a stream
rewinddir  directory.3  directory operations
rexec  rexec.3x  return stream to a remote command
rindex  string.3  string operations
rint  floor.3m  round to nearest integer
rnusers  rnusers.3r  return number of users on remote machine
rquota  rquota.3r  implement quotas on remote machines
rresvport  rcmd.3x  routines for returning a stream to a remote command
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rstat</td>
<td>get performance data from remote kernel</td>
</tr>
<tr>
<td>ruserok</td>
<td>routines for returning a stream to a remote command</td>
</tr>
<tr>
<td>rusers</td>
<td>return information about users on remote machines</td>
</tr>
<tr>
<td>rread</td>
<td>write to specified remote machines</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>directory operations</td>
</tr>
<tr>
<td>setbuffer</td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>setegid</td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>seteuid</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>setfsent</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>setgid</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>setgrent</td>
<td>get file system descriptor 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>setsent</td>
<td>get file system descriptor file entry</td>
</tr>
<tr>
<td>setuid</td>
<td>get group file entry</td>
</tr>
<tr>
<td>setuid</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>signal</td>
<td>get network entry</td>
</tr>
<tr>
<td>signal</td>
<td>get network entry</td>
</tr>
<tr>
<td>sin</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>sinh</td>
<td>get protocol entry</td>
</tr>
<tr>
<td>sleep</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>space</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>spray</td>
<td>get group file entry</td>
</tr>
<tr>
<td>sprintf</td>
<td>get password file entry</td>
</tr>
<tr>
<td>sqrt</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>srand</td>
<td>get network entry</td>
</tr>
<tr>
<td>random</td>
<td>get network entry</td>
</tr>
<tr>
<td>scanf</td>
<td>better random number generator</td>
</tr>
<tr>
<td>scanf</td>
<td>set user and group ID</td>
</tr>
<tr>
<td>stdio</td>
<td>simplified software signal facilities</td>
</tr>
<tr>
<td>store</td>
<td>trigonometric function</td>
</tr>
<tr>
<td>store</td>
<td>hyperbolic function</td>
</tr>
<tr>
<td>strcat</td>
<td>suspend execution for interval</td>
</tr>
<tr>
<td>strcmp</td>
<td>graphics interface</td>
</tr>
<tr>
<td>strcpy</td>
<td>scatter data in order to check the network</td>
</tr>
<tr>
<td>strlen</td>
<td>formatted output conversion</td>
</tr>
<tr>
<td>strncat</td>
<td>square root</td>
</tr>
<tr>
<td>strlen</td>
<td>random number generator</td>
</tr>
<tr>
<td>strncmp</td>
<td>random number generator</td>
</tr>
<tr>
<td>strncpy</td>
<td>better random number generator</td>
</tr>
<tr>
<td>stty</td>
<td>formatted input conversion</td>
</tr>
<tr>
<td>swab</td>
<td>standard buffered input/output package</td>
</tr>
<tr>
<td>stdio</td>
<td>data base subroutines</td>
</tr>
<tr>
<td>store</td>
<td>string operations</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>strncat</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>string operations</td>
</tr>
<tr>
<td>swab</td>
<td>set and get terminal state (defunct)</td>
</tr>
<tr>
<td>sys_errno</td>
<td>swap bytes</td>
</tr>
<tr>
<td>sys_errlist</td>
<td>system error messages</td>
</tr>
<tr>
<td>sys_errno</td>
<td>system error messages</td>
</tr>
<tr>
<td>sys_msglist</td>
<td>system signal messages</td>
</tr>
<tr>
<td>syslog</td>
<td>control system log</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</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>tgotostr</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>tbyname</td>
<td>find name of a terminal</td>
</tr>
<tr>
<td>ttyname</td>
<td>find name of a terminal</td>
</tr>
<tr>
<td>ttyslot</td>
<td>push character back into input stream</td>
</tr>
<tr>
<td>ungetc</td>
<td>set file times</td>
</tr>
<tr>
<td>utime</td>
<td>aligned memory allocator</td>
</tr>
<tr>
<td>valloc</td>
<td>variable argument list</td>
</tr>
<tr>
<td>vlimits</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>
<tr>
<td>ypppasswd</td>
<td>update user password in yellow pages</td>
</tr>
</tbody>
</table>
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.

BUGS
The abort() function does not flush standard I/O buffers. Use fflush(3S).
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

BUGS
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
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
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). On a VAX, acosh is accurate to about 3 ulps, asinh and atanh to about 2 ulps. An ulp is one Unit in the Last Place carried.

DIAGNOSTICS
Acosh returns the reserved operand on a VAX if the argument is less than 1.
Atanh returns the reserved operand on a VAX if the argument has absolute value bigger than or equal to 1.

SEE ALSO
math(3M), exp(3M), infnan(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)

BUGS
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.

BUGS
  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);
ulong netlong, hostlong;

netshort = htons(hostshort);
ushort netshort, hostshort;

hostlong = ntohl(netlong);
ulong hostlong, netlong;

hostshort = ntohs(netshort);
ushort hostshort, netshort;

DESCRIPTION
These routines convert 16 and 32 bit quantities between network byte order and 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)

BUGS
The VAX handles bytes backwards from most everyone else in the world. This is not expected to be fixed in the near future.
NAME
crypt, setkey, encrypt - DES encryption

SYNOPSIS
char *crypt(key, salt)
char *key, *salt;
setkey(key)
char *key;
encrypt(block, edflag)
char *block;

DESCRIPTION
Crypt is the password encryption routine. It is based on the NBS 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)

BUGS
The return value points to static data whose content is overwritten by each call.
NAME
cctime, localtime, gmtime, asctime, timezone – convert date and time to ASCII

SYNOPSIS
char *ctime(clock)
long *clock;
#include <time.h>
struct tm *localtime(clock)
long *clock;
struct tm *gmtime(clock)
long *clock;
char *asctime(tm)
struct tm *tm;
char *timezone(zone, dst)

DESCRIPTION
Ctime converts a time pointed to by clock such as returned by time(2) 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 UNIX 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:

```c
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 */
};
```

When local time is called for, the program consults the system to determine the time zone and whether the U.S.A., Australian, Eastern European, Middle European, or Western European daylight saving time adjustment is appropriate. The program knows about various peculiarities in time conversion over the past 10-20 years; if necessary, this understanding can be extended.

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
gmtimeofday(2), time(3)

BUGS
The return values point to static data whose content is overwritten by each call.

4th Berkeley Distribution May 27, 1986
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 (tildes)
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 -lcurses -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 initser() must be called before any of the other routines that
deal with windows and screens are used. The routine endwin() should be called before exit­
ing.

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
draw a box around a window
box(win,vert,hor)  set cbreak mode
cbreak()           clear stdscr
clrtobot()         set clear flag for scr
clear()            clear to bottom on stdscr
clearok(scr,boolt) clear to end of line on stdscr
delch()            delete a character
deleteln()         delete a line
delwin(win)        delete win
echo()             set echo mode
endwin()           end window modes
erase()            erase stdscr
flusok(win,boolt)  set flush-on-refresh flag for win
getch()            get a char through stdscr
getcap(name)       get terminal capability name
getsstr(str)       get a string through stdscr
getmode()          get tty modes
getyx(win,y,x)     get (y,x) co-ordinates
getchar()          get char at current (y,x) co-ordinates
initscr()          initialize screens
insch(c)           insert a char
insertln()         insert a line
leaveok(win,boolt) set leave flag for win
longname(termbuf,name) get long name from termbuf
move(y,x)          move to (y,x) on stdscr
mvcur(lasty,lastx,newy,newx) actually move cursor
newwin(lines,cols,begin_y,begin_x) create a new window
nl()               set newline mapping
nocbreak()         unset cbreak mode
noecho()           unset echo mode
nonl()             unset newline mapping
noraw()            unset raw mode
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>overlay(win1,win2)</td>
<td>overlay win1 on win2</td>
</tr>
<tr>
<td>overwrite(win1,win2)</td>
<td>overwrite win1 on top of win2</td>
</tr>
<tr>
<td>printf(fmt,arg1,arg2,...)</td>
<td>printf on stdscr</td>
</tr>
<tr>
<td>refresh()</td>
<td>set raw mode</td>
</tr>
<tr>
<td>resetty()</td>
<td>make current screen look like stdscr</td>
</tr>
<tr>
<td>savetty()</td>
<td>reset tty flags to stored value</td>
</tr>
<tr>
<td>scanw(fmt,arg1,arg2,...)</td>
<td>stored current tty flags</td>
</tr>
<tr>
<td>scroll(win)</td>
<td>scanf through stdscr</td>
</tr>
<tr>
<td>scrolllok(win,boolf)</td>
<td>scroll win one line</td>
</tr>
<tr>
<td>setterm(name)</td>
<td>set scroll flag</td>
</tr>
<tr>
<td>standend()</td>
<td>set term variables for name</td>
</tr>
<tr>
<td>standout()</td>
<td>end standout mode</td>
</tr>
<tr>
<td>subwin(win,lines,cols,beg_y,beg_x)</td>
<td>start standout mode</td>
</tr>
<tr>
<td>touche(line(win,y,sx,ex)</td>
<td>create a subwindow</td>
</tr>
<tr>
<td>touchoverlap(win1,win2)</td>
<td>mark overlap of win1 on win2 as changed</td>
</tr>
<tr>
<td>touchwin(win)</td>
<td>&quot;change&quot; all of win</td>
</tr>
<tr>
<td>unctrl(ch)</td>
<td>printable version of ch</td>
</tr>
<tr>
<td>waddch(win,ch)</td>
<td>add char to win</td>
</tr>
<tr>
<td>waddstr(win,str)</td>
<td>add string to win</td>
</tr>
<tr>
<td>wcrtobot(win)</td>
<td>clear win</td>
</tr>
<tr>
<td>wcrttyo(e)(win)</td>
<td>clear to bottom of win</td>
</tr>
<tr>
<td>wdelch(win,c)</td>
<td>clear to end of line on win</td>
</tr>
<tr>
<td>wdeleteln(win)</td>
<td>delete char from win</td>
</tr>
<tr>
<td>werase(win)</td>
<td>delete line from win</td>
</tr>
<tr>
<td>wgetch(win)</td>
<td>erase win</td>
</tr>
<tr>
<td>wgetstr(win,str)</td>
<td>get a char through win</td>
</tr>
<tr>
<td>winch(win)</td>
<td>get a string through win</td>
</tr>
<tr>
<td>winsch(win,c)</td>
<td>get char at current (y,x) in win</td>
</tr>
<tr>
<td>wininsertln(win)</td>
<td>insert char into win</td>
</tr>
<tr>
<td>wmove(win,y,x)</td>
<td>insert line into win</td>
</tr>
<tr>
<td>wprintf(win,fmt,arg1,arg2,...)</td>
<td>set current (y,x) co-ordinates on win</td>
</tr>
<tr>
<td>wrefresh(win)</td>
<td>printf on win</td>
</tr>
<tr>
<td>wscanw(fmt,arg1,arg2,...)</td>
<td>make screen look like win</td>
</tr>
<tr>
<td>wstandend(win)</td>
<td>scanf through win</td>
</tr>
<tr>
<td>wstandout(win)</td>
<td>end standout mode on win</td>
</tr>
<tr>
<td>wrefresh(win)</td>
<td>start standout mode on win</td>
</tr>
</tbody>
</table>

4th Berkeley Distribution

April 23, 1986

2
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 nextkey(key)
   datum key;

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))

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)

BUGS
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.

4th Berkeley Distribution May 12, 1986
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)

4.2 Berkeley Distribution September 24, 1985
NAME
ecvt, fcvt, gcvt - output conversion

SYNOPSIS
char *ecvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;
char *fcvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *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 decpt (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)

BUGS
The return values point to static data whose content is overwritten by each call.
NAME
end, etext, edata – last locations in program

SYNOPSIS
extern end;
extern etext;
extern edata;

DESCRIPTION
These names refer neither to routines nor to locations with interesting contents. The address
of etext is the first address above the program text, edata above the initialized data region,
and end above the uninitialized data region.

When execution begins, the program break coincides with end, but it is reset by the routines
brk(2), malloc(3), standard input/output (stdio(3S)), the profile (-p) option of cc(1), etc. The
current value of the program break is reliably returned by 'sbrk(0)', see brk(2).

SEE ALSO
brk(2), malloc(3)
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) := \frac{2}{\sqrt{\pi}} \int_{0}^{x} \exp(-t^2) 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)
ETHER (3R) UNIX Programmer's Manual

NAME
der — monitor traffic on the Ethernet

SYNOPSIS
#include <rpcsvc/ether.h>

RPC INFO
program number:
ETHERPROG
xdr routines:
   xdr_etherstat(xdrs, es)
     XDR *xdrs;
     struct etherstat *es;
   xdr_etheraddrs(xdrs, ea)
     XDR *xdrs;
     struct etheraddrs *ea;
   xdr_etherhtable(xdrs, hm)
     XDR *xdrs;
     struct etherhmem **hm;
   xdr_etherhmem(xdrs, hm)
     XDR *xdrs;
     struct etherhmem **hm;
   xdr_etherhtable(xdrs, hm)
     XDR *xdrs;
     struct etherhmem **hm;
   xdr_etherhbody(xdrs, hm)
     XDR *xdrs;
     struct etherhmem *hm;
   xdr_addrmask(xdrs, am)
     XDR *xdrs;
     struct addrmask *am;

Xdr_etherhmem processes a single etherhmem structure. Xdr_etherhtable processes an array of HASHSIZE *struct etherhmems. The *etherhmem field of etheraddrs is actually a hashtable, that is, it is a pointer to an array of HASHSIZE hmem pointers.

procs:
ETHERPROC_GETDATA
  no args, returns struct etherstat
ETHERPROC_ON
  no args or results, puts server in promiscuous mode
ETHERPROC_OFF
  no args or results, puts server in promiscuous mode
ETHERPROC_GETSRCDATA
  no args, returns struct etheraddrs with information about source of packets
ETHERPROC_GETDSTDATA
  no args, returns struct etheraddrs with information about destination of packets
ETHERPROC_SELECTSRC
  takes struct mask as argument, no results
  sets a mask for source
ETHERPROC_SELECTDST
  takes struct mask as argument, no results
  sets a mask for dst
ETHERPROC_SELECTPROTO
  takes struct mask as argument, no results
  sets a mask for proto
ETHERPROC_SELECTLNTH

takes struct mask as argument, no results
sets a mask for lnth

versions:
IELDVERS_ORIG

structures:
/
* all ether stat's except src, dst addresses
*/
struct etherstat {
    struct timeval e_time;
    unsigned long e_bytes;
    unsigned long e_packets;
    unsigned long e_bcast;
    unsigned long e_size[NBUCKETS];
    unsigned long e_proto[NPROTOS];
};
/
* member of address hash table
*/
struct etherhmem {
    int h_addr;
    unsigned h_cnt;
    struct etherhmem *h_nxt;
};
/
* src, dst address info
*/
struct etheraddr {
    struct timeval e_time;
    unsigned long e_bytes;
    unsigned long e_packets;
    unsigned long e_bcast;
    struct etherhmem **e_addrs;
};
/
* for size, a_addr is lowvalue, a_mask is high value
*/
struct addrmask {
    int a_addr;
    int a_mask:  /* 0 means wild card */
};

SEE ALSO
    traffic(?), etherfind(?), etherd(?)

7th Edition 10 August 1985
NAME
execl, execv, execlp, execvp, exec, execve, execlp, execvp, exec, execve, environ – execute a file

SYNOPSIS

execl(name, arg0, arg1, ..., argv, 0)
char *name, *arg0, *arg1, ..., *argv;

eencv(name, argv)
char *name, *argv[];

execl(name, arg0, arg1, ..., argv, 0, envp)
char *name, *arg0, *arg1, ..., *argv, *envp[];

exec(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 arguments 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 exec version is used when the executed file is to be manipulated with ptrace(2). The program is forced to single step a single instruction giving the parent an opportunity to manipulate its state. On the VAX-II this is done by setting the trace bit in the process status long-word.

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 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. 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 execlp 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.

BUGS
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;

double expm1(x)

double x;

double log(x)

double x;

double log10(x)

double x;

double log1p(x)

double x;

double pow(x,y)

double x,y;

DESCRIPTION
Exp returns the exponential function of x.
Expml returns exp(x)–1 accurately even for tiny x.
Log returns the natural logarithm of x.
Log10 returns the logarithm of x to base 10.
Log1p returns log(1+x) accurately even for tiny x.
Pow(x,y) returns \(x^y\).

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; that is 8 bits for VAX D and 11 bits for IEEE 754 Double. No such drastic loss has been exposed by testing; the worst errors observed have been below 20 ulps for VAX D, 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**56 on a VAX, 2**53 for IEEE 754.

DIAGNOSTICS
Exp, expm1 and pow return the reserved operand on a VAX when the correct value would overflow, and they set errno to ERANGE. Pow(x,y) returns the reserved operand on a VAX and sets errno to EDOM when x < 0 and y is not an integer.

On a VAX, errno is set to EDOM and the reserved operand is returned by log unless x > 0, by log1p unless x > -1.

NOTES
The functions exp(x)–1 and log(1+x) are called expm1 and log1p in BASIC on the Hewlett-Packard HP-71B and APPLE Macintosh, EXP1 and LN1 in Pascal, exp1 and log1 in C on APPLE Macintoshes, where they have been provided to make sure financial calculations of \(((1+x)\cdot n-1)/x\), namely expm1(n*log1p(x))/x, will be accurate when x is tiny. They also provide accurate inverse hyperbolic functions.
Pow(x,0) returns x**0 = 1 for all x including x = 0, ∞ (not found on a VAX), and NaN (the reserved operand on a VAX). Previous implementations of pow may have defined x**0 to be undefined in some or all of these cases. Here are reasons for returning x**0 = 1 always:

1. Any program that already tests whether x is zero (or infinite or NaN) before computing x**0 cannot care whether 0**0 = 1 or not. Any program that depends upon 0**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**0 = 1 for all x, including x = 0. This is compatible with the convention that accepts a[0] as the value of polynomial

   \[ p(x) = a[0]x^0 + a[1]x^1 + a[2]x^2 + \ldots + a[n]x^n \]

   at x = 0 rather than reject a[0]*0**0 as invalid.

3. Analysts will accept 0**0 = 1 despite that x**y can approach anything or nothing as x and y approach 0 independently. The reason for setting 0**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)**y(z) \to 1 as z \to 0.

4. If 0**0 = 1, then ∞**0 = 1/0**0 = 1 too; and then NaN**0 = 1 too because x**0 = 1 for all finite and infinite x, i.e., independently of x.

SEE ALSO
math(3M), infnan(3M)

AUTHOR
Kwok-Choi Ng, W. Kahan
NAME
fclose, flush - close or flush a stream

SYNOPSIS
#include <stdio.h>
fclose(stream)
FILE *stream;
fflush(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).

fflush 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
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;
double ceil(x)
double x;
double fabs(x)
double x;
double rint(x)
double x;

DESCRIPTION
Fabs returns the absolute value |x|.
Floor returns the largest integer no greater than x.
Ceil returns the smallest integer no less than x.
Rint returns the integer (represented as a double precision number) nearest x in the direction of the prevailing rounding mode.

NOTES
On a VAX, rint(x) is equivalent to adding half to the magnitude and then rounding towards zero.
In the default rounding mode, to nearest, on a machine that conforms to IEEE 754, 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;
Most C compilers round x towards 0 to get the integer k, but some do otherwise. If in doubt, use floor, ceil, or rint first, whichever you intend. 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.

SEE ALSO
abs(3), ieee(3M), math(3M)
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 preopened 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.

BUGS
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 malloc after 20 files have been opened. This confuses some programs which use their own memory allocators. An undocumented routine,
_fprealloc, may be called to force immediate allocation of all internal memory except for buffers.
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 out-
put 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

```c
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 \( \text{value} = x \cdot 2^n \) indirectly through `eptr`.

Ldexp returns the quantity \( \text{value} \cdot 2^\text{exp} \).

Modf returns the positive fractional part of `value` and stores the integer part indirectly through `iptr`. 

7th Edition May 15, 1985
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 equivalent to fseek (stream, 0L, 0).

SEE ALSO
lseek(2), fopen(3S)

DIAGNOSTICS
fseek returns -1 for improper seeks, otherwise zero.
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;
inw 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 feerror(3S) should be used to check the success of getw. Getw assumes no special align­
ment in the file.

SEE ALSO
clearerr(3S), fopen(3S), putc(3S), gets(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 clearerr(3S).

BUGS
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
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 (c) 1983 Regents of the University of California.
 * All rights reserved. The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 * @(#)disktab.h  5.2 (Berkeley) 10/1185
 */

/*
 * Disk description table, see disktab(5)
 */
#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 bad144 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[8];
};

struct disktab *getdiskbyname();

SEE ALSO
disktab(5)

BUGS
This information should be obtained from the system for locally available disks (in particular,
the disk partition tables).
NAME
getenv – value for environment name

SYNOPSIS
char *getenv(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, otherwise getenv returns the value 0 (NULL).

SEE ALSO
environ(7), execve(2)
NAME
getfsent, getfsspec, getfsfile, getfstype, 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 *getfstype(type)
char *type;
int setfsent()
int endfsent()

DESCRIPTION
Getfsent, getfsspec, getfstype, 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. Getfstype 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.

BUGS
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.

```c
/*
 * grp.h 4.1 83/05/03 *
*/

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 names.

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), getpwent(3), group(5)

DIAGNOSTICS
A null pointer (0) is returned on EOF or error.

BUGS
All information is contained in a static area so it must be copied if it is to be saved.
GETHOSTBYNAME(3N)  UNIX Programmer's Manual  GETHOSTBYNAME(3N)

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 struc-
ture. 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 run-
ing 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.

4.2 Berkeley Distribution  May 20, 1986
GETHOSTBYNAME(3N) UNIX Programmer's Manual GETHOSTBYNAME(3N)

TRY_AGAIN This is usually a temporary error and means that the local
er 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 database will not be closed after each call to gethostbyname or gethostbyaddr. Endhost-
tent is redefined to close the file.

BUGS
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
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
   getpwent(3), utmp(5), ttyslot(3)

DIAGNOSTICS
   Returns a NULL pointer (0) if name not found.

BUGS
   The return values point to static data whose content is overwritten by each call.
NAME
setmntent, getmntent, addmntent, endmntent, hasmntopt — 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 *hasmntopt(mnt, opt)
struct mntent *mnt;
char *opt;
int endmntent(filep)
FILE *filep;

DESCRIPTION
These routines replace the getfsent 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_fname; /* 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. Hasmntopt 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), getfsent(3)
DIAGNOSTICS
  Null pointer (0) returned on EOF or error.

BUGS
  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
Getnetent, 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.

Getnetent 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.
Getnetbyname 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.

BUGS
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 setnet­
grent 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(argc, argv, optstring)
int argc;
char **argv;
char *optstring;

extern char *optarg;
extern int optind;

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:

```c
main(argc, argv)
int argc;
char **argv;
{
    int c;
    extern int optind;
    extern char *optarg;

    while ((c = getopt(argc, argv, "afb:o:")) != EOF)
        switch (c) {
        case 'a':
            if (bflag)
                errflg++;
            else
                aflg++;
            break;
        case 'b':
            if (aflg)
                errflg++;
            else
                bproc();
            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.

BUGS
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)

BUGS
The return value points to static data whose content is overwritten by each call.
This page intentionally left almost blank.
NAME
getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent – get protocol entry

SYNOPSIS
#include <netdb.h>
struct protoent *getprotoent()
struct protoent *getprotobynumber(proto)
struct protoent *getprotobynumber(proto)
struct protoent *getprotobynumber(proto)
int proto;
setprotoent(stayopen)
int stayopen
endprotoent()
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 corresponding 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.

/* pwh h 4.1 83/05/03 */

struct passwd ( /* see getpwent(3) */
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. EOF 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 getpwent begin its search from the beginning of the database.

Setpwfile 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`, `getpwnam`, and `getpwuid`, return a null pointer (0) on EOF or error.

BUGS
All information is contained in a static area so it must be copied if it is to be saved.
NAME
getrpcent, getrpcbyname, getrpcbynumber — get rpc entry

SYNOPSIS
#include <netdb.h>
struct rpcent *getrpcent()
struct rpcent *getrpcbyname(name)
char *name;
struct rpcent *getrpcbynumber(number)
int number;
setrpcent(stayopen)
int stayopen
endrpcent();

DESCRIPTION
Getrpcent, getrpcbyname, and getrpcbynumber each return a pointer to an object with the
following structure containing the broken-out fields of a line in the rpc program number
data base, /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.
Getrpcent reads the next line of the file, opening the file if necessary.
Setrpcent opens and rewinds the file. If the stayopen flag is non-zero, the net data base will
not be closed after each call to getrpcent (either directly, or indirectly through one of the
other "getrpc" calls).
Endrpcent closes the file.
Getrpcbyname and getrpcbynumber 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.

BUGS
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, prognum, versnum, proto)
char *host;
int prognum, versnum, proto;

DESCRIPTION
Getrpcport returns the port number for version versnum of the RPC program prognum running on host and using protocol proto. It returns 0 if it cannot contact the portmapper, or if prognum is not registered. If prognum 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.

BUGS
Gets deletes a newline, fgets keeps it, all in the name of backward compatibility.
NAME
getservent, getservbyport, getservbyname, 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 follow­
ing 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.

BUGS
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
gettgyent, getttynam, setttyent, endttyent – get ttys file entry

SYNOPSIS
#include <ttyent.h>

struct ttyent *getttyent()

struct ttyent *getttynam(name)
char *name;

setttyent()

DESCRIPTION
Getttyent, 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 ttyent { /* see getttyent(3) */
    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 ttyent *getttyent();
extern struct ttyent *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.

Getttyent reads the next line from the ttys file, opening the file if necessary; setttyent 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.

BUGS
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.

BUGS
All information is contained in a static area so it must be copied if it is to be saved.
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;
double cabs(z)
struct {double x,y;} z;

DESCRIPTION
Hypot(x,y) and cabs(x,y) return sqrt(x*x+y*y) computed in such a way that underflow will not happen, and overflow occurs only if the final result deserves it.

hypot(∞,v) = hypot(v,∞) = +∞ for all v, including NaN.

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; with "reserved operand" in place of "NaN", the same is true on a VAX. But programmers on machines other than a VAX (it has no ∞) might be surprised at first to discover that hypot(±∞,NaN) = +∞. This is intentional; it happens because hypot(∞,v) = +∞ for all v, finite or infinite. Hence hypot(∞,v) is independent of v. Unlike the reserved operand on a VAX, the IEEE NaN is designed to disappear when it turns out to be irrelevant, as it does in hypot(∞,NaN).

SEE ALSO
math(3M), sqrt(3M)

AUTHOR
W. Kahan
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;
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| ≤ |y|/2. But drem(x, 0) is exceptional; see below under DIAGNOSTICS.
Finite(x) = 1 just when -∞ < x < +∞,
= 0 otherwise (when |x| = ∞ or x is NaN or
x is the VAX’s reserved operand.)
Logb(x) returns x’s exponent n, a signed integer converted to double-precision floating-point
and so chosen that 1 ≤ |x|/2**n < 2 unless x = 0 or (only on machines that conform to
IEEE 754) |x| = ∞ or x lies between 0 and the Underflow Threshold; see below under
“BUGS”.
Scalb(x, n) = x*(2**n) computed, for integer n, without first computing 2**n.

DIAGNOSTICS
IEEE 754 defines drem(x, 0) and drem(∞, y) to be invalid operations that produce a NaN. On
a VAX, drem(x, 0) returns the reserved operand. No ∞ exists on a VAX.
IEEE 754 defines logb(±∞) = +∞ and logb(0) = -∞, and requires the latter to signal
Division-by-Zero. But on a VAX, logb(0) = 1.0 - 2.0**31 = -2,147,483,647.0. And if the
correct value of scalb(x, n) would overflow on a VAX, it returns the reserved operand and sets
errno to ERANGE.

SEE ALSO
floor(3M), math(3M), infnan(3M)

AUTHOR
Kwok-Choi Ng

BUGS
Should drem(x, 0) and logb(0) on a VAX signal invalidity by setting errno = EDOM? Should
logb(0) return -1.7e38?
IEEE 754 currently specifies that \( \log_b(\text{denormalized no.}) = \log_b(\text{tiniest normalized no.} > 0) \) but the consensus has changed to the specification in the new proposed IEEE standard p854, namely that \( \log_b(x) \) satisfy

\[
1 \leq \text{scalb}(|x|, -\log_b(x)) < \text{Radix} \quad \ldots = 2 \text{ for IEEE 754}
\]

for every \( x \) except 0, \( \infty \) and \( \text{NaN} \). Almost every program that assumes 754's specification will work correctly if \( \log_b \) follows 854's specification instead.

IEEE 754 requires \( \text{copysign}(x, \text{NaN}) = \pm x \) but says nothing else about the sign of a \( \text{NaN} \). A \( \text{NaN} \) (Not a Number) is similar in spirit to the VAX's reserved operand, but very different in important details. Since the sign bit of a reserved operand makes it look negative,

\[
\text{copysign}(x, \text{reserved operand}) = -x;
\]

should this return the reserved operand instead?
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(ln)
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 addresses 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.

BUGS
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.
NAME
infnan – signals invalid floating-point operations on a VAX (temporary)

SYNOPSIS
#include <math.h>
double infnan(iarg)
int iarg;

DESCRIPTION
At some time in the future, some of the useful properties of the Infinities and NaNs in the
IEEE standard 754 for Binary Floating-Point Arithmetic will be simulated in UNIX on the
DEC VAX by using its Reserved Operands. Meanwhile, the Invalid, Overflow and
Divide-by-Zero exceptions of the IEEE standard are being approximated on a VAX by calls
to a procedure infnan in appropriate places in libm. When better exception-handling is
implemented in UNIX, only infnan among the codes in libm will have to be changed. And
users of libm can design their own infnan now to insulate themselves from future changes.

Whenever an elementary function code in libm has to simulate one of the aforementioned
IEEE exceptions, it calls infnan(iarg) with an appropriate value of iarg. Then a reserved
operand fault stops computation. But infnan could be replaced by a function with the same
name that returns some plausible value, assigns an apt value to the global variable errno, and
allows computation to resume. Alternatively, the Reserved Operand Fault Handler could be
changed to respond by returning that plausible value, etc. instead of aborting.

In the table below, the first two columns show various exceptions signaled by the IEEE stan-
dard, and the default result it prescribes. The third column shows what value is given to iarg
by functions in libm when they invoke infnan(iarg) under analogous circumstances on a VAX.
Currently infnan stops computation under all those circumstances. The last two columns
offer an alternative; they suggest a setting for errno and a value for a revised infnan to return.
And a C program to implement that suggestion follows.

<table>
<thead>
<tr>
<th>IEEE Signal</th>
<th>IEEE Default</th>
<th>iarg</th>
<th>errno</th>
<th>infnan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid</td>
<td>NaN</td>
<td>EDOM</td>
<td>EDOM</td>
<td>0</td>
</tr>
<tr>
<td>Overflow</td>
<td>±∞</td>
<td>ERANGE</td>
<td>ERANGE</td>
<td>HUGE</td>
</tr>
<tr>
<td>Div-by-0</td>
<td>±∞</td>
<td>±ERANGE</td>
<td>ERANGE or EDOM</td>
<td>±HUGE</td>
</tr>
</tbody>
</table>

(HUGE = 1.7e38 ... nearly 2.0**127)

ALTERNATIVE infnan:
#include <math.h>
#include <errno.h>
extern int errno;
double infnan(iarg)
int iarg;
{
    switch(iarg) {
    case ERANGE: errno = ERANGE; return(HUGE);
    case -ERANGE: errno = EDOM; return(-HUGE);
    default: errno = EDOM; return(0);
    }
}

SEE ALSO
math(3M), intro(2), signal(3).

ERANGE and EDOM are defined in <errno.h>. See intro(2) for explanation of EDOM and ERANGE.
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.

BUGS
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 be 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
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
These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS
On a VAX, negative arguments cause y0, y1 and yn to return the reserved operand and set errno to EDOM.

SEE ALSO
math(3M), infnan(3M)
NAME
lgamma - log gamma function

SYNOPSIS
#include <math.h>
double lgamma(x)
double x;

DESCRIPTION
Lgamma returns \( \ln |\Gamma(x)| \) where 
\[
\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} \, dt \quad \text{for } x > 0 \text{ and }
\]
\[
\Gamma(x) = \pi/(\Gamma(1-x)\sin(\pi x)) \quad \text{for } x < 1.
\]
The external integer signgam returns the sign of \( \Gamma(x) \).

IDIOSYNCRASIES
Do not use the expression signgam*exp(lgamma(x)) to compute \( g := \Gamma(x) \). Instead use a program like this (in C):
\[
lg = lgamma(x); \quad g = signgam*exp(lg);
\]
Only after lgamma has returned can signgam be correct. Note too that \( \Gamma(x) \) 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.

Only in the UNIX math library for C was the name gamma ever attached to \( \ln \Gamma \). Elsewhere, for instance in IBM's FORTRAN library, the name GAMMA belongs to \( \Gamma \) and the name ALGAMA to \( \ln \Gamma \) in single precision; in double the names are DGAMMA and DLGAMA. Why should C be different?

Archaeological records suggest that C's gamma originally delivered \( \ln(\Gamma(|x|)) \). 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 \( \Gamma(x) \) 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) Use the old math library, libom.
2) Add the following program to your others:
\[
#include <math.h>
double gamma(x)
double x;
{
    return (lgamma(x));
}
\]

DIAGNOSTICS
The reserved operand is returned on a VAX for negative integer arguments, errno is set to ERANGE; for very large arguments overflow/underflows will occur inside the lgamma routine.

SEE ALSO
math(3M), infnan(3M)
NAME
lib2648 - subroutines for the HP 2648 graphics terminal

SYNOPSIS
#include <stdio.h>

typedef char *bitmat;
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 newmat, 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 messing up the screen or the escape sequences being sent. Typical use of trace will include:

    switch (argv[1][1]) {
      case 'T':
        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.
area clear(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.
bit copy(dest, src, rows, cols) bitmat dest,
Copy a rows by cols bit matrix from src to (user provided) dest.
clear() Clear the alphanumeric display.
clear2()  
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.

error(msg) char *msg;
  Default error handler. Calls message(msg) and returns. This is called by certain routines 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.

goft()  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 function locally.

line(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.
lowleft()
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 HPIB.

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.)
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.

setset() Set the display to draw lines in normal (solid) mode. (This is reversed by inverse video mode.)

setxor() Set the display to draw lines in exclusive or mode.

csync() 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 csync, since rawchar and readline call sync() and fflush(stdout) automatically.

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.

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.

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 \([baser, basec]\). 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.

vidinv() Set inverse video mode.

vidnorm() Set normal video mode.

zermat(m, rows, cols) bitmat m;
Set the bit matrix \(m\) to all zeros.

zoomn(size)
Set the hardware zoom to value size, which can range from 1 to 15.

zoomoff() Turn zoom off. This forces the screen to zoom level 1 without affecting the current internal zoom number.

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

BUGS
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
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 previ-
ously 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.

BUGS
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.

`Alloc ` is machine dependent; its use is discouraged.
NAME
math – introduction to mathematical library functions

DESCRIPTION
These functions constitute the C math library, *libm*. The link editor searches this library under the “-lm” 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Appears on Page</th>
<th>Description</th>
<th>Error Bound (ULPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>acos</td>
<td>sin.3m</td>
<td>inverse trigonometric function</td>
<td>3</td>
</tr>
<tr>
<td>acosh</td>
<td>asinh.3m</td>
<td>inverse hyperbolic function</td>
<td>3</td>
</tr>
<tr>
<td>asin</td>
<td>sin.3m</td>
<td>inverse trigonometric function</td>
<td>3</td>
</tr>
<tr>
<td>asinh</td>
<td>asinh.3m</td>
<td>inverse hyperbolic function</td>
<td>3</td>
</tr>
<tr>
<td>atan</td>
<td>sin.3m</td>
<td>inverse trigonometric function</td>
<td>1</td>
</tr>
<tr>
<td>atanh</td>
<td>asinh.3m</td>
<td>inverse hyperbolic function</td>
<td>3</td>
</tr>
<tr>
<td>atan2</td>
<td>sin.3m</td>
<td>inverse trigonometric function</td>
<td>2</td>
</tr>
<tr>
<td>cabs</td>
<td>hypot.3m</td>
<td>complex absolute value</td>
<td>1</td>
</tr>
<tr>
<td>cbrt</td>
<td>sqrt.3m</td>
<td>cube root</td>
<td>1</td>
</tr>
<tr>
<td>ceil</td>
<td>floor.3m</td>
<td>integer no less than</td>
<td>0</td>
</tr>
<tr>
<td>copysign</td>
<td>ieee.3m</td>
<td>copy sign bit</td>
<td>0</td>
</tr>
<tr>
<td>cos</td>
<td>sin.3m</td>
<td>trigonometric function</td>
<td>1</td>
</tr>
<tr>
<td>cosh</td>
<td>sinh.3m</td>
<td>hyperbolic function</td>
<td>3</td>
</tr>
<tr>
<td>drem</td>
<td>ieee.3m</td>
<td>remainder</td>
<td>0</td>
</tr>
<tr>
<td>erf</td>
<td>erf.3m</td>
<td>error function</td>
<td>??</td>
</tr>
<tr>
<td>erfc</td>
<td>erf.3m</td>
<td>complementary error function</td>
<td>??</td>
</tr>
<tr>
<td>exp</td>
<td>exp.3m</td>
<td>exponential</td>
<td>1</td>
</tr>
<tr>
<td>expm1</td>
<td>exp.3m</td>
<td>exp(x)-1</td>
<td>1</td>
</tr>
<tr>
<td>fabs</td>
<td>floor.3m</td>
<td>absolute value</td>
<td>0</td>
</tr>
<tr>
<td>floor</td>
<td>floor.3m</td>
<td>integer no greater than</td>
<td>0</td>
</tr>
<tr>
<td>hypot</td>
<td>hypot.3m</td>
<td>Euclidean distance</td>
<td>1</td>
</tr>
<tr>
<td>infnan</td>
<td>infnan.3m</td>
<td>signals exceptions</td>
<td></td>
</tr>
<tr>
<td>j0</td>
<td>j0.3m</td>
<td>bessel function</td>
<td>??</td>
</tr>
<tr>
<td>j1</td>
<td>j0.3m</td>
<td>bessel function</td>
<td>??</td>
</tr>
<tr>
<td>jn</td>
<td>j0.3m</td>
<td>bessel function</td>
<td>??</td>
</tr>
<tr>
<td>lgamma</td>
<td>lgamma.3m</td>
<td>log gamma function; (formerly gamma.3m)</td>
<td></td>
</tr>
<tr>
<td>log</td>
<td>exp.3m</td>
<td>natural logarithm</td>
<td>1</td>
</tr>
<tr>
<td>logb</td>
<td>ieee.3m</td>
<td>exponent extraction</td>
<td>0</td>
</tr>
<tr>
<td>log10</td>
<td>exp.3m</td>
<td>logarithm to base 10</td>
<td>3</td>
</tr>
<tr>
<td>log1p</td>
<td>exp.3m</td>
<td>log(1+x)</td>
<td>1</td>
</tr>
<tr>
<td>pow</td>
<td>exp.3m</td>
<td>exponential x**y</td>
<td>60–500</td>
</tr>
<tr>
<td>rint</td>
<td>floor.3m</td>
<td>round to nearest integer</td>
<td>0</td>
</tr>
<tr>
<td>scalb</td>
<td>ieee.3m</td>
<td>exponent adjustment</td>
<td>0</td>
</tr>
<tr>
<td>sin</td>
<td>sin.3m</td>
<td>trigonometric function</td>
<td>1</td>
</tr>
<tr>
<td>sinh</td>
<td>sinh.3m</td>
<td>hyperbolic function</td>
<td>3</td>
</tr>
<tr>
<td>sqrt</td>
<td>sqrt.3m</td>
<td>square root</td>
<td>1</td>
</tr>
<tr>
<td>tan</td>
<td>sin.3m</td>
<td>trigonometric function</td>
<td>3</td>
</tr>
<tr>
<td>tanh</td>
<td>sinh.3m</td>
<td>hyperbolic function</td>
<td>3</td>
</tr>
<tr>
<td>y0</td>
<td>j0.3m</td>
<td>bessel function</td>
<td>??</td>
</tr>
<tr>
<td>y1</td>
<td>j0.3m</td>
<td>bessel function</td>
<td>??</td>
</tr>
<tr>
<td>yn</td>
<td>j0.3m</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:

\[
\begin{align*}
\text{sqrt}(-1.0) & = 0.0 \text{ and } \log(-1.0) = -1.7e38. \\
\cos(1.0e-11) & > \cos(0.0) > 1.0. \\
pow(x,1.0) & \neq x \text{ when } x = 2.0, 3.0, 4.0, ..., 9.0. \\
pow(-1.0,1.0e10) & \text{ trapped on Integer Overflow.} \\
sqrt(1.0e30) \text{ and } \sqrt(1.0e-30) & \text{ were very slow.}
\end{align*}
\]

However the two versions do differ in ways that have to be explained, to which end the following notes are provided.

**DEC VAX–11 D_floating-point:**

This is the format for which the original math library libm was developed, and to which this manual is still principally dedicated. It is the double-precision format for the PDP–11 and the earlier VAX–11 machines; VAX–11s after 1983 were provided with an optional "G" format closer to the IEEE double-precision format. The earlier DEC MicroVAXs have no D format, only G double-precision. (Why? Why not?)

Properties of D_floating-point:

- **Wordsize:** 64 bits, 8 bytes. **Radix:** Binary.
- **Precision:** 56 significant bits, roughly like 17 significant decimals.
  - If \( x \) and \( x' \) are consecutive positive D_floating-point numbers (they differ by 1 ulp), then
    \[
    1.3e-17 < 0.5*56 < (x'-x)/x \leq 0.5*55 < 2.8e-17.
    \]
- **Range:**
  - **Overflow threshold** = \( 2.0*127 = 1.7e38. \)
  - **Underflow threshold** = \( 0.5*128 = 2.9e-39. \)

**NOTE:** THIS RANGE IS COMPARATIVELY NARROW.
- **Overflow customarily stops computation.**
- **Underflow is customarily flushed quietly to zero.**

**CAUTION:**

- It is possible to have \( x \neq y \) and yet \( x-y = 0 \) because of underflow.
- Similarly \( x > y > 0 \) cannot prevent either \( x*y = 0 \) or \( y/x = 0 \) from happening without warning.

Zero is represented ambiguously.
- Although \( 2^{55} \) different representations of zero are accepted by the hardware, only the obvious representation is ever produced. There is no \(-0\) on a VAX.

\( \infty \) is not part of the VAX architecture.

**Reserved operands:**

- of the \( 2^{55} \) that the hardware recognizes, only one of them is ever produced.
- Any floating-point operation upon a reserved operand, even a MOVF or MOVD, customarily stops computation, so they are not much used.

**Exceptions:**

- Divisions by zero and operations that overflow are invalid operations that customarily stop computation or, in earlier machines, produce reserved operands that will stop computation.

**Rounding:**

Every rational operation (+, −, *, /) on a VAX (but not necessarily on a PDP–11), if not an overflow/underflow nor division by zero, is rounded to within half an ulp, and when the rounding error is exactly half an ulp then rounding...
is away from 0.

Except for its narrow range, D_floating-point is one of the better computer arithmetics designed in the 1960’s. Its properties are reflected fairly faithfully in the elementary functions for a VAX distributed in 4.3 BSD. They over/underflow only if their results have to lie out of range or very nearly so, and then they behave much as any rational arithmetic operation that over/underflowed would behave. Similarly, expressions like log(0) and atanh(1) behave like 1/0; and sqrt(-3) and acos(3) behave like 0/0; they all produce reserved operands and/or stop computation! The situation is described in more detail in manual pages.

This response seems excessively punitive, so it is destined to be replaced at some time in the foreseeable future by a more flexible but still uniform scheme being developed to handle all floating-point arithmetic exceptions neatly. See infnan(3M) for the present state of affairs.

How do the functions in 4.3 BSD’s new libm for UNIX compare with their counterparts in DEC’s VAX/VMS library? Some of the VMS functions are a little faster, some are a little more accurate, some are more puritanical about exceptions (like pow(0.0,0.0) and atan2(0.0,0.0)), and most occupy much more memory than their counterparts in libm. The VMS codes interpolate in large table to achieve speed and accuracy; the libm codes use tricky formulas compact enough that all of them may some day fit into a ROM.

More important, DEC regards the VMS codes as proprietary and guards them zealously against unauthorized use. But the libm codes in 4.3 BSD 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 whose arithmetic resembles VAX D_floating-point need use anything worse than the new libm.

IEEE STANDARD 754 Floating-Point Arithmetic:

This standard is on its way to becoming more widely adopted than any other design for computer arithmetic. VLSI chips that conform to some version of that standard have been produced by a host of manufacturers, among them ...

- Intel i8087, i80287
- Motorola 68881
- Zilog Z8070
- National Semiconductor 32081
- Weitek WTL-1032, ..., -1165
- Western Electric (AT&T) WE32106.

Other implementations range from software, done thoroughly in the Apple Macintosh, through VLSI in the Hewlett-Packard 9000 series, to the ELXSI 6400 running ECL at 3 Megaflops. Several other companies have adopted the formats of IEEE 754 without, alas, adhering to the standard’s way of handling rounding and exceptions like over/underflow. The DEC VAX G_floating-point format is very similar to the IEEE 754 Double format, so similar that the C programs for the IEEE versions of most of the elementary functions listed above could easily be converted to run on a MicroVAX, though nobody has volunteered to do that yet.

The codes in 4.3 BSD’s libm for machines that conform to IEEE 754 are intended primarily for the National Semi. 32081 and WTL 1164/65. To use these codes with the Intel or Zilog chips, or with the Apple Macintosh or ELXSI 6400, is to forego the use of better codes provided (perhaps freely) by those companies and designed by some of the authors of the codes above. Except for atan, cabs, cbt, erf, erfc, hypot, j0-jn, lgamma, pow and y0-y-n, the Motorola 68881 has all the functions in libm on chip, and faster and more accurate; it, Apple, the i8087, Z8070 and WE32106 all use 64 significant bits. 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 \( \text{ulp} \)), then
  \[ 1.1 \times 10^{-16} < 0.5 \times 53 < (x'-x)/x \leq 0.5 \times 52 < 2.3 \times 10^{-16}. \]

Range:
- Overflow threshold = \( 2.0 \times 10^{24} = 1.8 \times 308 \)
- Underflow threshold = \( 0.5 \times 10^{22} = 2.2 \times 308 \)
- Overflow goes by default to a signed \( \infty \).
- Underflow is \( \text{Gradual} \), rounding to the nearest integer multiple of \( 0.5 \times 10^{74} = 4.9 \times 324 \).

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 copysign(\( x, \pm 0 \)). In particular, comparison (\( x > y, x \geq y \), 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 \( (\text{finite})/\infty = \text{finite} / 0 = \text{finite} \). But \( \infty - \infty \), \( \infty \times 0 \) and \( \infty / \infty \) are, like \( 0/0 \) and \( \text{sqrt}(-3) \), invalid operations that produce \text{NaN}.

Reserved operands:
- there are \( 2^{53-2} \) of them, all called \text{NaN} (Not a Number). Some, called \text{Signaling} \text{NaNs}, trap any floating-point operation performed upon them; they are used to mark missing or uninitialized values, or nonexistent elements of arrays. The rest are \text{Quiet} \text{NaNs}; they are the default results of \text{Invalid Operations}, and propagate through subsequent arithmetic operations. If \( x \neq x \) then \( x \text{ is } \text{NaN} \); every other predicate (\( x > y, x = y, x < y, ... \)) is \text{FALSE} if \text{NaN} is involved.

NOTE: Trichotomy is violated by \text{NaN}.
- Besides being \text{FALSE}, predicates that entail ordered comparison, rather than mere (in)equality, signal \text{Invalid Operation} when \text{NaN} is involved.

Rounding:
- Every algebraic operation (+, -, *, /, \( \sqrt{\cdot} \)) is rounded by default to within half an \( \text{ulp} \), and when the rounding error is exactly half an \( \text{ulp} \) 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 \( x = 1.0, 2.0, 3.0, 4.0, ... \), \( 2.0 \times 52 \), we find \( (x/3.0) \times 3.0 = x \) and \( (x/10.0) \times 10.0 = x \) and ... 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 \( 1.0 \times 10^{-10} \) and \( 1.0 \times 37 \).

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>\text{NaN}, or \text{FALSE}</td>
</tr>
</tbody>
</table>
Overflow  ±∞  
Divide by Zero  ±∞  
Underflow  Gradual Underflow  
Inexact  Rounded value  

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 ≠ 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 solution 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 indivisible, or atomic, in the sense that ...

i) No exception should be signaled that is not deserved by the data supplied to that function.

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 program 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 exception 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.

**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**

An explanation of IEEE 754 and its proposed extension p854 was published in the IEEE magazine MICRO in August 1984 under the title "A Proposed Radix-- and Word-length--independent Standard for Floating-point Arithmetic" by W. J. Cody et al. The manuals for Pascal, C and BASIC on the Apple Macintosh document the features of IEEE 754 pretty well. Articles in the IEEE magazine COMPUTER vol. 14 no. 3 (Mar. 1981), and in the ACM SIGNUM Newsletter Special Issue of Oct. 1979, may be helpful although they pertain to superseded drafts of the standard.
AUTHOR

W. Kahan, with the help of Z-S. Alex Liu, Stuart I. McDonald, Dr. Kwok-Choi Ng, Peter Tang.
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 con­
  tain 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
  Mkstemp 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 (*lowpc)(), (*highpc)();
short buffer[];

monstartup(lowpc, highpc)
int (*lowpc)(), (*highpc)();

moncontrol(mode)

DESCRIPTION

There are two different forms of monitoring available: An executable program created by:

    cc -p ...

automatically includes calls for the prof(1) monitor and includes an initial call to its start-up routine monstartup with default parameters; monitor need not be called explicitly except to gain fine control over profil buffer allocation. An executable program created by:

    cc -pg ...

automatically includes calls for the gprof(1) monitor.

Monstartup is a high level interface to profil(2). Lowpc and highpc specify the address range that is to be sampled; the lowest address sampled is that of lowpc and the highest is just below highpc. Monstartup allocates space using sbrk(2) and passes it to monitor (see below) to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. Only calls of functions compiled with the profiling option -p of cc(1) are recorded.

To profile the entire program, it is sufficient to use

    extern etext();
    ...
    monstartup((int) 2, etext);

Ettext lies just above all the program text, see end(3).

To stop execution monitoring and write the results on the file mon.out, use

    monitor(0);
then prof(1) can be used to examine the results.

Moncontrol is used to selectively control profiling within a program. This works with either prof(1) or gprof(1) type profiling. When the program starts, profiling begins. To stop the collection of histogram ticks and call counts use moncontrol(0); to resume the collection of histogram ticks and call counts use moncontrol(1). This allows the cost of particular operations to be measured. Note that an output file will be produced upon program exit irregardless of the state of moncontrol.

Monitor is a low level interface to profil(2). Lowpc and highpc are the addresses of two functions; buffer is the address of a (user supplied) array of bufsize short integers. At most nfunc call counts can be kept. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled. Monitor divides the buffer into space to record the histogram of program counter samples over the range lowpc to highpc, and space to record call counts of functions compiled with the -p option to cc(1).
To profile the entire program, it is sufficient to use

```c
extern etext();
...
monitor((int) 2, etext, buf, bufsize, nfunc);
```

**FILES**

`mon.out`

**SEE ALSO**

`cc(1), prof(1), gprof(1), profil(2), sbrk(2)`
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:

Sun Microsystems Rel 3.0 10 August 1985
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), NFS Protocol Spec, section 3.
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. pow calculates in c the value of a raised to the ("regular" integral) power n,
while rpow 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 \texttt{-Imp}.

**FILES**

- `/usr/include/mp.h` include file
- `/usr/lib/libmp.a` object code library

**SEE ALSO**

- dc(1), bc(1)

**DIAGNOSTICS**

Illegal operations and running out of memory produce messages and core images.

**BUGS**

- Bases for input and output should be \(<= 10\).
- \texttt{dc(1)} and \texttt{bc(1)} don't use this library.
- The input and output routines are a crock.
- \texttt{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
database.

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.

**BUGS**

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[];

DESCRIPTION
  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. See a.out(5) for the structure
  declaration.

  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(cp)
char *cp;

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 hexade-
cimal 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 mil-
lenia. 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 is 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 BUGS).

BUGS
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
    during a pause, the pause call will return.

RETURN VALUE
    Always returns -1.

ERRORS
    Pause always returns:
    [EINVAL]  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 external 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 provided; 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
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)
arc(x, y, x0, y0, x1, y1)
move(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:

-plot 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
-lvt0 DEC vt100 terminals
-lplot2648 Hewlett Packard 2648 graphics terminal
-lplot7221 Hewlett Packard 7221 graphics terminal
-lplotimage Imagen laser printer (default 240 dots-per-inch resolution).
On many devices, it is necessary to pause after \texttt{erase()}, otherwise plotting commands are lost. The pause is normally done by the tty driver if at login time, \texttt{tset} found a \texttt{df} field in the \texttt{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 \texttt{erase()}.

SEE ALSO
\texttt{plot(5)}, \texttt{plot(1G)}, \texttt{plot(3F)}, \texttt{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.

BUGS
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, sprintf – formatted output conversion

SYNOPSIS
#include <stdio.h>
printf(format [, arg ] ... )
char *format;

fprintf(stream, format [, arg ] ... )
FILE *stream;
char *format;

sprintf(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. Sprintf places ‘output’ in the string s, followed by the character ‘\0’.
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 con­
trol 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 print/.

Each conversion specification is introduced by the character %. The remainder of the conver­
sion 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 e, d, s, and u, conversions, this option has no effect. For o
    conversions, the precision of the number is increased to force the first chara­
ccter 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 (nor­
mally, 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 char­
 acters 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;
• an optional period '.' which serves to separate the field width from the next digit string;
• 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;
• the character I specifying that a following d, o, x, or u corresponds to a long integer arg.
• 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

do  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.ddde±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 MAXUINT, where MAXUINT equals 4294967295 on a VAX-II and 65535 on a PDP-11).

% 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:

```
printf("%s, %s %d, %02d:%02d", weekday, month, day, hour, min);
```

To print π to 5 decimals:

```
printf("pi = %.5f", 4*atan(1.0));
```

SEE ALSO
putc(3S), scanf(3S), ecvt(3)

BUGS
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 newline.
  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 provided;
  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)
putc, putchar, fputc, putw — put character or word on a stream

SYNOPSIS

```c
#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.

BUGS

Because it is implemented as a macro, putc treats a stream argument with side effects improperly. In particular
putc(c, *f++);

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

BUGS
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
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
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
$16 \times (2^{31} - 1)$.

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
1 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 argu­
ment. 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 set­
state and srandom is that the size of the state array does not have to be remembered after it is
initialized.

With 256 bytes of state information, the period of the random number generator is greater
than $2^{69}$, which should be sufficient for most purposes.

AUTHOR
Earl T. Cohen
DIAGNOSTICS
If `intstate` 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)`

BUGS
About 2/3 the speed of `rand(3C)`.
NAME
rcmd, rresvport, 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 = rresvport(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. Rresvport 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 rshd(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 rshd(8C).

The rresvport 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 current working directory (normally the local 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), rexecd(8C), rlogind(8C), rshd(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 char-
  acters; 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_expant – 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, eomorig, comp_dn, exp_dn, length)
char *msg, *eomorig, *comp_dn, exp_dn;
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_AAONLY
Accept authoritative answers only. Res_send will continue until it finds an authorita-
tive 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 buflen. 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. Dnptrs is a list of pointers to previously compressed names in the current message. The first pointer points to 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
NAME
rexec – return stream to a remote command

SYNOPSIS
rem = rexec(ahost, inport, user, passwd, cmd, fd2p);
char *ahost;
int inport;
char *user, *passwd, *cmd;
int *fd2p;

DESCRIPTION
Rexec 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. 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 rexeced(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), rexeced(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 suc-
cessful. 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 **uia_arr;
    int uia_cnt
};

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 utmp **up;
int xdr_utmpidleptr(xdrs, up);
    XDR *xdrs;
    struct utmpidle **up;
int xdr_utmparr(xdrs, up);
    XDR *xdrs;
    struct utmparr *up;
int xdr_utmpidlearr(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(1), rusersd(8c)
NAME
rquota — implement quotas on remote machines

SYNOPSIS
#include <rpcsvc/rquota.h>

RPC INFO
program number:
RQUOTAPROG
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:
RQUOTAPROC_GETQUOTA
RQUOTAPROC_GETACTIVEQUOTA
Arguments of struct getquota_args.
Returns struct getquota_rslt.
Uses UNIX authentication.
Returns quota only on filesystems with quota active.

versions:
RQUOTAVERS_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), quotact1(2)
NAME
havedisk, rstat — get performance data from remote kernel

SYNOPSIS
#include <rpcsvc/rstat.h>
havedisk(host)
    char *host;
rstat(host, statp)
    char *host;
    struct statstime *statp;

DESCRIPTION
Havedisk returns 1 if host has a disk, 0 if it does not, and —1 if this cannot be determined.
Rstat fills in the statstime structure for host, and returns 0 if it was successful. The relevant structures are:

```
struct stats {
    /* RSTATVERS_ORIG */
    int cp_time[CPUSTATES];
    int dk_xfer[DK_NDRIYE];
    unsigned v_pgpgin; /* these are cumulative sum */
    unsigned v_pgpgout;
    unsigned v_pswpin;
    unsigned v_pswpout;
    unsigned v_intr;
    int if_ipackets;
    int if_ierrors;
    int if_opackets;
    int if_oerrors;
    int if_collisions;
};
```

```
struct statsswitch {
    /* RSTATVERS_SWITCH */
    int cp_time[CPUSTATES];
    int dk_xfer[DK_NDRIYE];
    unsigned v_pgpgin; /* these are cumulative sum */
    unsigned v_pgpgout;
    unsigned v_pswpin;
    unsigned v_pswpout;
    unsigned v_intr;
    int if_ipackets;
    int if_ierrors;
    int if_opackets;
    int if_oerrors;
    int if_collisions;
    unsigned v_swtxch;
    long avenrun[3];
    struct timeval boottime
};
```

```
struct statstime {
    /* RSTATVERS_TIME */
    int cp_time[CPUSTATES];
    int dk_xfer[DK_NDRIYE];
    unsigned v_pgpgin; /* these are cumulative sum */
    unsigned v_pgpgout;
    unsigned v_pswpin;
    unsigned v_pswpout;
```

7th Edition 10 August 1985
unsigned v__intr;
int if__ipackets;
int if__errors;
int if__opackets;
int if__oerrors;
int if__collisions;
unsigned v__swtch;
long avenrun[3];
struct timeval boottime;
struct timeval curtime;
};

RPC INFO

program number:
RSTATPROG

xdr routines:
int xdr_stats(xdrs, stat)
  XDR *xdrs;
  struct stats *stat;
int xdr_statswitch(xdrs, stat)
  XDR *xdrs;
  struct statswitch *stat;
int xdr_statsctime(xdrs, stat)
  XDR *xdrs;
  struct statctime *stat;
int xdr_timeval(xdrs, tv)
  XDR *xdrs;
  struct timeval *tv;

procs:
RSTATPROC_HAVEDISK
  Takes no arguments, returns long which is true if remote host has a disk.
RSTATPROC_STATS
  Takes no arguments, return struct statxxx, depending on version.

versions:
RSTATVERS_ORIG
RSTATVERS_SWITCH
RSTATVERS_TIME

SEE ALSO
perfmeter(1), rup(1), rstatd(8c)
NAME
rwall — write to specified remote machines

SYNOPSIS
#include <rpcsvc/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)
This page intentionally left almost blank.
NAME
scandir, alphasort — scan a directory

SYNOPSIS
#include <sys/types.h>
#include <sys/dir.h>
srandir(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
scanf, 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 a 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.
c 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 ‘%ls’. 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 I 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 I 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 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

```
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,

```
int i; float x; char name[50];
scanf("%2d%f%*d%[1234567890r", &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.

BUGS

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. `fflush` (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)`

BUGS

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.

  Set jmp saves its stack environment in env for later use by long jmp. It returns value 0.

  Long jmp restores the environment saved by the last call of set jmp. It then returns in such a
  way that execution continues as if the call of set jmp had just returned the value val to the
  function that invoked set jmp, which must not itself have returned in the interim. All accessible
  data have values as of the time long jmp was called.

  Set jmp and long jmp save and restore the signal mask sigmask(2), while _set jmp and _long jmp
  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, long jmp calls the routine longjmperror. If longjmperror returns the program
  is aborted. The default version of longjmperror prints the message “long jmp botch” to stan-
  dard 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)
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
    setreuid(2), setregid(2), getuid(2), getgid(2)

DIAGNOSTICS
    Zero is returned if the user (group) ID is set; -1 is returned otherwise.
NAME
siginterrupt - allow signals to interrupt system calls

SYNOPSIS
siginterrupt(sig, flag);
int sig, flag;

DESCRIPTION
Siginterrrupt 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>

(*signal(sig, func))()
int (*func);)

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
SIGURG 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
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 (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)

**NOTES (VAX-11)**

The handler routine can be declared:

```c
handler(sig, code, scp)
```

Here _sig_ is the signal number, into which the hardware faults and traps are mapped as defined below. Code is a parameter which is either a constant as given below or, for compatibility mode faults, the code provided by the hardware. _Scp_ is a pointer to the struct sigcontext used by the system to restore the process context from before the signal. Compatibility mode faults are distinguished from the other SIGILL traps by having PSL_CM set in the psl.

The following defines the mapping of hardware traps to signals and codes. All of these symbols are defined in `<signal.h>`:

<table>
<thead>
<tr>
<th>Hardware condition</th>
<th>Signal</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic traps:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer overflow</td>
<td>SIGFPE</td>
<td>FPE_INTOVF_TRAP</td>
</tr>
<tr>
<td>Integer division by zero</td>
<td>SIGFPE</td>
<td>FPE_INTDIV.TRAP</td>
</tr>
<tr>
<td>Floating overflow trap</td>
<td>SIGFPE</td>
<td>FPE_FLTOVF.TRAP</td>
</tr>
<tr>
<td>Floating/decimal division by zero</td>
<td>SIGFPE</td>
<td>FPE_FLTDIV.TRAP</td>
</tr>
<tr>
<td>Floating underflow trap</td>
<td>SIGFPE</td>
<td>FPE_FLTUND.TRAP</td>
</tr>
<tr>
<td>Decimal overflow trap</td>
<td>SIGFPE</td>
<td>FPE_DECOFV.TRAP</td>
</tr>
<tr>
<td>Subscript-range</td>
<td>SIGFPE</td>
<td>FPE_SUBRNG.TRAP</td>
</tr>
<tr>
<td>Floating overflow fault</td>
<td>SIGFPE</td>
<td>FPE_FLTOVF_FAULT</td>
</tr>
</tbody>
</table>

4th Berkeley Distribution May 20, 1986
<table>
<thead>
<tr>
<th>Condition</th>
<th>Signal</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating divide by zero fault</td>
<td>SIGFPE</td>
<td>FPE_FLTDIV_FAULT</td>
</tr>
<tr>
<td>Floating underflow fault</td>
<td>SIGFPE</td>
<td>FPE_FLTUND_FAULT</td>
</tr>
<tr>
<td>Length access control</td>
<td>SIGSEGV</td>
<td></td>
</tr>
<tr>
<td>Protection violation</td>
<td>SIGBUS</td>
<td></td>
</tr>
<tr>
<td>Reserved instruction</td>
<td>SIGILL</td>
<td>ILL_RESAD_FAULT</td>
</tr>
<tr>
<td>Customer-reserved instr.</td>
<td>SIGEMT</td>
<td></td>
</tr>
<tr>
<td>Reserved operand</td>
<td>SIGILL</td>
<td>ILL_PRIVIN_FAULT</td>
</tr>
<tr>
<td>Reserved addressing</td>
<td>SIGILL</td>
<td>ILL_RESOP_FAULT</td>
</tr>
<tr>
<td>Trace pending</td>
<td>SIGTRAP</td>
<td></td>
</tr>
<tr>
<td>Bpt instruction</td>
<td>SIGTRAP</td>
<td></td>
</tr>
<tr>
<td>Compatibility-mode</td>
<td>SIGILL</td>
<td></td>
</tr>
<tr>
<td>Chme</td>
<td>SIGSEGV</td>
<td></td>
</tr>
<tr>
<td>Chms</td>
<td>SIGSEGV</td>
<td></td>
</tr>
<tr>
<td>Chmu</td>
<td>SIGSEGV</td>
<td></td>
</tr>
</tbody>
</table>

*hardware supplied code*
NAME
sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions and their inverses

SYNOPSIS
#include <math.h>

double sin(x)
double x;
double cos(x)
double x;
double tan(x)
double x;
double asin(x)
double x;
double acos(x)
double x;
double atan(x)
double x;
double atan2(y,x)
double y,x;

DESCRIPTION
Sin, cos and tan return trigonometric functions of radian arguments x.
Asin returns the arc sine in the range \(-\pi/2\) to \(\pi/2\).
Acos returns the arc cosine in the range \(0\) to \(\pi\).
Atan returns the arc tangent in the range \(-\pi/2\) to \(\pi/2\).

On a VAX,
atan2(y,x) :=
    \begin{align*}
    \text{atan}(y/x) & \quad \text{if } x > 0, \\
    \text{sign}(y)*(-\pi - \text{atan}(|y/x|)) & \quad \text{if } x < 0, \\
    0 & \quad \text{if } x = y = 0, \text{ or } \\
    \text{sign}(y)*\pi/2 & \quad \text{if } x = 0 \neq y.
    \end{align*}

DIAGNOSTICS
On a VAX, if |x| > 1 then asin(x) and acos(x) will return reserved operands and errno will be set to EDOM.

NOTES
Atan2 defines atan2(0,0) = 0 on a VAX despite that previously atan2(0,0) may have generated an error message. 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,\theta) coordinates that must satisfy x = r*cos\theta and y = r*sin\theta. These equations are satisfied when (x=0,y=0) is mapped to (r=0,\theta=0) on a VAX. In general, conversions to polar coordinates should be computed thus:
    \begin{align*}
    r & := \text{hypot}(x,y); \quad \text{... := } \sqrt{x^2+y^2} \\
    \theta & := \text{atan2}(y,x).
    \end{align*}

3) The foregoing formulas need not be altered to cope in a reasonable way with signed zeros and infinities on a machine 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(\pm0,-0)
= ±π, for instance. In general the formulas above are equivalent to these:

\begin{align*}
r &:= \sqrt{(x^2 + y^2)}; \quad \text{if } r = 0 \text{ then } x := \text{copysign}(1,x); \\
\text{if } x > 0 &\text{ then } \theta := 2 \cdot \text{atan}(y/(r+x)) \\
\text{else } \theta &:= 2 \cdot \text{atan}((r-x)/y);
\end{align*}

except if \( r \) is infinite then \( \text{atan2} \) will yield an appropriate multiple of \( \pi/4 \) that would otherwise have to be obtained by taking limits.

**ERROR (due to Roundoff etc.)**

Let \( P \) stand for the number stored in the computer in place of \( \pi = 3.14159\, 26535\, 89793\, 23846\, 26433 \ldots \). Let "trig" stand for one of "sin", "cos" or "tan". Then the expression "trig(x)" in a program actually produces an approximation to \( \text{trig}(x \cdot \pi/P) \), and "atrig(x)" approximates \( (P/\pi) \cdot \text{atrig}(x) \). The approximations are close, within 0.9 \text{ ulps} for sin, cos and atan, within 2.2 \text{ ulps} for tan, asin, acos and atan2 on a VAX. Moreover, \( P = \pi \) in the codes that run on a VAX.

In the codes that run on other machines, \( P \) differs from \( \pi \) by a fraction of an \text{ 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 \( \pi \) explicitly is satisfied equally well regardless of whether \( P = \pi \). For instance, \( \sin^2(x) + \cos^2(x) = 1 \) and \( \sin(2x) = 2 \sin(x) \cos(x) \) to within a few \text{ ulps} no matter how big \( x \) may be. Therefore the difference between \( P \) and \( \pi \) is most unlikely to affect scientific and engineering computations.

**SEE ALSO**

math(3M), hypot(3M), sqrt(3M), infnan(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;

double cosh(x)
double x;

double tanh(x)
double x;

DESCRIPTION
These functions compute the designated hyperbolic functions for real arguments.

ERROR (due to Roundoff etc.)
Below 2.4 ulps; an ulp is one Unit in the Last Place.

DIAGNOSTICS
Sinh and cosh return the reserved operand on a VAX if the correct value would overflow.

SEE ALSO
math(3M), infnan(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
spray — scatter data in order to check the network

SYNOPSIS
#include <rpcsvc/spray.h>

RPC INFO
program number:
SPRAYPROG
xdr routines:
  xdr_sprayarr(xdrs, arr);
    XDR *xdrs;
    struct sprayarr *arr;
  xdr_spraycumul(xdrs, cumul);
    XDR *xdrs;
    struct spraycumul *cumul;

procs:
SPRAYPROC_SPA/Y
  Takes no arguments, returns no value.
  Increments a counter in server daemon.
  The server does not return this call, so the caller should have a timeout of 0.
SPRAYPROC_GET
  Takes no arguments, returns struct spraycumul with value of counter and clock.
SPRAYPROC_CLEAR
  Takes no arguments and returns no value.
  Zeros out counter and clock.

versions:
SPRAYVERS_ORIG

structures:
struct spraycumul {
  unsigned counter;
  struct timeval clock;
};
struct sprayarr {
  int *data,
  int nth
};

SEE ALSO
spray(8), sprayd(8)
This page intentionally left almost blank.
NAME
cbrt, sqrt – cube root, square root

SYNOPSIS
#include <math.h>
double cbrt(x)
double x;
double sqrt(x)
double x;

DESCRIPTION
Cbrt(x) returns the cube root of x.
Sqrt(x) returns the square root of x.

DIAGNOSTICS
On a VAX, sqrt(negative) returns the reserved operand and sets errno to EDOM.

ERROR (due to Roundoff etc.)
Cbrt is accurate to within 0.7 ulps.
Sqrt on a VAX is accurate to within 0.501 ulps.
Sqrt on a machine that conforms to IEEE 754 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), infnan(3M).

AUTHOR
W. Kahan
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 getc and putc(3S) handle characters quickly. The higher level routines gets, fgets, scanf, fscanf, fread, puts, fputs, printf, fprintf, fwrite all use getc 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 perilous: getc, getchar, putc, putchar, jeof, jeor, fput.

SEE ALSO
open(2), close(2), read(2), write(2), fread(3S), fseek(3S), fput(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 transparent, but may cause confusion or malfunctioning of programs which use standard i/o routines 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>

4th Berkeley Distribution May 13, 1986
### STDIO(3S)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fclose</td>
<td>close or flush a stream</td>
</tr>
<tr>
<td>fopen</td>
<td>open a stream</td>
</tr>
<tr>
<td>feof</td>
<td>stream status inquiries</td>
</tr>
<tr>
<td>feof</td>
<td>stream status inquiries</td>
</tr>
<tr>
<td>fclose</td>
<td>close or flush a stream</td>
</tr>
<tr>
<td>fgetc</td>
<td>get character or word from stream</td>
</tr>
<tr>
<td>fgets</td>
<td>get a string from a stream</td>
</tr>
<tr>
<td>fileno</td>
<td>stream status inquiries</td>
</tr>
<tr>
<td>fopen</td>
<td>open a stream</td>
</tr>
<tr>
<td>fprintf</td>
<td>formatted output conversion</td>
</tr>
<tr>
<td>putc</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>fread</td>
<td>buffered binary input/output</td>
</tr>
<tr>
<td>freopen</td>
<td>open a stream</td>
</tr>
<tr>
<td>fscanf</td>
<td>formatted input conversion</td>
</tr>
<tr>
<td>fseek</td>
<td>reposition a stream</td>
</tr>
<tr>
<td>ftell</td>
<td>reposition a stream</td>
</tr>
<tr>
<td>fwrite</td>
<td>buffered binary input/output</td>
</tr>
<tr>
<td>getc</td>
<td>get character or word from stream</td>
</tr>
<tr>
<td>getchar</td>
<td>get character or word from stream</td>
</tr>
<tr>
<td>gets</td>
<td>get a string from a stream</td>
</tr>
<tr>
<td>getw</td>
<td>get character or word from stream</td>
</tr>
<tr>
<td>printf</td>
<td>formatted output conversion</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>rewind</td>
<td>reposition a stream</td>
</tr>
<tr>
<td>scanf</td>
<td>formatted input conversion</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>setlinebuf</td>
<td>assign buffering to a stream</td>
</tr>
<tr>
<td>sprintf</td>
<td>formatted output conversion</td>
</tr>
<tr>
<td>sscanf</td>
<td>formatted input conversion</td>
</tr>
<tr>
<td>ungetc</td>
<td>push character back into input stream</td>
</tr>
</tbody>
</table>
NAME
strcat,strncat,strcmp,strncmp,strcpy,strncpy,strlen,index,rindex—string operations

SYNOPSIS

```c
#include <strings.h>
char *strcat(s1, s2)
char *s1, *s2;
char *strncat(s1, s2, n)
char *s1, *s2;
strcmp(s1, s2)
char *s1, *s2;
strncmp(s1, s2, n)
char *s1, *s2;
char *strcpy(s1, s2)
char *s1, *s2;
char *strncpy(s1, s2, n)
char *s1, *s2;
strlen(s)
char *s;
char *index(s, c)
char *s, c;
char *rindex(s, c)
char *s, c;
```

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

`Strcat` appends a copy of string `s2` to the end of string `s1`. `Strncat` copies at most `n` characters. Both return a pointer to the null-terminated result.

`Strcmp` compares its arguments and returns an integer greater than, equal to, or less than 0, according as `s1` is lexicographically greater than, equal to, or less than `s2`. `Strncmp` makes the same comparison but looks at at most `n` characters.

`Strcpy` copies string `s2` to `s1`, stopping after the null character has been moved. `Strncpy` copies exactly `n` characters, truncating or null-padding `s2`; the target may not be null-terminated if the length of `s2` is `n` or more. Both return `s1`.

`Strlen` returns the number of non-null characters in `s`.

`Index` (`rindex`) returns a pointer to the first (last) occurrence of character `c` in string `s`, or zero if `c` does not occur in the string.
NAME
stty, gtty – set and get terminal state (defunct)

SYNOPSIS
#include <sgtty.h>

stty(fd, buf)
int fd;
struct sgttyb *buf;

gtty(fd, buf)
int fd;
struct sgttyb *buf;

DESCRIPTION
This interface is obsoleted by ioctl(2).

Stty 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 stty 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 command 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
tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs – terminal independent operation routines

SYNOPSIS
cchar PC;
cchar *BC;
cchar *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, outc)
register char *cp;
int affcnt;
int (*outc)();

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 1024 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 dest­
line. 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 ter­
minals use control I for other functions, such as nondestructive space.) If a % sequence is
given which is not understood, then tgoto returns “OOPS”: 
Tputs decodes the leading padding information of the string cp; affcnt gives the number of lines affected by the operation, or 1 if this is not applicable, outc is a routine which is called with each character in turn. The external variable ospeed should contain the output speed of the terminal as encoded by stty(3). The external variable PC should contain a pad character to be used (from the pc capability) if a null ("@") is inappropriate.

FILES
/usr/lib/libtermcap.a -ltermcap library
/etc/termcap data base

SEE ALSO
ex(1), curses(3X), termcap(5)

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 Regents of the University of California.
 * All rights reserved. The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 * @(#)timeb.h6.2 (Berkeley) 6/8/85
 *
/*
 * Structure returned by ftime 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
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 Regents of the University of California.
 * All rights reserved. The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 * ...
 * @(#)times.h 6.2 (Berkeley) 6/8/85
 */

/*
 * 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)
TTYNAME(3) UNIX Programmer's Manual TTYNAME(3)

NAME

ttyname, isatty, ttyslot — find name of a terminal

SYNOPSIS

cchar *ttyname(filedes)
isatty(filedes)
ttyslot()

DESCRIPTION

TTYname 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).

Isatty 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

TTYname 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.

BUGS

The return value points to static data whose content is overwritten by each call.
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
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
  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 pre­
  vious 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.

BUGS
Vfree isn’t implemented.
NAME
varargs – variable argument list

SYNOPSIS
#include <varargs.h>

function(va_alist)
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_alist is used in a function header to declare a variable argument list.
va_dcl is a declaration for va_alist. 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>
execl(va_alist)
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);
}

BUGS
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, execl 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 \textit{va_start} and \textit{va_end} may be arbitrarily complex; for example, \textit{va_start} might contain an opening brace, which is closed by a matching brace in \textit{va_end}. Thus, they should only be used where they could be placed within a single complex statement.
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)

BUGS
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 */
    int vm_stime; /* system time */
    /* divide next two by utime+stime to get averages */
    unsigned vm_idsrss; /* 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_idsrss 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_idsrss would have the value
5*60, where vm_utime+vm_stime would be the 60. Vm_idsrss integrates data and stack seg-
mement 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
ypclnt yp_get_default_domain yp_bind yp_unbind yp_match yp_first yp_next yp_all
yp_order yp_master yperr_string ypprot_err — yellow pages client interface

SYNOPSIS
#include <rpcsvc/ypclnt.h>

yp_bind(indomain);
char *indomain;

void yp_unbind(indomain)
char *indomain;

yp_get_default_domain(outdomain);
char **outdomain;

yp_match(indomain, inmap, inkey, inkeylen, outval, outvallen)
char *indomain;
char *inmap;
char *inkey;
int inkeylen;
char **outval;
int *outvallen;

yp_first(indomain, inmap, outkey, outkeylen, outval, outvallen)
char *indomain;
char *inmap;
char **outkey;
int *outkeylen;
char **outval;
int *outvallen;

yp_next(indomain, inmap, inkey, inkeylen, outkey, outkeylen, outval, outvallen);
char *indomain;
char *inmap;
char *inkey;
int inkeylen;
char **outkey;
int *outkeylen;
char **outval;
int *outvallen;

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 ypfiles(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 processing; 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 \texttt{yp\_first()} function is called on a particular map, and then the \texttt{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.

\texttt{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 \texttt{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 \texttt{yp\_all} only when the transaction is completed (successfully or unsuccessfully), or your "\texttt{foreach}" function decides that it doesn't want to see any more key-value pairs.

The third parameter to \texttt{yp\_all} is

\begin{verbatim}
struct ypall_callback *incallback {
    int (*foreach)(
        int instatus,
        char *inkey,
        int inkeylen,
        char *inval,
        int invallen,
        char *indata);
}
\end{verbatim}

The function \texttt{foreach} is called

\begin{verbatim}
foreach(instatus, inkey, inkeylen, inval, invallen, indata);
\end{verbatim}

The \texttt{instatus} parameter will hold one of the return status values defined in \texttt{<rpcsvc/yp\_prot.h>} — either YP\_TRUE or an error code. (See \texttt{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 \texttt{inkey} and \texttt{inval} parameters is private to the \texttt{yp\_all} function, and is overwritten with the arrival of each new key-value pair. It is the responsibility of the \texttt{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 \texttt{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 \texttt{indata} parameter is the contents of the \texttt{incallback->data} element passed to \texttt{yp\_all}. The \texttt{data} element of the callback structure may be used to share state information between the \texttt{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

yppasswd — update user password in yellow pages

SYNOPSIS

#include <rpcsvc/yppasswd.h>

yppasswd(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:

YPPASSWDPROG

xdr routines:

    xdr_yppasswd(xdrs, yp)
        XDR *xdrs;
        struct yppasswd *yp;

    xdr_yppasswd(xdrs, pw)
        XDR *xdrs;
        struct passwd *pw;

procs:

YPPASSWDPROCUPDATE
    Takes struct yppasswd as argument, returns integer.
    Same behavior as yppasswd() wrapper.
    Uses UNIX authentication.

versions:

YPPASSWDVERS_ORIG

structures:

    struct yppasswd {
        char *oldpass; /* old (unencrypted) password */
        struct passwd newpw; /* new pw structure */
    };

SEE ALSO

yppasswd(1), yppasswdd(8C)
NAME
intro – introduction to FORTRAN library functions

DESCRIPTION
This section describes those functions that are in the Fortran run time library. The functions
listed here provide an interface from f77 programs to the system in the same manner as the C
library does for C programs. They are automatically loaded as needed by the Fortran com­
piler f77(1), except for the graphics interface routines. Those must be explicitly requested, see
plot(3f).

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 may be referenced using the generic names listed above, or
they may 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), zsqr(zz), or cdsqr(zz)
compute the square root of zz. The dcmplx intrinsic forms a double complex value from two
double precision variables or expressions, and the name of the specific function for the conjuga­
tate of a double complex value is dconjg.

Most of these functions are in libU77.a. Some are in libF77.a or libI77.a. A few intrinsic
functions are described for the sake of completeness.

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>and</td>
<td>bit.3f</td>
<td>bitwise and</td>
</tr>
<tr>
<td>arc</td>
<td>plot.3f</td>
<td>f77 interface to plot(3x)</td>
</tr>
<tr>
<td>bessel</td>
<td>bessel.3f</td>
<td>bessel functions of two kinds for integer orders</td>
</tr>
<tr>
<td>box</td>
<td>plot.3f</td>
<td>f77 interface to plot(3x)</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>circle</td>
<td>plot.3f</td>
<td>f77 interface to plot(3x)</td>
</tr>
<tr>
<td>clospl</td>
<td>plot.3f</td>
<td>f77 interface to plot(3x)</td>
</tr>
<tr>
<td>cont</td>
<td>plot.3f</td>
<td>f77 interface to plot(3x)</td>
</tr>
<tr>
<td>ctime</td>
<td>time.3f</td>
<td>return system time</td>
</tr>
<tr>
<td>dffrac</td>
<td>flmin.3f</td>
<td>return extreme values</td>
</tr>
<tr>
<td>dffmax</td>
<td>flmin.3f</td>
<td>return extreme values</td>
</tr>
<tr>
<td>dffmin</td>
<td>flmin.3f</td>
<td>return extreme values</td>
</tr>
<tr>
<td>drand</td>
<td>rand.3f</td>
<td>return random values</td>
</tr>
<tr>
<td>drandm</td>
<td>random.3f</td>
<td>better random number generator</td>
</tr>
<tr>
<td>dtime</td>
<td>etime.3f</td>
<td>return elapsed execution time</td>
</tr>
<tr>
<td>erase</td>
<td>plot.3f</td>
<td>f77 interface to plot(3x)</td>
</tr>
<tr>
<td>etime</td>
<td>etime.3f</td>
<td>return elapsed execution time</td>
</tr>
<tr>
<td>exit</td>
<td>exit.3f</td>
<td>terminate process with status</td>
</tr>
<tr>
<td>falloc</td>
<td>malloc.3f</td>
<td>memory allocator</td>
</tr>
<tr>
<td>fdate</td>
<td>fdate.3f</td>
<td>return date and time in an ASCII string</td>
</tr>
<tr>
<td>ffrac</td>
<td>flmin.3f</td>
<td>return extreme values</td>
</tr>
</tbody>
</table>
fgetc  getc.3f  get a character from a logical unit
fmax  flmin.3f  return extreme values
fmin  flmin.3f  return extreme values
flush  flush.3f  flush output to a logical unit
fork  fork.3f  create a copy of this process
fpecnt  trpfpe.3f  trap and repair floating point faults
fputc  putc.3f  write a character to a fortran logical unit
free  malloc.3f  memory allocator
fseek  fseek.3f  reposition a file on a logical unit
fstat  stat.3f  get file status
ftell  fseek.3f  reposition a file on a logical unit
gerror  perror.3f  get system error messages
getarg  getarg.3f  return command line arguments
getc  getc.3f  get a character from a logical unit
getcwd  getcwd.3f  get pathname of current working directory
getenv  getenv.3f  get value of environment variables
gerror  perror.3f  get system error messages
getgid  getuid.3f  get user or group ID of the caller
getlog  getlogin.3f  get user's login name
getpid  getpid.3f  get process id
getuid  getuid.3f  get user or group ID of the caller
gmtime  time.3f  return system time
hostnm  hostnm.3f  get name of current host
iargc  getarg.3f  return command line arguments
idate  idate.3f  return date or time in numerical form
ierrno  perror.3f  get system error messages
index  index.3f  tell about character objects
inmax  flmin.3f  return extreme values
ioinit  ioinit.3f  change f77 1/O initialization
irand  rand.3f  return random values
irandom  random.3f  better random number generator
isatty  ttynam.3f  find name of a terminal port
it ime  idate.3f  return date or time in numerical form
kill  kill.3f  send a signal to a process
label  plot.3f  f77 interface to plot(3x)
len  index.3f  tell about character objects
line  plot.3f  f77 interface to plot(3x)
linemd  plot.3f  f77 interface to plot(3x)
link  link.3f  make a link to an existing file
lnblink  index.3f  tell about character objects
loc  loc.3f  return the address of an object
long  long.3f  integer object conversion
lshift  bit.3f  left shift
lstat  stat.3f  get file status
ltime  time.3f  return system time
malloc  malloc.3f  memory allocator
move  plot.3f  f77 interface to plot(3x)
not  bit.3f  bitwise complement
openpl  plot.3f  f77 interface to plot(3x)
or  bit.3f  bitwise or
perror  perror.3f  get system error messages
point  plot.3f  f77 interface to plot(3x)
putc  putc.3f  write a character to a fortran logical unit
qsort  qsort.3f  quick sort
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rand</td>
<td>return random values</td>
</tr>
<tr>
<td>random</td>
<td>better random number generator</td>
</tr>
<tr>
<td>rename</td>
<td>rename a file</td>
</tr>
<tr>
<td>rindex</td>
<td>tell about character objects</td>
</tr>
<tr>
<td>rshift</td>
<td>right shift</td>
</tr>
<tr>
<td>short</td>
<td>integer object conversion</td>
</tr>
<tr>
<td>signal</td>
<td>change the action for a signal</td>
</tr>
<tr>
<td>sleep</td>
<td>suspend execution for an interval</td>
</tr>
<tr>
<td>space</td>
<td>f77 interface to plot(3x)</td>
</tr>
<tr>
<td>stat</td>
<td>get file status</td>
</tr>
<tr>
<td>symlink</td>
<td>make a symbolic link</td>
</tr>
<tr>
<td>system</td>
<td>execute a UNIX command</td>
</tr>
<tr>
<td>tclose</td>
<td>f77 tape I/O</td>
</tr>
<tr>
<td>time</td>
<td>return system time</td>
</tr>
<tr>
<td>topen</td>
<td>f77 tape I/O</td>
</tr>
<tr>
<td>traper</td>
<td>trap arithmetic errors</td>
</tr>
<tr>
<td>trapov</td>
<td>trap and repair floating point overflow</td>
</tr>
<tr>
<td>tread</td>
<td>f77 tape I/O</td>
</tr>
<tr>
<td>trewin</td>
<td>f77 tape I/O</td>
</tr>
<tr>
<td>trfpfe</td>
<td>trap and repair floating point faults</td>
</tr>
<tr>
<td>tskipf</td>
<td>f77 tape I/O</td>
</tr>
<tr>
<td>tstate</td>
<td>f77 tape I/O</td>
</tr>
<tr>
<td>ttynam</td>
<td>find name of a terminal port</td>
</tr>
<tr>
<td>twrite</td>
<td>f77 tape I/O</td>
</tr>
<tr>
<td>unlink</td>
<td>remove a directory entry</td>
</tr>
<tr>
<td>wait</td>
<td>wait for a process to terminate</td>
</tr>
<tr>
<td>xor</td>
<td>bitwise exclusive or</td>
</tr>
</tbody>
</table>
NAME
abort - abnormal termination

SYNOPSIS
subroutine abort (string)
character*(*) string

DESCRIPTION
Abort cleans up the I/O buffers and then terminates execution. If string is given, it is written to logical unit 0 preceded by "abort:"

If the -g flag was specified during loading, then execution is terminated by calling abort (3) which aborts producing a core file in the current directory. If -g was not specified while loading, then *** Execution terminated is written on logical unit 0 and execution is terminated.

If the f77_dump_flag environment variable has been set to a value which begins with y, abort (3) is called whether or not -g was specified during loading. Similarly, if the value of f77_dump_flag begins with n, abort is not called.

FILES
/usr/lib/libF77.a

SEE ALSO
abort(3)

BUGS
String is ignored on the PDP11.
NAME
access – determine accessibility of a file

SYNOPSIS
integer function access (name, mode)
character(*) 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)

BUGS
Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>. 

ALARM(3F)

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
bessel functions – of two kinds for integer orders

SYNOPSIS
function besj0 (x)
function besj1 (x)
function besjn (n, x)
function besy0 (x)
function besy1 (x)
function besyn (n, x)
double precision function dbesj0 (x)
double precision x
double precision function dbesj1 (x)
double precision x
double precision function dbesjn (n, x)
double precision x
double precision function dbesy0 (x)
double precision x
double precision function dbesy1 (x)
double precision x
double precision function dbesyn (n, x)
double precision x

DESCRIPTION
These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS
Negative arguments cause besy0, besy1, and besyn to return a huge negative value. The system error code will be set to EDOM (33).

FILES
/usr/lib/libF77.a

SEE ALSO
j0(3M), perror(3F)
NAME
bit – and, or, xor, not, rshift, lshift bitwise functions

SYNOPSIS
(intrinsic) function and (word1, word2)
(intrinsic) function or (word1, word2)
(intrinsic) function xor (word1, word2)
(intrinsic) function not (word)
(intrinsic) function rshift (word, nbits)
(intrinsic) function lshift (word, nbits)

DESCRIPTION
These bitwise functions are built into the compiler and return the data type of their argument(s). Their arguments must be integer or logical values.

The bitwise combinatorial functions return the bitwise “and” (and), “or” (or), or “exclusive or” (xor) of two operands. Not returns the bitwise complement of its operand.

Lshift, or rshift with a negative nbits, is a logical left shift with no end around carry. Rshift, or lshift with a negative nbits, is an arithmetic right shift with sign extension. No test is made for a reasonable value of nbits.

These functions may be used to create a variety of general routines, as in the following statement function definitions:

    integer bitset, bitclr, getbit, word, bitnum
    bitset( word, bitnum ) = or( word, lshift(1, bitnum) )
    bitclr( word, bitnum ) = and( word, not( lshift(1, bitnum) ) )
    getbit( word, bitnum ) = and( rshift( word, bitnum ), 1 )

FILES
These functions are generated in-line by the f77 compiler.
NAME
    chdir – change default directory

SYNOPSIS
    integer function chdir (dirname)
    character(*) 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
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 ele-
ment. 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
exit – terminate process with status

SYNOPSIS
subroutine exit (status)
integer status

DESCRIPTION
Exit flushes and closes all the process's files, and notifies the parent process if it is executing a wait. The low-order 8 bits of status are available to the parent process. (Therefore status should be in the range 0 – 255)

This call will never return.

The C function exit may cause cleanup actions before the final 'sys exit'.

FILES
/usr/lib/libF77.a

SEE ALSO
exit(2), fork(2), fork(3F), wait(2), wait(3F)
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
cotime(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
    ctime(3), time(3F), itime(3F), idate(3F), ltime(3F)
NAME
flmin, flmax, ffrac, dflmin, dflmax, dffrac, inmax – return extreme values

SYNOPSIS
function flmin()
function flmax()
function ffrac()
double precision function dflmin()
double precision function dflmax()
double precision function dffrac()
function inmax()

DESCRIPTION
Functions flmin and flmax return the minimum and maximum positive floating point values respectively. Functions dflmin and dflmax return the minimum and maximum positive double precision floating point values. Function inmax returns the maximum positive integer value.

The functions ffrac and dffrac return the fractional accuracy of single and double precision floating point numbers respectively. This is the difference between 1.0 and the smallest real number greater than 1.0.

These functions can be used by programs that must scale algorithms to the numerical range of the processor.

FILES
/usr/lib/libF77.a
NAME
flush – flush output to a logical unit

SYNOPSIS
subroutine flush (lunit)

DESCRIPTION
Flush causes the contents of the buffer for logical unit \textit{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
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 indicates 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
getarg, iargc – return command line arguments

SYNOPSIS
subroutine getarg (k, arg)
character(*) 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

type 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
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
  getenv - get value of environment variables

SYNOPSIS
  subroutine getenv (ename, evalue)
  character(*) ename, evalue

DESCRIPTION
  getenv searches the environment list (see environ(7)) for a string of the form
  ename=value and returns value in evalue if such a string is present, otherwise fills
evalue with blanks.

FILES
  /usr/lib/libU77.a

SEE ALSO
  environ(7), execve(2)
NAME
getlog – get user's login name

SYNOPSIS
subroutine getlog (name)
character(*) name

character(*) 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
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
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
  hostnm - get name of current host

SYNOPSIS
  integer function hostnm (name)
  character(*) name

DESCRIPTION
  This function puts the name of the current host into character string name. The return value
  should be 0; any other value indicates an error.

FILES
  /usr/lib/libU77.a

SEE ALSO
  gethostname(2)
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

index, rindex, lnblnk, len – tell about character objects

SYNOPSIS

(intrinsic) function index (string, substr)
character*(*) string, substr

integer function rindex (string, substr)
character*(*) string, substr

function lnblnk (string)
character*(*) string

(intrinsic) function len (string)
character*(*) string

DESCRIPTION

Index (rindex) returns the index of the first (last) occurrence of the substring substr in string, or zero if it does not occur. Index is an f77 intrinsic function; rindex is a library routine.

Lnblnk returns the index of the last non-blank character in string. This is useful since all f77 character objects are fixed length, blank padded. Intrinsic function len returns the size of the character object argument.

FILES

/usr/lib/libF77.a
NAME
  ioinit – change f77 I/O initialization

SYNOPSIS
  logical function ioinit (cctl, bzero, apnd, prefix, verbose)
  logical cctl, bzero, apnd, verbose
  character(*) prefix

DESCRIPTION
  This routine will initialize several global parameters in the f77 I/O system, and attach externally defined files to logical units at run time. The effect of the flag arguments applies to logical units opened after ioinit is called. The exception is the preassigned units, 5 and 6, to which cctl and bzero will apply at any time. ioinit is written in Fortran-77.

  By default, carriage control is not recognized on any logical unit. If cctl is .true. then carriage control will be recognized on formatted output to all logical units except unit 0, the diagnostic channel. Otherwise the default will be restored.

  By default, trailing and embedded blanks in input data fields are ignored. If bzero is .true. then such blanks will be treated as zeros. Otherwise the default will be restored.

  By default, all files opened for sequential access are positioned at their beginning. It is sometimes necessary or convenient to open at the END-OF-FILE so that a write will append to the existing data. If apnd is .true. then files opened subsequently on any logical unit will be positioned at their end upon opening. A value of .false. will restore the default behavior.

  ioinit may be used to associate file names with Fortran logical unit numbers through environment variables (see "Introduction to the f77 I/O Library" for a more general way of doing this). If the argument prefix is a non-blank string, then names of the form prefixNN will be sought in the program environment. The value associated with each such name found will be used to open logical unit NN for formatted sequential access. For example, if f77 program myprogram included the call

  call ioinit (.true., .false., .false., 'FORT', .false.)

  then when the following sequence

  % setenv FORT01 mydata
  % setenv FORT12 myresults
  % myprogram

  would result in logical unit 1 opened to file mydata and logical unit 12 opened to file myresults. Both files would be positioned at their beginning. Any formatted output would have column 1 removed and interpreted as carriage control. Embedded and trailing blanks would be ignored on input.

  If the argument verbose is .true. then ioinit will report on its activity.

  The effect of

  call ioinit (.true., .true., .false., '', .false.)

  can be achieved without the actual call by including "-IF66" on the f77 command line. This gives carriage control on all logical units except 0, causes files to be opened at their beginning, and causes blanks to be interpreted as zero's.

  The internal flags are stored in a labeled common block with the following definition:

  integer*2 ieof, ictl, ibzer
common /ioiflg/ ieof, ictl, ibzr

FILES
/usr/lib/lib177.a f77 I/O library
/usr/lib/lib166.a sets older fortran I/O modes

SEE ALSO
getarg(3F), getenv(3F), “Introduction to the f77 I/O Library”

BUGS
Prefix can be no longer than 30 characters. A pathname associated with an environment name can be no longer than 255 characters.
The “+” carriage control does not work.
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
    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
long, short – integer object conversion

SYNOPSIS
integer*4 function long (int2)
integer*2 int2

integer*2 function short (int4)
integer*4 int4

DESCRIPTION
These functions provide conversion between short and long integer objects. Long is useful when constants are used in calls to library routines and the code is to be compiled with "-i2". Short is useful in similar context when an otherwise long object must be passed as a short integer.

FILES
/usr/lib/libF77.a
NAME
malloc, free, falloc – memory allocator

SYNOPSIS

subroutine malloc (size, addr)
  integer size, addr

subroutine free (addr)
  integer addr

subroutine falloc (nelem, elsize, clean, basevec, addr, offset)
  integer nelem, elsize, clean, addr, offset

DESCRIPTION

Malloc, falloc and free provide a general-purpose memory allocation package. Malloc returns
in addr the address of a block of at least size bytes beginning on an even-byte boundary.

Falloc allocates space for an array of nelem elements of size elsize and returns the address of
the block in addr. It zeros the block if clean is 1. It returns in offset an index such that the
storage may be addressed as basevec(offset+1) ... basevec(offset+nelem). Falloc gets extra bytes
so that after address arithmetic, all the objects so addressed are within the block.

The argument to free is the address of a block previously allocated by malloc or falloc; this
space is made available for further allocation, but its contents are left undisturbed. To free
blocks allocated by falloc, use addr in calls to free, do not use basevec(offset+1).

Needless to say, grave disorder will result if the space assigned by malloc or falloc is overrun or
if some random number is handed to free.

DIAGNOSTICS

Malloc and falloc set addr to 0 if there is no available memory or if the arena has been detect-
ably corrupted by storing outside the bounds of a block.

The following example shows how to obtain memory and use it within a subprogram:

    integer addr, work(1), offset
    ...
    call falloc ( n, 4, 0, work, addr, offset )
    do 10 i = 1, n
       work(offset+i) = ...
    10 continue

The next example reads in dimension information, allocates space for two arrays and two vec-
tors, and calls subroutine doit to do the computations:

    integer addr, dummy(1), offs
    read *, k, l, m
    indm1 = l
    indm2 = indm1 + k*l
    indm3 = indm2 + l*m
    indsym = indm3 + k*m
    lsym = n*(n+1)/2
    indv = indsym + lsym
    indtot = indv + m
    call falloc ( indtot, 4, 0, dummy, addr, offs )
    call doit(dummy(indm1+offs), dummy(indm2+offs),
              dummy(indm3+offs), dummy(indsym+offs),
              dummy(indv+offs), m, n, lsym )
    end
subroutine doit( arr1, arr2, arr3, vsym, vec, m, n, lsym )
real arr1(k,l), arr2(l,m), arr3(k,m), vsym(lsym), v2(m)
...

FILES
/usr/lib/libU77.a

SEE ALSO
malloc(3)
NAME
perror, gerror, ierrno - get system error messages

SYNOPSIS
subroutine perror (string)
character(*) string

subroutine gerror (string)
character(*) string

character(*) function gerror()

function 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 et al. - f77 library interface to plot (3X) libraries.

SYNOPSIS
subroutine openpl()
subroutine erase()
subroutine label(str)
   character str(*)
subroutine line(ix1, iy1, ix2, iy2)
subroutine box(ix1, iy1, ix2, iy2)
   Draw a rectangle and leave the cursor at (ix2,iy2).
subroutine circle(ix, iy, ir)
subroutine arc(ix, iy, ix0, iy0, ix1, iy1)
subroutine move(ix, iy)
subroutine cont(ix, iy)
subroutine point(ix, iy)
subroutine linemd(str)
   character str(*)
subroutine space(ix0, iy0, ix1, iy1)
subroutine clospl()

DESCRIPTION
These are interface subroutines in the library -lf77plot, allowing f77 users to call the plot(3X) graphics routines which generate graphic output in a relatively device-independent manner. The f77 subroutine names are the same as the C function names except that linemod and closepl have been shortened to linemd and clospl. See plot(5) and plot(3X) for a description of their effect.

Only the first 255 character in string arguments to label and linemd are used.

This library must be specified in the f77(1) command before the device specific graphics library; for example, to compile and load a FORTRAN program in prog.f to run on a Tektronix 4014 terminal:

    f77 prog.f -lf77plot -l4014

See plot(3X) for a complete list of device specific plotting libraries.

SEE ALSO
plot(5), plot(1G), plot(3X), graph(1G)
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
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*2 function that will determine the sorting
order. 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, drand, irand – return random values

SYNOPSIS
function irand (iflag)

function rand (iflag)

double precision function drand (iflag)

DESCRIPTION
The newer random(3f) should be used in new applications; rand remains for compatibility.
These functions use rand(3C) to generate sequences of random numbers. If iflag is '1', the generator is restarted and the first random value is returned. If iflag is otherwise non-zero, it is used as a new seed for the random number generator, and the first new random value is returned.
irand returns positive integers in the range 0 through 2147483647. Rand and drand return values in the range 0 through 1.0.

FILES
/usr/lib/libF77.a

SEE ALSO
random(3F), rand(3C)

BUGS
The algorithm returns a 15 bit quantity on the PDP11; a 31 bit quantity on the VAX. Irand on the PDP11 calls rand(3C) twice to form a 31 bit quantity, but bit 15 will always be 0.
NAME
random, drandm, irandm – better random number generator

SYNOPSIS

function irandm (iflag)

function random (iflag)

double precision function drandm (iflag)

DESCRIPTION
These functions use random(3) to generate sequences of random numbers, and should be used rather than the older functions described in man 3f rand. If iflag is non-zero, it is used as a new seed for the random number generator, and the first new random value is returned.

Irandm returns positive integers in the range 0 through 2147483647 (2
•\n31-1). Random and drandm return values in the range 0. through 1.0 by dividing the integer random number from random(3) by 2147483647.

FILES
/ust/lib/libF77.a

SEE ALSO
random(3)
NAME
rename – rename a file

SYNOPSIS
integer function rename (from, to)
character(*) from, to

DESCRIPTION
From must be the pathname of an existing file. To will become the new pathname for the file. If to exists, then both from and to must be the same type of file, and must reside on the same filesystem. If to exists, it will be removed first.

The returned value will be 0 if successful; a system error code otherwise.

FILES
/usr/lib/libU77.a

SEE ALSO
rename(2), perror(3F)

BUGS
Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.
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
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
stat, lstat, fstat - get file status

SYNOPSIS

integer function stat (name, statb)
character(*) name
integer statb(12)

integer function lstat (name, statb)
character(*) name
integer statb(12)

integer function fstat (lunit, statb)
integer statb(12)

DESCRIPTION
These routines return detailed information about a file. Stat and lstat return information
about file name; 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
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
command. If environment variable SHELL is found, its value will be used as the command
interpreter (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 wait(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
    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
topen, tclose, tread, twrite, trewin, tskipf, tstate – f77 tape I/O

SYNOPSIS

integer function topen (tlu, devnam, label)
integer tlu
character(*) devnam
logical label

integer function tclose (tlu)
integer tlu

integer function tread (tlu, buffer)
integer tlu
character(*) buffer

integer function twrite (tlu, buffer)
integer tlu
character(*) buffer

integer function trewin (tlu)
integer tlu

integer function tskipf (tlu, nfiles, nrecs)
integer tlu, nfiles, nrecs

integer function tstate (tlu, fileno, recno, errf, eoff, eotf, tcsr)
integer tlu, fileno, recno, tcsr
logical errf, eoff, eotf

DESCRIPTION

These functions provide a simple interface between f77 and magnetic tape devices. A "tape logical unit", tlu, is "topen"ed in much the same way as a normal f77 logical unit is "open"ed. All other operations are performed via the tlu. The tlu has no relationship at all to any normal f77 logical unit.

Topen associates a device name with a tlu. Tlu must be in the range 0 to 3. The logical argument label should indicate whether the tape includes a tape label. This is used by trewin below. Topen does not move the tape. The normal returned value is 0. If the value of the function is negative, an error has occurred. See perror(3F) for details.

Tclose closes the tape device channel and removes its association with tlu. The normal returned value is 0. A negative value indicates an error.

Tread reads the next physical record from tape to buffer. Buffer must be of type character. The size of buffer should be large enough to hold the largest physical record to be read. The actual number of bytes read will be returned as the value of the function. If the value is 0, the end-of-file has been detected. A negative value indicates an error.

Twrite writes a physical record to tape from buffer. The physical record length will be the size of buffer. Buffer must be of type character. The number of bytes written will be returned. A value of 0 or negative indicates an error.

Trewin rewinds the tape associated with tlu to the beginning of the first data file. If the tape is a labelled tape (see topen above) then the label is skipped over after rewinding. The normal returned value is 0. A negative value indicates an error.
Tskipf allows the user to skip over files and/or records. First, nfiles end-of-file marks are skipped. If the current file is at EOF, this counts as 1 file to skip. (Note: This is the way to reset the EOF status for a tlu.) Next, nrecs physical records are skipped over. The normal returned value is 0. A negative value indicates an error.

Finally, tstate allows the user to determine the logical state of the tape I/O channel and to see the tape drive control status register. The values of fileno and recono will be returned and indicate the current file and record number. The logical values errf, eoff, and eotf indicate an error has occurred, the current file is at EOF, or the tape has reached logical end-of-tape. End-of-tape (EOT) is indicated by an empty file, often referred to as a double EOF mark. It is not allowed to read past EOT although it is allowed to write. The value of tcsr will reflect the tape drive control status register. See ht(4) for details.

FILES
/usr/lib/libU77.a

SEE ALSO
ht(4), perror(3F), rewind(1)
NAME
taper – trap arithmetic errors

SYNOPSIS
    integer function traper (mask)

DESCRIPTION
    NOTE: This routine applies only to the $vax$. It is ignored on the PDP11.
    Integer overflow and floating point underflow are not normally trapped during execution. This
    routine enables these traps by setting status bits in the process status word. These bits are
    reset on entry to a subprogram, and the previous state is restored on return. Therefore, this
    routine must be called inside each subprogram in which these conditions should be trapped.
    If the condition occurs and trapping is enabled, signal SIGFPE is sent to the process. (See
    signal(3C))

    The argument has the following meaning:

    
    value  meaning
    0      do not trap either condition
    1      trap integer overflow only
    2      trap floating underflow only
    3      trap both the above

    The previous value of these bits is returned.

FILES
    /usr/lib/libF77.a

SEE ALSO
    signal(3C), signal(3F)
NAME

trapov - trap and repair floating point overflow

SYNOPSIS

subroutine trapov (numesg, rtnval)
    double precision rtnval

DESCRIPTION

NOTE: This routine applies only to the older VAX 11/780's. VAX computers made or upgraded since spring 1983 handle errors differently. See trpfope(3F) for the newer error handler. This routine has always been ineffective on the VAX 11/750. It is a null routine on the PDP11.

This call sets up signal handlers to trap arithmetic exceptions and the use of illegal operands. Trapping arithmetic exceptions allows the user's program to proceed from instances of floating point overflow or divide by zero. The result of such operations will be an illegal floating point value. The subsequent use of the illegal operand will be trapped and the operand replaced by the specified value.

The first numesg occurrences of a floating point arithmetic error will cause a message to be written to the standard error file. If the resulting value is used, the value given for rtnval will replace the illegal operand generated by the arithmetic error. Rtnval must be a double precision value. For example, "0d0" or "dflmax()".

FILES

/usr/lib/libF77.a

SEE ALSO

trpfope(3F), signal(3F), range(3F)

BUGS

Other arithmetic exceptions can be trapped but not repaired.

There is no way to distinguish between an integer value of 32768 and the illegal floating point form. Therefore such an integer value may get replaced while repairing the use of an illegal operand.
NAME
trpfpe, fpecnt – trap and repair floating point faults

SYNOPSIS
subroutine trpfpe (numesg, rtnval)
double precision rtnval

integer function fpecnt ()

common /fpeflt/ fperr
logical fperr

DESCRIPTION
NOTE: This routine applies only to Vax computers. It is a null routine on the PDP11.

Trpfpe sets up a signal handler to trap arithmetic exceptions. If the exception is due to a
floating point arithmetic fault, the result of the operation is replaced with the rtnval specified.
Rtnval must be a double precision value. For example, “Od0” or “dfi max()”.
The first numesg occurrences of a floating point arithmetic error will cause a message to be
written to the standard error file. Any exception that can't be repaired will result in the
default action, typically an abort with core image.

Fpecnt returns the number of faults since the last call to trpfpe.
The logical value in the common block labelled fpeflt will be set to .true. each time a fault
occurs.

FILES
/usr/lib/libF77.a

SEE ALSO
signal(3F), range(3F)

BUGS
This routine works only for faults, not traps. This is primarily due to the Vax architecture.
If the operation involves changing the stack pointer, it can't be repaired. This seldom should
be a problem with the f77 compiler, but such an operation might be produced by the optim­
izer.
The POLY and EMOD opcodes are not dealt with.
NAME
tynam, isatty – find name of a terminal port

SYNOPSIS
character迸function ttynam (lunit)

   logical function isatty (lunit)

DESCRIPTION
Ttynam 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
Ttynam returns an empty string (all blanks) if lunit is not associated with a terminal device in
directory ‘/dev’.
NAME
unlink – remove a directory entry

SYNOPSIS
integer function unlink (name)
character(*) 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
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
intro – introduction to special files and hardware support

DESCRIPTION
This section describes the special files, related driver functions, and networking support available in the system. In this part of the manual, the SYNOPIS section of each configurable device gives a sample specification for use in constructing a system description for the config(8) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log /usr/adm/messages due to errors in device operation; see syslogd(8) for more information.

This section contains both devices which may be configured into the system, “4” entries, and network related information, “4N”, “4P”, and “4F” entries; The networking support is introduced in intro(4N).

VAX DEVICE SUPPORT
This section describes the hardware supported on the DEC VAX-11. Software support for these devices comes in two forms. A hardware device may be supported with a character or block device driver, or it may be used within the networking subsystem and have a network interface driver. Block and character devices are accessed through files in the file system of a special type; c.f. mknod(8). Network interfaces are indirectly accessed through the interprocess communication facilities provided by the system; see socket(2).

A hardware device is identified to the system at configuration time and the appropriate device or network interface driver is then compiled into the system. When the resultant system is booted, the autoconfiguration facilities in the system probe for the device on either the UNIBUS (or Q-bus) or MASSBUS and, if found, enable the software support for it. If a UNIBUS device does not respond at autoconfiguration time it is not accessible at any time afterwards. To enable a UNIBUS device which did not autoconfigure, the system will have to be rebooted. If a MASSBUS device comes “on-line” after the autoconfiguration sequence it will be dynamically autoconfigured into the running system.

The autoconfiguration system is described in autoconf(4). A list of the supported devices is given below.

SEE ALSO
intro(4), intro(4N), autoconf(4), config(8).

Building 4.3BSD UNIX Systems with Config (SMM:2)

LIST OF DEVICES
The devices listed below are supported in this incarnation of the system. Pseudo-devices are not listed. Devices are indicated by their functional interface. If second vendor products provide functionally identical interfaces they should be usable with the supplied software. (Beware, however, that we promise the software works ONLY with the hardware indicated on the appropriate manual page.) Occasionally, new devices of a similar type may be added simply by creating appropriate table entries in the driver.

acc  ACC LH/DH IMP communications interface
ad   Data translation A/D interface
css  DEC IMP-11A communications interface
crl  VAX 8600, 8650 console RL02 disk
cr   C/A/T or APS phototypesetter
ddn  ACC ACP625 DDN Standard Mode X.25 IMP interface
de  DEC DEUNA 10Mb/s Ethernet controller
dh   DH-11 emulators, terminal multiplexor
dhu  DHU-11 terminal multiplexor
dmc  DEC DMC-11/DMR-11 point-to-point communications device
DMF DEC DMF-32 terminal multiplexor and parallel printer interface
DMZ DEC DMZ-32 terminal multiplexor
DN DEC DN-11 autodialer interface
DZ DZ-11 terminal multiplexor
EC 3Com 10Mb/s Ethernet controller
EN Xerox 3Mb/s Ethernet controller (obsolete)
EX Excelan 10Mb/s Ethernet controller
FL VAX-11/780 console floppy interface
HDH ACC IF-11/HDH IMP interface
HK RK5-11/RK06 and RK07 moving head disk
HP MASSBUS disk interface (with RP06, RM03, RM05, etc.)
HT TM03 MASSBUS tape drive interface (with TE-16, TU-45, TU-77)
HY DR-11B or G1-13 interface to an NSC Hyperchannel
IK Ikonas frame buffer graphics device interface
IL Interlan 1010, 1010A 10Mb/s Ethernet controller
IX Interlan NP-100 10Mb/s Ethernet controller
KG KL-11/DL-11W line clock
LP LP-11 parallel line printer interface
MT TM78 MASSBUS tape drive interface
NP Interlan NP-100 10Mb/s Ethernet controller (intelligent mode)
PCL DEC PCL-11 communications interface
PS Evans and Sutherland Picture System 2 graphics interface
QE DEC DEQNA Q-bus 10Mb/s Ethernet interface
RX DEC RX02 floppy interface
TM TM-11/TE-10 tape drive interface
TMSCP TMSCP-compatible tape controllers (e.g., TU81, TK50)
TS TS-11 tape drive interface
TU VAX-11/730 TU58 console cassette interface
UDA DEC UDA-50 disk controller
UN DR-11W interface to Ungermann-Bass
UP Emulex SC-21V, SC-31 UNIBUS disk controller
UT UNIBUS TU-45 tape drive interface
UU TU58 dual cassette drive interface (DL11)
VA Benson-Varian printer/plotter interface
VP Versatec printer/plotter interface
VV Proteon proNET 10Mb/s and 80Mb/s ring network interface
NAME
networking – introduction to networking facilities

SYNOPSIS
#include <sys/socket.h>
#include <net/route.h>
#include <net/if.h>

DESCRIPTION
This section briefly describes the networking facilities available in the system. Documentation in this part of section 4 is broken up into three areas: protocol families (domains), protocols, and network interfaces. Entries describing a protocol family are marked “4F,” while entries describing protocol use are marked “4P.” Hardware support for network interfaces are found among the standard “4” entries.

All network protocols are associated with a specific protocol family. A protocol family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. A protocol family may support multiple methods of addressing, though the current protocol implementations do not. A protocol family is normally comprised of a number of protocols, one per socket(2) type. It is not required that a protocol family support all socket types. A protocol family may contain multiple protocols supporting the same socket abstraction.

A protocol supports one of the socket abstractions detailed in socket(2). A specific protocol may be accessed either by creating a socket of the appropriate type and protocol family, or by requesting the protocol explicitly when creating a socket. Protocols normally accept only one type of address format, usually determined by the addressing structure inherent in the design of the protocol family/network architecture. Certain semantics of the basic socket abstractions are protocol specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide non-standard facilities or extensions to a mechanism. For example, a protocol supporting the SOCK_STREAM abstraction may allow more than one byte of out-of-band data to be transmitted per out-of-band message.

A network interface is similar to a device interface. Network interfaces comprise the lowest layer of the networking subsystem, interacting with the actual transport hardware. An interface may support one or more protocol families and/or address formats. The SYNOPSIS section of each network interface entry gives a sample specification of the related drivers for use in providing a system description to the config(8) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log, /usr/adm/messages (see syslogd(8)), due to errors in device operation.

PROTOCOLS
The system currently supports the DARPA Internet protocols and the Xerox Network Systems(tm) protocols. Raw socket interfaces are provided to the IP protocol layer of the DARPA Internet, to the IMP link layer (1822), and to the IDP protocol of Xerox NS. Consult the appropriate manual pages in this section for more information regarding the support for each protocol family.

ADDRESSING
Associated with each protocol family is an address format. The following address formats are used by the system (and additional formats are defined for possible future implementation):

```c
#define AF_UNIX 1 /* local to host (pipes, portals) */
#define AF_INET 2 /* internetwork: UDP, TCP, etc. */
#define AF_IMPLINK 3 /* arpanet imp addresses */
#define AF_PUP 4 /* pup protocols: e.g. BSP */
#define AF_NS 6 /* Xerox NS protocols */
```
#define AF_HYLINK 15 /* NSC Hyperchannel */

ROUTING
The network facilities provided limited packet routing. A simple set of data structures comprise a "routing table" used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. A user process, the routing daemon, maintains this data base with the aid of two socket-specific ioctl(2) commands, SIOCADDRT and SIOCDELRT. The commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by super-user.

A routing table entry has the following form, as defined in <net/route.h>:

```c
struct rteventy {
    u_long rt_hash;
    struct sockaddr rt_dst;
    struct sockaddr rt_gateway;
    short rtflags;
    short rt_refcnt;
    u_long rt_use;
    struct ifnet *rt_ifp;
};
```

with rt_flags defined from,

```
#define RTF_UP 0x1 /* route usable */
#define RTF_GATEWAY 0x2 /* destination is a gateway */
#define RTF_HOST 0x4 /* host entry (net otherwise) */
#define RTF_DYNAMIC 0x10 /* created dynamically (by redirect) */
```

Routing table entries come in three flavors: for a specific host, for all hosts on a specific network, for any destination not matched by entries of the first two types (a wildcard route). When the system is booted and addresses are assigned to the network interfaces, each protocol family installs a routing table entry for each interface when it is ready for traffic. Normally the protocol specifies the route through each interface as a "direct" connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface is requested to address the packet to the gateway listed in the routing entry (i.e. the packet is forwarded).

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (rt_refcnt is non-zero), the routing entry will be marked down and removed from the routing table, but the resources associated with it will not be reclaimed until all references to it are released. The routing code returns EEXIST if requested to duplicate an existing entry, ESRCH if requested to delete a non-existent entry, or ENOBUFS if insufficient resources were available to install a new route. User processes read the routing tables through the /dev/kmem device. The rt_use field contains the number of packets sent along the route.

When routing a packet, the kernel will first attempt to find a route to the destination host. Failing that, a search is made for a route to the network of the destination. Finally, any route to a default ("wildcard") gateway is chosen. If multiple routes are present in the table, the first route found will be used. If no entry is found, the destination is declared to be unreachable.
A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

INTERFACES

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, lo(4), do not.

The following ioctl calls may be used to manipulate network interfaces. The ioctl is made on a socket (typically of type SOCK_DGRAM) in the desired domain. Unless specified otherwise, the request takes an ifreq structure as its parameter. This structure has the form:

```c
struct ifreq {
    char ifr_name[16]; /* name of interface (e.g. "ec0") */
    union {
        struct sockaddr ifru_addr;
        struct sockaddr ifru_dstaddr;
        struct sockaddr ifru_broadaddr;
        short ifru_flags;
        int ifru_metric;
    } ifr_ifru;
    #define ifr_addr ifr_ifru.ifru_addr /* address */
    #define ifr_dstaddr ifr_ifru.ifru_dstaddr /* other end of p-to-p link */
    #define ifr_broadaddr ifr_ifru.ifru_broadaddr /* broadcast address */
    #define ifr_flags ifr_ifru.ifru_flags /* flags */
    #define ifr_metric ifr_ifru.ifru_metric /* routing metric */
};
```

- SIOCSIFADDR
  Set interface address for protocol family. Following the address assignment, the “initialization” routine for the interface is called.

- SIOCGIFADDR
  Get interface address for protocol family.

- SIOCSIFDSTADDR
  Set point to point address for protocol family and interface.

- SIOCGIFDSTADDR
  Get point to point address for protocol family and interface.

- SIOCSIFBRDADDR
  Set broadcast address for protocol family and interface.

- SIOCGIFBRDADDR
  Get broadcast address for protocol family and interface.

- SIOCSIFFLAGS
  Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified; some interfaces may be reset so that incoming packets are no longer received. When marked up again, the interface is reinitialized.

- SIOCGIFFLAGS
  Get interface flags.

- SIOCSIFMETRIC
  Set interface routing metric. The metric is used only by user-level routers.
SIOCGIFMETRIC
Get interface metric.

SIOCGIFCONF
Get interface configuration list. This request takes an ifconf structure (see below) as a value-result parameter. The ifc_len field should be initially set to the size of the buffer pointed to by ifc_buf. On return it will contain the length, in bytes, of the configuration list.

```c
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */
struct ifconf {
    int ifc_len;   /* size of associated buffer */
    union {
        caddr_t ifcu_buf;
        struct ifreq *ifcu_req;
    } ifc_ifcu;
#define ifc_buf ifc_ifcu.ifcu_buf  /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req  /* array of structures returned */
};
```

SEE ALSO
socket(2), ioctl(2), intro(4), config(8), routed(8C)
NAME
acc - ACC LH/DH IMP interface

SYNOPSIS
pseudo-device imp
device acc0 at uba0 csr 167600 vector accrnt accxnt

DESCRIPTION
The acc device provides a Local Host/Distant Host interface to an IMP. It is normally used when participating in the DARPA Internet. The controller itself is not accessible to users, but instead provides the hardware support to the IMP interface described in imp(4). When configuring, the imp pseudo-device must also be included.

DIAGNOSTICS
acc%d: not alive. The initialization routine was entered even though the device did not autoconfigure. This indicates a system problem.
acc%d: can't initialize. Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).
acc%d: imp doesn't respond, icsr=%b. The driver attempted to initialize the device, but the IMP failed to respond after 500 tries. Check the cabling.
acc%d: stray xmit interrupt, csr=%b. An interrupt occurred when no output had previously been started.
acc%d: output error, ocsr=%b, icsr=%b. The device indicated a problem sending data on output.
acc%d: input error, csr=%b. The device indicated a problem receiving data on input.
acc%d: bad length=%d. An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never happen as the maximum size of a host-IMP message is 1008 bytes.
NAME
   ad – Data Translation A/D converter

SYNOPSIS
   device ad0 at uba0 csr 0170400 vector adintr

DESCRIPTION
   Ad provides the interface to the Data Translation A/D converter. This is not a real-time
   driver, but merely allows the user process to sample the board’s channels one at a time. Each
   minor device selects a different A/D board.

   The driver communicates to a user process by means of ioctls. The AD_CHAN ioctl selects
   which channel of the board to read. For example,
      chan = 5; ioctl(fd, AD_CHAN, &chan);
   selects channel 5. The AD_READ ioctl actually reads the data and returns it to the user
   process. An example is
      ioctl(fd, AD_READ, &data);

FILES
   /dev/ad

DIAGNOSTICS
   None.
NAME
arp – Address Resolution Protocol

SYNOPSIS
pseudo-device ether

DESCRIPTION
ARP is a protocol used to dynamically map between DARPA Internet and 10Mb/s Ethernet addresses. It is used by all the 10Mb/s Ethernet interface drivers. It is not specific to Internet protocols or to 10Mb/s Ethernet, but this implementation currently supports only that combination.

ARP caches Internet-Ethernet address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message which requires the mapping and broadcasts a message on the associated network requesting the address mapping. If a response is provided, the new mapping is cached and any pending message is transmitted. ARP will queue at most one packet while waiting for a mapping request to be responded to; only the most recently “transmitted” packet is kept.

To facilitate communications with systems which do not use ARP, ioctl’s are provided to enter and delete entries in the Internet-to-Ethernet tables. Usage:

```c
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>

struct arpreq arpreq;

ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```

Each ioctl takes the same structure as an argument. SIOCSARP sets an ARP entry, SIOCGARP gets an ARP entry, and SIOCDARP deletes an ARP entry. These ioctl’s may be applied to any socket descriptor `s`, but only by the super-user. The arpreq structure contains:

```c
/*
 * ARP ioctl request
 */

struct arpreq {
    struct sockaddr      arp_pa; /* protocol address */
    struct sockaddr      arp_ha; /* hardware address */
    int                  arp_flags; /* flags */
};
```

The address family for the `arp_pa` sockaddr must be AF_INET; for the `arp_ha` sockaddr it must be AF_UNSPEC. The only flag bits which may be written are ATF_PERM, ATF_PUBL and ATF_USETRAILERS. ATF_PERM causes the entry to be permanent if the ioctl call succeeds. The peculiar nature of the ARP tables may cause the ioctl to fail if more than 8 (permanent) Internet host addresses hash to the same slot. ATF_PUBL specifies that the ARP code should respond to ARP requests for the indicated host coming from other machines. This allows a host to act as an “ARP server,” which may be useful in convincing an ARP-only machine to talk to a non-ARP machine.
ARP is also used to negotiate the use of trailer IP encapsulations; trailers are an alternate encapsulation used to allow efficient packet alignment for large packets despite variable-sized headers. Hosts which wish to receive trailer encapsulations so indicate by sending gratuitous ARP translation replies along with replies to IP requests; they are also sent in reply to IP translation replies. The negotiation is thus fully symmetrical, in that either or both hosts may request trailers. The ATF_USETRAILERS flag is used to record the receipt of such a reply, and enables the transmission of trailer packets to that host.

ARP watches passively for hosts impersonating the local host (i.e. a host which responds to an ARP mapping request for the local host's address).

**DIAGNOSTICS**

duplicate IP address sent from ethernet address: %x:%x:%x:%x:%x:%x. ARP has discovered another host on the local network which responds to mapping requests for its own Internet address.

**SEE ALSO**

dc(4), de(4), il(4), inet(4F), arp(8C), ifconfig(8C)

"An Ethernet Address Resolution Protocol," RFC826, Dave Plummer, Network Information Center, SRI.

"Trailer Encapsulations," RFC893, S.J. Leffler and M.J. Karels, Network Information Center, SRI.

**BUGS**

ARP packets on the Ethernet use only 42 bytes of data; however, the smallest legal Ethernet packet is 60 bytes (not including CRC). Some systems may not enforce the minimum packet size, others will.
NAME
autoconf – diagnostics from the autoconfiguration code

DESCRIPTION
When UNIX bootstraps it probes the innards of the machine on which it is running and
locates controllers, drives, and other devices, printing out what it finds on the console. This
procedure is driven by a system configuration table which is processed by config(8) and compi­
elle into each kernel.

On the VAX, devices in NEXUS slots are normally noted, thus memory controllers, UNIBUS
and MASSBUS adaptors. Devices which are not supported which are found in NEXUS slots
are noted also. The Q-bus on the MICROVAX is configured in the same way as the
UNIBUS.

MASSBUS devices are located by a very deterministic procedure since MASSBUS space is
completely probe-able. If devices exist which are not configured they will be silently ignored;
if devices exist of unsupported type they will be noted.

UNIBUS devices are located by probing to see if their control-status registers respond. If not,
they are silently ignored. If the control status register responds but the device cannot be
made to interrupt, a diagnostic warning will be printed on the console and the device will not
be available to the system.

Normally, the system uses the disk from which it was loaded as the root filesystem. If that is
not possible, a generic system will pick its root device as the “best” available device
(MASSBUS disks are better than SMD UNIBUS disks are better than RK07’s; the device
must be drive 0 to be considered). If such a system is booted with the RB_ASKNAME
option (see reboot(2)), then the name of the root device is read from the console terminal at
boot time, and any available device may be used.

SEE ALSO
intro(4), boot(8), config(8)

DIAGNOSTICS
cpu type %d not configured. You tried to boot UNIX on a cpu type which it doesn’t (or at
least this compiled version of UNIX doesn’t) understand.

mba%d at tr%d. A MASSBUS adapter was found in tr%d (the NEXUS slot number). UNIX
will call it mba%d.

%d mba’s not configured. More MASSBUS adapters were found on the machine than were
declared in the machine configuration; the excess MASSBUS adapters will not be accessible.

uba%d at tr%d. A UNIBUS adapter was found in tr%d (the NEXUS slot number). UNIX
will call it uba%d.

dr32 unsupported (at tr %d). A DR32 interface was found in a NEXUS, for which UNIX
does not have a driver.

ci unsupported (at tr %d). A CI interface was found in a NEXUS, for which UNIX does not
have a driver.

mcr%d at tr%d. A memory controller was found in tr%d (the NEXUS slot number). UNIX
will call it mcr%d.

5 mcr’s unsupported. UNIX supports only 4 memory controllers per cpu.

mpm unsupported (at tr%d). Multi-port memory is unsupported in the sense that UNIX does
not know how to poll it for ECC errors.

%s%d at mba%d drive %d. A tape formatter or a disk was found on the MASSBUS; for disks
%s%d will look like “hp0”, for tape formatters like “ht1”. The drive number comes from the
unit plug on the drive or in the TM formatter (not on the tape drive; see below).
%s%d at %s%d slave %d. (For MASSBUS devices). Which would look like “tu0 at ht0 slave 0”, where tu0 is the name for the tape device and ht0 is the name for the formatter. A tape slave was found on the tape formatter at the indicated drive number (on the front of the tape drive). UNIX will call the device, e.g., tu0.

%s%d at uba%d csr %o vec %o ipl %x. The device %s%d, e.g. dz0 was found on uba%d at control-status register address %o and with device vector %o. The device interrupted at priority level %x.

%s%d at uba%d csr %o zero vector. The device did not present a valid interrupt vector, rather presented 0 (a passive release condition) to the adapter.

%s%d at uba%d csr %o didn't interrupt. The device did not interrupt, likely because it is broken, hung, or not the kind of device it is advertised to be.

%s%d at %s%d slave %d. (For UNIBUS devices). Which would look like “up0 at sc0 slave 0”, where up0 is the name of a disk drive and sc0 is the name of the controller. Analogous to MASSBUS case.
NAME
bk – line discipline for machine-machine communication (obsolete)

SYNOPSIS
pseudo-device bk

DESCRIPTION
This line discipline provides a replacement for the old and new tty drivers described in tty(4)
when high speed output to and especially input from another machine is to be transmitted
over a asynchronous communications line. The discipline was designed for use by the Berke­
ley network. It may be suitable for uploading of data from microprocessors into the system.
If you are going to send data over asynchronous communications lines at high speed into the
system, you must use this discipline, as the system otherwise may detect high input data rates
on terminal lines and disables the lines; in any case the processing of such data when normal
terminal mechanisms are involved saturates the system.

The line discipline is enabled by a sequence:

#include <sgtty.h>
int ldisc = NETLDISC, fildes; ...
ioctl(fildes, TIOCSETD, &ldisc);

A typical application program then reads a sequence of lines from the terminal port, checking
header and sequencing information on each line and acknowledging receipt of each line to the
sender, who then transmits another line of data. Typically several hundred bytes of data and
a smaller amount of control information will be received on each handshake.

The old standard teletype discipline can be restored by doing:

ldisc = OTTYDISC;
ioctl(fildes, TIOCSETD, &ldisc);

While in networked mode, normal teletype output functions take place. Thus, if an 8 bit out­
put data path is desired, it is necessary to prepare the output line by putting it into RAW
mode using ioctl(2). This must be done before changing the discipline with TIOCSETD, as
most ioctl(2) calls are disabled while in network line-discipline mode.

When in network mode, input processing is very limited to reduce overhead. Currently the
input path is only 7 bits wide, with newline the only recognized character, terminating an
input record. Each input record must be read and acknowledged before the next input is read
as the system refuses to accept any new data when there is a record in the buffer. The buffer
is limited in length, but the system guarantees to always be willing to accept input resulting in
512 data characters and then the terminating newline.

User level programs should provide sequencing and checksums on the information to guaran­
tee accurate data transfer.

SEE ALSO
tty(4)

DIAGNOSTICS
None.

BUGS
The Purdue uploading line discipline, which provides 8 bits and uses timeout’s to terminate
uploading should be incorporated into the standard system, as it is much more suitable for
microprocessor connections.
NAME
cons - VAX-11 console interface

DESCRIPTION
The console is available to the processor through the console registers. It acts like a normal terminal, except that when the local functions are not disabled, control-P puts the console in local console mode (where the prompt is “>>>”). The operation of the console in this mode varies slightly per-processor.

On an 11/780 or 785 the processor is not stopped by entering local console mode. The CPU may be halted with the “halt” command, which may be abbreviated to “h.” Conversational mode is re-entered by using the command “set tp” (set terminal program) if the processor is still running, or “continue” if it is halted. The latter command may be abbreviated “c”. If you hit the break key on the console, then the console LSI-11 will go into ODT (console debugger mode). Hit a “P” (upper-case letter p; “proceed”) to get out of this mode.

On an 11/750 or an 11/730 the processor is halted whenever the console is not in conversational mode, and typing “C” returns to conversational mode. When in console mode on an 11/750 which has a remote diagnosis module, a “D” will put you in remote diagnosis mode, where the prompt will be “ROM>”. The command “ret” will return from remote diagnosis mode to local console mode.

The VAX 8600 (8650) console normally works in the same way as the 11/750, except that there are many additional modes and commands. In the normal mode control-P halts the processor, and “c” or “continue” returns to conversational mode. If HEX debug is enabled, control-P does not halt the CPU; the “halt” command stops the CPU as on the 11/780.

With the above provisos the console works like any other UNIX terminal.

FILES
/dev/console

SEE ALSO
tty(4), reboot(8)
VAX Hardware Handbook
NAME
crl - VAX 8600 console RL02 interface

DESCRIPTION
This is a simple interface to the DEC RL02 disk unit which is part of the console subsystem on the VAX 8600 and 8650. Access is given to the entire RL02 disk; the pack format is the same as that of RL02 disks on other controllers. As on other VAX console media, transfers are done a word at a time using privileged registers (i.e., slowly).

All I/O is raw; the seek addresses in raw transfers should be a multiple of 512 bytes and a multiple of 512 bytes should be transferred, as in other "raw" disk interfaces. (Although the sector size is actually 256 bytes, the driver allows operations only on 512-byte boundaries.)

FILES
/dev/crl

SEE ALSO
arff(8V)
NAME
css – DEC IMP-11A LH/DH IMP interface

SYNOPSIS
pseudo-device imp
device css0 at uba0 csr 167600 flags 10 vector cssrint cssxint

DESCRIPTION
The css device provides a Local Host/Distant Host interface to an IMP. It is normally used when participating in the DARPA Internet. The controller itself is not accessible to users, but instead provides the hardware support to the IMP interface described in imp(4). When configuring, the imp pseudo-device is also included.

DIAGNOSTICS
css%d: not alive. The initialization routine was entered even though the device did not autoconfigure. This indicates a system problem.
css%d: can't initialize. Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).
css%d: imp doesn’t respond, icsr=%b. The driver attempted to initialize the device, but the IMP failed to respond after 500 tries. Check the cabling.
css%d: stray output interrupt csr=%b. An interrupt occurred when no output had previously been started.
css%d: output error, ocsr=%b icsr=%b. The device indicated a problem sending data on output.
css%d: recv error, csr=%b. The device indicated a problem receiving data on input.
css%d: bad length=%d. An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never happen as the maximum size of a host-IMP message is 1008 bytes.
NAME
ct – phototypesetter interface

SYNOPSIS
device ct0 at uba0 csr 0167760 vector ctinr

DESCRIPTION
This provides an interface to a Graphic Systems C/A/T phototypesetter or an Autologic APS-Micro5 using a DR11C interface. Bytes written on the file specify font, size, and other control information as well as the characters to be flashed. The coding is not described here. Only one process may have this file open at a time. It is write-only.

FILES
/dev/cat

SEE ALSO
troff(1)
Phototypesetter interface specification

DIAGNOSTICS
None.
NAME
ddn – DDN Standard Mode X.25 IMP interface

SYNOPSIS
device ddn0 at uba0 csr 166740 vector ddnintr

DESCRIPTION
The ddn device provides a DDN Standard Mode X.25 interface to an IMP using the ACC
ACP62S X.25 board. It is normally used for connecting to the Defense Data Network
(DDN). The controller itself is not accessible to users, but instead provides a network inter­
face for the Internet Protocol described in ip(4P).

SEE ALSO
intro(4N), ip(4P)

DIAGNOSTICS

ddn%d: not alive. The initialization routine was entered even though the device did not
autoconfigure. This indicates a system problem.

ddn%d: failed getting UBA resources for lcn %d. Insufficient UNIBUS resources existed to
initialize the device. This is likely to be a shortage of UNIBUS mapping registers.

ddn%d: couldn't get X25 init buffer. This indicates that an mbuf could not be allocated for
sending the initialization message to the ACP625.

DDN: illegal X25 address length!
DDN: illegal X25 address format!
These errors indicate a problem with the called X.25 address received from the IMP on an
incoming call.

X25 RESET on lcn = %d. This indicates that an unexpected X.25 RESET was received on
the indicated LCN.

X25 INTERRUPT on lcn = %d, code = %d. This indicates that an unexpected X.25 INTER­
RUPT Packet was received on the indicated LCN.

ddn%d: failed to get supr msg bfr!. This indicates that an mbuf could not be allocated for
sending a supervisor message to the ACP625.

Any other error message from ddn%d: indicates a serious error detected by either the driver
or the ACP625 firmware.
NAME
de – DEC DEUNA 10 Mb/s Ethernet interface

SYNOPSIS
de at uba0 csr 174510 vector deintr

DESCRIPTION
The de interface provides access to a 10 Mb/s Ethernet network through a Digital Equipment UNIBUS Network Adapter (DEUNA).

Each of the host's network addresses is specified at boot time with an SIOCSIFADDR ioctl. The de interface employs the address resolution protocol described in arp(4P) to dynamically map between Internet and Ethernet addresses on the local network.

The interface normally tries to use a "trailer" encapsulation to minimize copying data on input and output. The use of trailers is negotiated with ARP. This negotiation may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIF-FLAGS ioctl.

DIAGNOSTICS
de%d: hardware address %. This is a normal autoconfiguration message noting the 6 byte physical ethernet address of the adapter.
de%d: oerror, flags=%b tdrerr=%b (len=%d). The hardware indicated an error in transmitting a packet to the cable. The status and error flags are reported.
de%d: ierror, flags=%b lenerr=%b (len=%d). The hardware indicated an error in reading a packet from the cable. The status and error flags are reported.
de%d: can't handle af%. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.
de%d: buffer unavailable. The interface received more packets than it had buffers allocated to receive them.
de%d: address change failed, csr0=%b csr1=%b. The interface was unable to reprogram its physical ethernet address. This may happen with very early models of the interface. This facility is used only when the controller is not the first network interface configured for XNS.
The following messages indicate a probable hardware error performing the indicated operation during autoconfiguration or initialization. The two control and status registers should indicate the nature of the failure. See the hardware manual for details.
de%d: reset failed, csr0=%b csr1=%b.
de%d: ppcb failed, csr0=%b csr1=%b.
de%d: read addr failed, csr0=%b csr1=%b.
de%d: wstring failed, csr0=%b csr1=%b.
de%d: wtmode failed, csr0=%b csr1=%b.

SEE ALSO
intro(4N), inet(4F), arp(4P)
NAME
dh – DH-11/DM-11 communications multiplexer

SYNOPSIS
device dh0 at uba0 csr 0160020 vector dhintr dhxint
device dm0 at uba0 csr 0170500 vector dmintr

DESCRIPTION
A dh-11 provides 16 communication lines; dm-11's may be optionally paired with dh-11's to provide modem control for the lines.

Each line attached to the DH-11 communications multiplexer behaves as described in tty(4). Input and output for each line may independently be set to run at any of 16 speeds; see tty(4) for the encoding.

Bit i of flags may be specified for a dh to say that a line is not properly connected, and that the line should be treated as hard-wired with carrier always present. Thus specifying "flags 0x0004" in the specification of dh0 would cause line ttyh2 to be treated in this way.

The dh driver monitors the rate of input on each board, and switches between the use of character-at-a-time interrupts and input silos. While the silo is enabled during periods of high-speed input, the driver polls for input 30 times per second.

FILES
/dev/tty[h-o][0-9a-t]
/dev/ttyd[0-9a-t]

SEE ALSO
tty(4)

DIAGNOSTICS
dh%d: NXM. No response from UNIBUS on a dma transfer within a timeout period. This is often followed by a UNIBUS adapter error. This occurs most frequently when the UNIBUS is heavily loaded and when devices which hog the bus (such as rk07's) are present. It is not serious.

dh%d: silo overflow. The character input silo overflowed before it could be serviced. This can happen if a hard error occurs when the CPU is running with elevated priority, as the system will then print a message on the console with interrupts disabled. It is not serious.
NAME
dhu – DHU-11 communications multiplexer

SYNOPSIS
device dhu0 at uba0 csr 0160440 vector dhurint dhuxint

DESCRIPTION
A DHU-11 provides 16 communication lines.
Each line attached to the DHU-11 communications multiplexer behaves as described in tty(4). Input and output for each line may independently be set to run at any of 13 speeds (50, 200 and 38400 baud are not available); see tty(4) for the encoding.

Bit i of flags may be specified for a DHU-11 to say that a line is not properly connected, and that the line should be treated as hard-wired with carrier always present. Thus specifying “flags 0x0004” in the specification of dhu0 would cause line ttyS2 to be treated in this way.

The DHU-11 driver normally uses input silos and delays receiver interrupts by 20 milliseconds rather than taking an interrupt on each input character.

FILES
/dev/tty[S-Z][0-9a-f]

SEE ALSO
tty(4)

DIAGNOSTICS
dhu(%d,%d): NXM fault. No response from UNIBUS on a DMA transfer within a timeout period. This is often followed by a UNIBUS adapter error. This occurs most frequently when the UNIBUS is heavily loaded and when devices which hog the bus (such as RK07s) are present. It is not serious.

dhu%d: silo overflow. The character input silo overflowed before it could be serviced. This can happen if a hard error occurs when the CPU is running with elevated priority, as the system may then print a message on the console with interrupts disabled.

NOTES
The driver currently does not make full use of the hardware capabilities of the DHU-11, for dealing with XON/XOFF flow-control or hard-wired lines for example.

Although the devices are not the same, a DHU-11 can convince the DH-11 autoconfiguration code that it is a DH-11.

The 440-way cables are a pain.
NAME
dmc – DEC DMC-11/DMR-11 point-to-point communications device

SYNOPSIS
device dmc0 at uba0 csr 167600 vector dmcrint dmcxint

DESCRIPTION
The dmc interface provides access to a point-to-point communications device which runs at either 1 Mb/s or 56 Kb/s. DMC-11’s communicate using the DEC DDCMP link layer protocol.

The dmc interface driver also supports a DEC DMR-11 providing point-to-point communication running at data rates from 2.4 Kb/s to 1 Mb/s. DMR-11’s are a more recent design and thus are preferred over DMC-11’s. The NXMT and NRCV constants in the driver should be increased in this case, as the DMR can accept up to 64 transmit and receive buffers, as opposed to 7 for the DMC.

The configuration flags specify how to set up the device,
0 – full duplex DDCMP (normal mode)
1 – DDCMP Maintenance mode (generally useless)
2 – DDCMP Half Duplex, primary station
3 – DDCMP Half Duplex, secondary station

Several device error counters are available via "adb", for more information see the adb script /usr/lib/adb/dmcstats, or the DMC11 technical manual.

The host’s address must be specified with an SIOCSIFADDR ioctl, and the destination address specified with a SIOCSIFDSTADDR ioctl, before the interface will transmit or receive any packets.

ROUTING
The driver places a HOST entry in the kernel routing tables for the address given in the SIOCSIFDSTADDR ioctl. To use the DMC as a link between local nets, the route to the remote net must be added manually with the route(8) command, or by the use of the routing process routed(8) on each end of the link.

DIAGNOSTICS
dmc%d: bad control %. A bad parameter was passed to the dmcload routine.
dmc%d: unknown address type %. An input packet was received which contained a type of address unknown to the driver.
DMC fatal error 0%. A fatal error in DDMCP occurred, causing the device to be restarted.
DMC soft error 0%. A non-fatal error in DDMCP has occurred.
dmc%d: af%d not supported. The interface was handed a message which has addresses formatted in an unsuitable address family.

SEE ALSO
intro(4N), inet(4F)

BUGS
The current version of the driver uses a link-level encapsulation so that multiple protocol types may be used. It is thus incompatible with earlier drivers, including the 4.2BSD version.
NAME
dmf – DMF-32, terminal multiplexor

SYNOPSIS
device dmf0 at uba? csr 0160340
vector dmfprint dmfdsxint dmfdaaint dmfdbhint dmfrint dmfint dmflint

DESCRIPTION
The dmf device provides 8 lines of asynchronous serial line support. The first two of these
have full modem control. The device also provides a line printer port similar to the LP-11.
Other features of the DMF-32 are not supported. During autoconfiguration, the driver exam­
ines the configuration of each DMF-32 and adjusts the interrupt vectors so that fewer vector
locations are used if possible.

Each line attached to a DMF-32 serial line port behaves as described in tty(4). Input and out­
put for each line may independently be set to run at any of 16 speeds; see tty(4) for the
encoding.

Bit i of flags may be specified for a dmf to to say that a line is not properly connected, and
that the line should be treated as hard-wired with carrier always present. Thus specifying
“flags 0x04” in the specification of dmf0 would cause line ttyA2 to be treated in this way.
Flags should be set for all lines without hardware support for modem control.

The serial line part of the dmf driver normally enables the input silos with a short timeout (30
milliseconds); this allows multiple characters to be received per interrupt during periods of
high-speed input.

A line printer port on dmf/n is designated by a minor device number of the form 128+n.
Columns and lines per page may be changed from the default 132 columns and 66 lines by
encoding the number of columns in bits 8-15 of flags and the number of lines in bits 16-23.
This device does not provide the fancy output canonicalization features of the lp(4) driver.

FILES
/dev/tty[A-CE-I][0-7]
/dev/ttyd[0-7]
/dev/lp

SEE ALSO
tty(4)

diagnostic

dmfsrint, dmfdsxint, dmfdaaint, dmfdbhint. One of the unsupported parts of the dmf interrupted;
something is amiss, check your interrupt vectors for a conflict with another device.

BUGS
It should be possible to set the silo timeout with a configuration file option, as the value is a
trade-off between efficiency and response time for flow control and character echo.
NAME
dmz - DMZ-32 terminal multiplexor

SYNOPSIS
device dmz0 at uba? csr 0160540
   vector dmzrinta dmzxinat dmzrintb dmzxinatb dmzrintc dmzxinatc

DESCRIPTION
The dmz device provides 24 lines of asynchronous serial line support. Modem control on all ports is available as an option for the H3014 distribution panel.

Each line attached to a DMZ-32 serial line port behaves as described in tty(4). Input and output for each line may independently be set to run at any of 16 speeds; see tty(4) for the encoding.

Bit i of flags may be specified for a dmz to say that a line is not properly connected, and that the line should be treated as hard-wired with carrier always present. Thus specifying "flags 0x000004" in the specification of dmz0 would cause line ttya2 to be treated in this way.

The dmz driver normally enables the input silos with a short timeout (30 milliseconds); this allows multiple characters to be received per interrupt during periods of high-speed input.

FILES
/dev/tty[abcefg][0-9a-n]

SEE ALSO
tty(4)

DIAGNOSTICS
dmz%d: NXM line %d. No response from the UNIBUS on a DMA transfer within a timeout period. This is often followed by a UNIBUS adapter error. This occurs most frequently when the UNIBUS is heavily loaded and when devices which hog the bus (such as RK07s) are present. It is not serious.

dmz%d: silo overflow. The character input silo overflowed before it could be serviced. This can happen if a hard error occurs when the CPU is running with elevated priority, as the system will then print a message on the console with interrupts disabled. It is not serious.

BUGS
It should be possible to set the silo timeout with a configuration file option, as the value is a trade-off between efficiency and response time for flow control and character echo.
NAME

dn – DN-11 autocall unit interface

SYNOPSIS

device dn0 at uba? csr 0160020 vector dnnintr

DESCRIPTION

The dn device provides an interface through a DEC DN-11 (or equivalent such as the Able Quadracall) to an auto-call unit (ACU). To place an outgoing call one forks a sub-process which opens the appropriate call unit file, /dev/cua? and writes the phone number on it. The parent process then opens the corresponding modem line /dev/cul?. When the connection has been established, the open on the modem line, /dev/cul? will return and the process will be connected. A timer is normally used to timeout the opening of the modem line.

The codes for the phone numbers are:

0-9  dial 0-9
*  dial * ('.' is a synonym)
#  dial # ('.' is a synonym)
-  delay 20 milliseconds
<  end-of-number ('e' is a synonym)
=  delay for a second dial tone ('w' is a synonym)
f  force a hangup of any existing connection

The entire telephone number must be presented in a single write system call.

By convention, even numbered call units are for 300 baud modem lines, while odd numbered units are for 1200 baud lines. For example, /dev/cua0 is associated with a 300 baud modem line, /dev/cul0, while /dev/cua1 is associated with a 1200 baud modem line, /dev/cul1. For devices such as the Quadracall which simulate multiple DN-11 units, the minor device indicates which outgoing modem to use.

FILES

/dev/cua?  call units
/dev/cul?  associated modem lines

SEE ALSO

tip(1C)

DIAGNOSTICS

Two error numbers are of interest at open time.

[EBUSY]  The dialer is in use.

[ENXIO]  The device doesn’t exist, or there’s no power to it.
NAME
   drum - paging device

DESCRIPTION
   This file refers to the paging device in use by the system. This may actually be a subdevice of one of the disk drivers, but in a system with paging interleaved across multiple disk drives it provides an indirect driver for the multiple drives.

FILES
   /dev/drum

BUGS
   Reads from the drum are not allowed across the interleaving boundaries. Since these only occur every .5Mbytes or so, and since the system never allocates blocks across the boundary, this is usually not a problem.
NAME
dz – DZ-11 communications multiplexer

SYNOPSIS
device dz0 at uba0 csr 0160100 vector dzrint dzxint

DESCRIPTION
A DZ11 provides 8 communication lines with partial modem control, adequate for UNIX
dialup use. Each line attached to the DZ11 communications multiplexer behaves as described
in tty(4) and may be set to run at any of 16 speeds; see tty(4) for the encoding.

Bit i of flags may be specified for a dz to say that a line is not properly connected, and that
the line should be treated as hard-wired with carrier always present. Thus specifying “flags
0x04” in the specification of dz0 would cause line tty02 to be treated in this way.

The dz driver monitors the rate of input on each board, and switches between the use of
character-at-a-time interrupts and input silos. While the silo is enabled during periods of
high-speed input, the driver polls for input 30 times per second.

FILES
/dev/tty[0-9][0-9]
/dev/ttyd[0-9a-f] dialups

SEE ALSO
tty(4)

DIAGNOSTICS
dzinf: silo overflow. The 64 character input silo overflowed before it could be serviced. This
can happen if a hard error occurs when the CPU is running with elevated priority, as the sys-
tem will then print a message on the console with interrupts disabled. It is not serious.
NAME
ec – 3Com 10 Mb/s Ethernet interface

SYNOPSIS
device ec0 at uba0 csr 161000 vector ecrint eccolide ecxint flags 0

DESCRIPTION
The ec interface provides access to a 10 Mb/s Ethernet network through a 3Com controller.
The hardware has 32 kilobytes of dual-ported memory on the UNIBUS. This memory is used
for internal buffering by the board, and the interface code reads the buffer contents directly
through the UNIBUS. The address of this memory is given in the flags field in the
configuration file. The first interface normally has its memory at Unibus address 0.

Each of the host's network addresses is specified at boot time with an SIOCSIFADDR ioctl.
The ec interface employs the address resolution protocol described in arp(4P) to dynamically
map between Internet and Ethernet addresses on the local network.

The interface normally tries to use a “trailer” encapsulation to minimize copying data on
input and output. The use of trailers is negotiated with ARP. This negotiation may be dis­
abled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIF­
FLAGS ioctl.

The interface software implements an exponential backoff algorithm when notified of a colli­
sion on the cable. This algorithm utilizes a 16-bit mask and the VAX-11's interval timer in
calculating a series of random backoff values. The algorithm is as follows:
1. Initialize the mask to be all 1's.
2. If the mask is zero, 16 retries have been made and we give up.
3. Shift the mask left one bit and formulate a backoff by masking the interval timer with
   the smaller of the complement of this mask and a 5-bit mask, resulting in a pseudo­
   random number between 0 and 31. This produces the number of slot times to delay,
   where a slot is 51 microseconds.
4. Use the value calculated in step 3 to delay before retransmitting the packet. The delay
   is done in a software busy loop.

DIAGNOSTICS
ec%d: send error. After 16 retransmissions using the exponential backoff algorithm described
above, the packet was dropped.
ec%d: input error (offset=0%d). The hardware indicated an error in reading a packet off the
cable or an illegally sized packet. The buffer offset value is printed for debugging purposes.
ec%d: can't handle af%d. The interface was handed a message with addresses formatted in an
unsuitable address family; the packet was dropped.

SEE ALSO
intro(4N), inet(4F), arp(4P)

BUGS
The hardware is not capable of talking to itself. The software implements local sending and
broadcast by sending such packets to the loop interface. This is a kludge.
Backoff delays are done in a software busy loop. This can degrade the system if the network
experiences frequent collisions.
NAME

en – Xerox 3 Mb/s Ethernet interface

SYNOPSIS

device en0 at uba0 csr 161000 vector enrint enxint encollide

DESCRIPTION

The en interface provides access to a 3 Mb/s Ethernet network. Due to limitations in the hardware, DMA transfers to and from the network must take place in the lower 64K bytes of the UNIBUS address space, and thus this must be among the first UNIBUS devices enabled after boot.

Each of the host's network addresses is specified at boot time with an SIOCSIFADDR ioctl. The station address is discovered by probing the on-board Ethernet address register, and is used to verify the protocol addresses. No packets will be sent or accepted until a network address is supplied.

The interface software implements an exponential backoff algorithm when notified of a collision on the cable. This algorithm utilizes a 16-bit mask and the VAX-11's interval timer in calculating a series of random backoff values. The algorithm is as follows:

1. Initialize the mask to be all 1's.
2. If the mask is zero, 16 retries have been made and we give up.
3. Shift the mask left one bit and formulate a backoff by masking the interval timer with the mask (this is actually the two's complement of the value).
4. Use the value calculated in step 3 to delay before retransmitting the packet.

The interface handles both Internet and NS protocol families. It normally tries to use a "trailer" encapsulation to minimize copying data on input and output. The use of trailers is negotiated with ARP. This negotiation may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS

en%d: output error. The hardware indicated an error on the previous transmission.
en%d: send error. After 16 retransmissions using the exponential backoff algorithm described above, the packet was dropped.
en%d: input error. The hardware indicated an error in reading a packet off the cable.
en%d: can't handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

SEE ALSO

intro(4N), inet(4F)

BUGS

The device has insufficient buffering to handle back to back packets. This makes use in a production environment painful.

The hardware does word at a time DMA without byte swapping. To compensate, byte swapping of user data must either be done by the user or by the system. A kludge to byte swap only IP packets is provided if the ENF_SWABIPS flag is defined in the driver and set at boot time with an SIOCSIFFLAGS ioctl.
NAME
  ex – Excelan 10 Mb/s Ethernet interface

SYNOPSIS
  device ex0 at uba0 csr 164000 vector excdint

DESCRIPTION
  The ex interface provides access to a 10 Mb/s Ethernet network through an Excelan controller
  used as a link-layer interface.

  Each of the host's network addresses is specified at boot time with an SIOCSIFADDR ioctl. The ex
  interface employs the address resolution protocol described in arp(4P) to dynamically
  map between Internet and Ethernet addresses on the local network.

  The interface normally tries to use a “trailer” encapsulation to minimize copying data on
  input and output. The use of trailers is negotiated with ARP. This negotiation may be dis­
  abled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIF­
  FLAGS ioctl.

DIAGNOSTICS
  ex%d: HW %c.%c, NX %c.%c, hardware address %s. This provides firmware revisions levels,
  and is expected during autoconfiguration.

  ex%d: can't initialize. There was a failure in allocating unibus resources for the device.

  ex%d: configuration failed; cc = %x. The hardware indicated an error when trying to initalize
  itself. The error code returned is described at length in the device Reference Manual.

  ex%d: receive error %b. The hardware indicated an error in reading a packet from the cable.
  Specific Error bits are provided

  ex%d: transmit error %b. The hardware indicated an error in transmitting a packet to the
  cable or an illegally sized packet. Specific Error bits are provided

  ex%d: can't handle af%d. The interface was handed a message with addresses formatted in an
  unsuitable address family; the packet was dropped.

SEE ALSO
  intro(4N), inet(4F), arp(4P)
NAME
fl – console floppy interface

DESCRIPTION
This is a simple interface to the DEC RX01 floppy disk unit, which is part of the console
LSI-11 subsystem for VAX-11/780’s. Access is given to the entire floppy consisting of 77
tracks of 26 sectors of 128 bytes.

All i/o is raw; the seek addresses in raw transfers should be a multiple of 128 bytes and a mul­
tiple of 128 bytes should be transferred, as in other “raw” disk interfaces.

FILES
/dev/floppy

SEE ALSO
arff(8V)

DIAGNOSTICS
None.

BUGS
Multiple console floppies are not supported.

If a write is given with a count not a multiple of 128 bytes then the trailing portion of the last
sector will be zeroed.
NAME
    hdh – ACC IF-11/HDH IMP interface

SYNOPSIS
    pseudo-device imp
device hdh0 at uba0 csr 166740 vector hdhintr

DESCRIPTION
    The hdh device provides an HDLC Host (HDH) interface to an IMP. It is normally used
    when participating in the DARPA Internet. The controller itself is not accessible to users, but
    instead provides the hardware support to the IMP interface described in imp(4). When
    configuring, the imp pseudo-device must also be included.

DIAGNOSTICS
    hdh%d: not alive. The initialization routine was entered even though the device did not
    autoconfigure. This indicates a system problem.
    hdh%d: cannot get chan %d uba resources. Insufficient UNIBUS resources existed to initialize
    the device. This is likely to be a shortage of UNIBUS mapping registers.
    hdh%d: LINE UP. This indicates that both the HDLC and HDH protocols have declared the
    link to the IMP alive.
    hdh%d: LINE DOWN. This indicates that the link to the IMP has died.
    hdh%d: HOST SEQUENCE ERROR
    hdh%d: IMP SEQUENCE ERROR
    hdh%d: HOST DATA ERROR
    hdh%d: TIMEOUT
    These errors indicate that an HDH protocol error has been detected.
    hdh%d: cannot get supervisor cmd buffer. This error indicates that an mbuf could not be
    allocated to send a command to the IF-11/HDH.

Any other error message from hdh%d: indicates a serious error detected by either the driver
or the IF-11/HDH firmware.
NAME
hk - RK6-11/RK06 and RK07 moving head disk

SYNOPSIS
controller hk0 at uba? csr 0177440 vector rkintr
disk rk0 at hk0 drive 0

DESCRIPTION
Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, etc. The standard device names begin with "hk" followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-7.

The block files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise seek calls should specify a multiple of 512 bytes.

DISK SUPPORT
The origin and size (in sectors) of the pseudo-disks on each drive are as follows:

RK07 partitions:
disk start length cyl
hk?a 0 15884 0-240
hk?b 15906 10032 241-392
hk?c 0 53790 0-814
hk?d 25938 15884 393-633
hk?f 41844 11792 634-814
hk?g 25938 27786 393-813

RK06 partitions:
disk start length cyl
hk?a 0 15884 0-240
hk?b 15906 11154 241-409
hk?c 0 27126 0-410

On a dual RK-07 system partition hk?a is used for the root for one drive and partition hk?g for the /usr file system. If large jobs are to be run using hk?b on both drives as swap area provides a 10Mbyte paging area. Otherwise partition hk?c on the other drive is used as a single large file system.

FILES
/dev/hk[0-7][a-h] block files
/dev/rhk[0-7][a-h] raw files

SEE ALSO
hp(4), uda(4), up(4), syslogd(8)

DIAGNOSTICS
rk%d%c: hard error sn%d cs2=%b ds=%b er=%b. An unrecoverable error occurred during transfer of the specified sector of the specified disk partition. The contents of the cs2, ds and er registers are printed in octal and symbolically with bits decoded. The error was either unrecoverable, or a large number of retry attempts (including offset positioning and drive recalibration) could not recover the error.
rk%d: write locked. The write protect switch was set on the drive when a write was attempted. The write operation is not recoverable.

rk%d: not ready. The drive was spun down or offline when it was accessed. The i/o operation is not recoverable.

rk%d: not ready (came back!). The drive was not ready, but after printing the message about being not ready (which takes a fraction of a second) was ready. The operation is recovered if no further errors occur.

rk%d%c: soft ecc sn%d. A recoverable ECC error occurred on the specified sector in the specified disk partition. This happens normally a few times a week. If it happens more frequently than this the sectors where the errors are occurring should be checked to see if certain cylinders on the pack, spots on the carriage of the drive or heads are indicated.

hk%d: lost interrupt. A timer watching the controller detected no interrupt for an extended period while an operation was outstanding. This indicates a hardware or software failure. There is currently a hardware/software problem with spinning down drives while they are being accessed which causes this error to occur. The error causes a UNIBUS reset, and retry of the pending operations. If the controller continues to lose interrupts, this error will recur a few seconds later.

BUGS

In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

DEC-standard error logging should be supported.

A program to analyze the logged error information (even in its present reduced form) is needed.

The partition tables for the file systems should be read off of each pack, as they are never quite what any single installation would prefer, and this would make packs more portable.

The rk07g partition size in rk.c disagrees with that in /etc/disktab.
NAME
hp - MASSBUS disk interface

SYNOPSIS
disk hp0 at mba0 drive 0

DESCRIPTION
Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, etc. The standard device names begin with "hp" followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-7.

The block file's access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise seek calls should specify a multiple of 512 bytes.

DISK SUPPORT
This driver handles both standard DEC controllers and Emulex SC750 and SC780 controllers. Standard DEC drive types are recognized according to the MASSBUS drive type register. For the Emulex controller the drive type register should be configured to indicate the drive is an RM02. When this is encountered, the driver checks the holding register to find out the disk geometry and, based on this information, decides what the drive type is. The following disks are supported: RM03, RM05, RP06, RM80, RP05, RP07, ML11A, ML11B, CDC 9775, CDC 9730, AMPEX Capricorn (32 sectors/track), FUJITSU Eagle (48 sectors/track), Fujitsu 2361, and AMPEX 9300. The origin and size (in sectors) of the pseudo-disks on each drive are as follows:

RM03 partitions
<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
<th>cyls</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp?a</td>
<td>0</td>
<td>15884</td>
<td>0-99</td>
</tr>
<tr>
<td>hp?b</td>
<td>16000</td>
<td>33440</td>
<td>100-309</td>
</tr>
<tr>
<td>hp?c</td>
<td>0</td>
<td>131680</td>
<td>0-822</td>
</tr>
<tr>
<td>hp?d</td>
<td>49600</td>
<td>15884</td>
<td>309-408</td>
</tr>
<tr>
<td>hp?e</td>
<td>65440</td>
<td>55936</td>
<td>409-758</td>
</tr>
<tr>
<td>hp?f</td>
<td>121440</td>
<td>10080</td>
<td>759-822</td>
</tr>
<tr>
<td>hp?g</td>
<td>49600</td>
<td>82080</td>
<td>309-822</td>
</tr>
</tbody>
</table>

RM05 partitions
<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
<th>cyls</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp?a</td>
<td>0</td>
<td>15884</td>
<td>0-26</td>
</tr>
<tr>
<td>hp?b</td>
<td>16416</td>
<td>33440</td>
<td>27-81</td>
</tr>
<tr>
<td>hp?c</td>
<td>0</td>
<td>500384</td>
<td>0-822</td>
</tr>
<tr>
<td>hp?d</td>
<td>341696</td>
<td>15884</td>
<td>562-588</td>
</tr>
<tr>
<td>hp?e</td>
<td>358112</td>
<td>55936</td>
<td>589-680</td>
</tr>
<tr>
<td>hp?f</td>
<td>414048</td>
<td>86176</td>
<td>681-822</td>
</tr>
<tr>
<td>hp?g</td>
<td>341696</td>
<td>158528</td>
<td>562-822</td>
</tr>
<tr>
<td>hp?h</td>
<td>49856</td>
<td>291346</td>
<td>82-561</td>
</tr>
</tbody>
</table>

RP06 partitions
<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
<th>cyls</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp?a</td>
<td>0</td>
<td>15884</td>
<td>0-37</td>
</tr>
<tr>
<td>hp?b</td>
<td>15884</td>
<td>33440</td>
<td>38-117</td>
</tr>
<tr>
<td>Disk</td>
<td>Start</td>
<td>Length</td>
<td>Cyls</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>15884</td>
<td>0-36</td>
</tr>
<tr>
<td>b</td>
<td>16058</td>
<td>33440</td>
<td>37-114</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>242606</td>
<td>0-558</td>
</tr>
<tr>
<td>d</td>
<td>49910</td>
<td>15884</td>
<td>115-151</td>
</tr>
<tr>
<td>e</td>
<td>68096</td>
<td>55936</td>
<td>152-280</td>
</tr>
<tr>
<td>f</td>
<td>125888</td>
<td>120466</td>
<td>281-558</td>
</tr>
<tr>
<td>g</td>
<td>49910</td>
<td>192510</td>
<td>115-558</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disk</th>
<th>Start</th>
<th>Length</th>
<th>Cyls</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>15884</td>
<td>0-36</td>
</tr>
<tr>
<td>b</td>
<td>16000</td>
<td>66880</td>
<td>10-51</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>1008000</td>
<td>0-629</td>
</tr>
<tr>
<td>d</td>
<td>376000</td>
<td>15884</td>
<td>235-244</td>
</tr>
<tr>
<td>e</td>
<td>392000</td>
<td>307200</td>
<td>245-436</td>
</tr>
<tr>
<td>f</td>
<td>699200</td>
<td>308600</td>
<td>437-629</td>
</tr>
<tr>
<td>g</td>
<td>376000</td>
<td>631800</td>
<td>235-629</td>
</tr>
<tr>
<td>h</td>
<td>83200</td>
<td>291346</td>
<td>52-234</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disk</th>
<th>Start</th>
<th>Length</th>
<th>Cyls</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>15884</td>
<td>0-36</td>
</tr>
<tr>
<td>b</td>
<td>16000</td>
<td>33440</td>
<td>50-154</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>263360</td>
<td>0-822</td>
</tr>
<tr>
<td>d</td>
<td>49600</td>
<td>15884</td>
<td>155-204</td>
</tr>
<tr>
<td>e</td>
<td>65600</td>
<td>55936</td>
<td>205-379</td>
</tr>
<tr>
<td>f</td>
<td>121600</td>
<td>141600</td>
<td>380-822</td>
</tr>
</tbody>
</table>
It is unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter. The hp?a partition is normally used for the root file system, the hp?b partition as a paging area, and the hp?c partition for pack-pack copying (it maps the entire disk). On disks larger than about 205 Megabytes, the hp?h partition is inserted prior to the hp?d or hp?g partition; the hp?g partition then maps the remainder of the pack. All disk partition tables are calculated using the `diskpart(8)` program.

### AMPEX Capricorn partitions

<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
<th>cyls</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp?a</td>
<td>0</td>
<td>15884</td>
<td>0-31</td>
</tr>
<tr>
<td>hp?b</td>
<td>16384</td>
<td>33440</td>
<td>32-97</td>
</tr>
<tr>
<td>hp?c</td>
<td>524288</td>
<td>0-1023</td>
<td></td>
</tr>
<tr>
<td>hp?d</td>
<td>15884</td>
<td>668-699</td>
<td></td>
</tr>
<tr>
<td>hp?e</td>
<td>55936</td>
<td>700-809</td>
<td></td>
</tr>
<tr>
<td>hp?f</td>
<td>109408</td>
<td>810-1023</td>
<td></td>
</tr>
<tr>
<td>hp?g</td>
<td>182112</td>
<td>668-1023</td>
<td></td>
</tr>
</tbody>
</table>

### FUJITSU Eagle partitions

<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
<th>cyls</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp?a</td>
<td>0</td>
<td>15884</td>
<td>0-16</td>
</tr>
<tr>
<td>hp?b</td>
<td>16320</td>
<td>66880</td>
<td>17-86</td>
</tr>
<tr>
<td>hp?c</td>
<td>808320</td>
<td>0-841</td>
<td></td>
</tr>
<tr>
<td>hp?d</td>
<td>15884</td>
<td>391-407</td>
<td></td>
</tr>
<tr>
<td>hp?e</td>
<td>55936</td>
<td>408-727</td>
<td></td>
</tr>
<tr>
<td>hp?f</td>
<td>109248</td>
<td>728-841</td>
<td></td>
</tr>
<tr>
<td>hp?g</td>
<td>432768</td>
<td>391-841</td>
<td></td>
</tr>
</tbody>
</table>

### FUJITSU 2361 partitions

<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
<th>cyls</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp?a</td>
<td>0</td>
<td>15884</td>
<td>0-16</td>
</tr>
<tr>
<td>hp?b</td>
<td>16640</td>
<td>66880</td>
<td>13-65</td>
</tr>
<tr>
<td>hp?c</td>
<td>1077760</td>
<td>0-841</td>
<td></td>
</tr>
<tr>
<td>hp?d</td>
<td>15884</td>
<td>294-306</td>
<td></td>
</tr>
<tr>
<td>hp?e</td>
<td>307200</td>
<td>307-546</td>
<td></td>
</tr>
<tr>
<td>hp?f</td>
<td>377408</td>
<td>547-841</td>
<td></td>
</tr>
<tr>
<td>hp?g</td>
<td>701248</td>
<td>294-841</td>
<td></td>
</tr>
</tbody>
</table>

### AMPEX 9300 partitions

<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
<th>cyl</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp?a</td>
<td>0</td>
<td>15884</td>
<td>0-26</td>
</tr>
<tr>
<td>hp?b</td>
<td>16416</td>
<td>33440</td>
<td>27-81</td>
</tr>
<tr>
<td>hp?c</td>
<td>495520</td>
<td>0-814</td>
<td></td>
</tr>
<tr>
<td>hp?d</td>
<td>15884</td>
<td>562-588</td>
<td></td>
</tr>
<tr>
<td>hp?e</td>
<td>55936</td>
<td>589-680</td>
<td></td>
</tr>
<tr>
<td>hp?f</td>
<td>81312</td>
<td>681-814</td>
<td></td>
</tr>
<tr>
<td>hp?g</td>
<td>153664</td>
<td>562-814</td>
<td></td>
</tr>
<tr>
<td>hp?h</td>
<td>291346</td>
<td>82-561</td>
<td></td>
</tr>
</tbody>
</table>

FILES

/\dev/hp[0-7][a-h] block files
/\dev/rhp[0-7][a-h] raw files
SEE ALSO
hk(4), uda(4), up(4)

DIAGNOSTICS
hp%d%c:: hard error sn%d mbsr=%b er1=%b er2=%b. An unrecoverable error occurred during transfer of the specified sector of the specified disk partition. The MASSBUS status register is printed in hexadecimal and with the error bits decoded if any error bits other than MBEXC and DTABT are set. In any case the contents of the two error registers are also printed in octal and symbolically with bits decoded. (Note that er2 is what old rp06 manuals would call er3; the terminology is that of the rm disks). The error was either unrecoverable, or a large number of retry attempts (including offset positioning and drive recalibration) could not recover the error.

hp%d: write locked. The write protect switch was set on the drive when a write was attempted. The write operation is not recoverable.

hp%d: not ready. The drive was spun down or off line when it was accessed. The I/O operation is not recoverable.

hp%d%c:: soft eec sn%d. A recoverable ECC error occurred on the specified sector of the specified disk partition. This happens normally a few times a week. If it happens more frequently than this the sectors where the errors are occurring should be checked to see if certain cylinders on the pack, spots on the carriage of the drive or heads are indicated.

During autoconfiguration one of the following messages may appear on the console indicating the appropriate drive type was recognized. The last message indicates the drive is of an unknown type.

hp%d: 9775 (direct).
hp%d: 9730 (direct).
hp%d: 9300.
hp%d: 9762.
hp%d: capricorn.
hp%d: eagle.
hp%d: 2361.
hp%d: ntracks %d, nsectors %d: unknown device.

BUGS
In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and /seek(2) should always deal in 512-byte multiples.

DEC-standard error logging should be supported.

A program to analyze the logged error information (even in its present reduced form) is needed.

The partition tables for the file systems should be read off of each pack, as they are never quite what any single installation would prefer, and this would make packs more portable.
NAME
ht - TM-03/TE-16,TU-45,TU-77 MASSBUS magtape interface

SYNOPSIS
master ht0 at mba? drive ?
tape tu0 at ht0 slave 0

DESCRIPTION
The tm-03/transport combination provides a standard tape drive interface as described in
mtio(4). All drives provide both 800 and 1600 bpi; the TE-16 runs at 45 ips, the TU-45 at 75
ips, while the TU-77 runs at 125 ips and autoloads tapes.

SEE ALSO
mt(1), tar(1), tp(1), mtio(4), tm(4), ts(4), mt(4), ut(4)

DIAGNOSTICS

tu%d: no write ring. An attempt was made to write on the tape drive when no write ring was
present; this message is written on the terminal of the user who tried to access the tape.

tu%d: not online. An attempt was made to access the tape while it was offline; this message is
written on the terminal of the user who tried to access the tape.

tu%d: can't change density in mid-tape. An attempt was made to write on a tape at a different
density than is already recorded on the tape. This message is written on the terminal of the
user who tried to switch the density.


tu%d: hard error bn%d mbsr=%b er=%b ds=%b. A tape error occurred at block bn; the ht
error register and drive status register are printed in octal with the bits symbolically decoded.
Any error is fatal on non-raw tape; when possible the driver will have retried the operation
which failed several times before reporting the error.

BUGS
If any non-data error is encountered on non-raw tape, it refuses to do anything more until
closed.
NAME

hy – Network Systems Hyperchannel interface

SYNOPSIS

device hy0 at uba0 csr 0172410 vector hyint

DESCRIPTION

The hy interface provides access to a Network Systems Corporation Hyperchannel Adapter.

The network to which the interface is attached is specified at boot time with an SIOCSI-
FADDR ioctl. The host's address is discovered by reading the adapter status register. The
interface will not transmit or receive packets until the network number is known.

DIAGNOSTICS

hy%d: unit number 0x%x port %d type %x microcode level 0x%x. Identifies the device during
autoconfiguration.

hy%d: can't handle af%d. The interface was handed a message with addresses formatted in an
unsuitable address family; the packet was dropped.

hy%d: can't initialize. The interface was unable to allocate UNIBUS resources. This is usually
due to having too many network devices on an 11/750 where there are only 3 buffered data
paths.

hy%d: NEX - Non Existent Memory. Non existent memory error returned from hardware.

hy%d: BAR overflow. Bus address register overflow error returned from hardware.

hy%d: Power Off bit set, trying to reset. Adapter has lost power, driver will reset the bit and
see if power is still out in the adapter.

hy%d: Power Off Error, network shutdown. Power was really off in the adapter, network con-
nexions are dropped. Software does not shut down the network unless power has been off for
a while.

hy%d: RECVD MP > MPSIZE (%d). A message proper was received that is too big. Prob-
able a driver bug. Shouldn't happen.

hy%d: xmit error - len > hy_olen [%d > %d]. Probable driver error. Shouldn't happen.

hy%d: DRIVER BUG - INVALID STATE %d. The driver state machine reached a non-
existent state. Definite driver bug.

hy%d: watchdog timer expired. A command in the adapter has taken too long to complete.
Driver will abort and retry the command.

hy%d: adapter power restored. Software was able to reset the power off bit, indicating that the
power has been restored.

SEE ALSO

intro(4N), inet(4F)

BUGS

If the adapter does not respond to the status command issued during autoconfigure, the
adapter is assumed down. A reboot is required to recognize it.

The adapter power fail interrupt seems to occur sporadically when power has, in fact, not
failed. The driver will believe that power has failed only if it can not reset the power fail
latch after a “reasonable” time interval. These seem to appear about 2-4 times a day on some
machines. There seems to be no correlation with adapter rev level, number of ports used etc.
and whether a machine will get these “bogus powerfails”. They don't seem to cause any real
problems so they have been ignored.
NAME
icmp - Internet Control Message Protocol

SYNOPSIS
#include <sys/socket.h>
#include <netinet/in.h>
s = socket(AF_INET, SOCK_RAW, proto);

DESCRIPTION
ICMP is the error and control message protocol used by IP and the Internet protocol family. It may be accessed through a "raw socket" for network monitoring and diagnostic functions. The proto parameter to the socket call to create an ICMP socket is obtained from getprotobyname(3N). ICMP sockets are connectionless, and are normally used with the sendto and recvfrom calls, though the connect(2) call may also be used to fix the destination for future packets (in which case the read(2) or recv(2) and write(2) or send(2) system calls may be used).

Outgoing packets automatically have an IP header prepended to them (based on the destination address). Incoming packets are received with the IP header and options intact.

DIAGNOSTICS
A socket operation may fail with one of the following errors returned:

[EISCONN] when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;

[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;

[ENOBUFS] when the system runs out of memory for an internal data structure;

[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO
send(2), recv(2), intro(4N), inet(4F), ip(4P)
NAME

idp - Xerox Internet Datagram Protocol

SYNOPSIS

```c
#include <sys/socket.h>
#include <netas/ns.h>
#include <netns/idp.h>

s = socket(AF_NS, SOCK_DGRAM, 0);
```

DESCRIPTION

IDP is a simple, unreliable datagram protocol which is used to support the SOCK_DGRAM abstraction for the Internet protocol family. IDP sockets are connectionless, and are normally used with the `sendto` and `recvfrom` calls, though the `connect(2)` call may also be used to fix the destination for future packets (in which case the `recv(2)` or `read(2)` and `send(2)` or `write(2)` system calls may be used).

Xerox protocols are built vertically on top of IDP. Thus, IDP address formats are identical to those used by SPP. Note that the IDP port space is the same as the SPP port space (i.e. a IDP port may be "connected" to a SPP port, with certain options enabled below). In addition broadcast packets may be sent (assuming the underlying network supports this) by using a reserved "broadcast address"; this address is network interface dependent.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

- `[EISCONN]` when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
- `[ENOTCONN]` when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
- `[ENOBUFS]` when the system runs out of memory for an internal data structure;
- `[EADDRINUSE]` when an attempt is made to create a socket with a port which has already been allocated;
- `[EADDRNOTAVAIL]` when an attempt is made to create a socket with a network address for which no network interface exists.

SOCKET OPTIONS

**[SO_HEADERS_ON_INPUT]**

When set, the first 30 bytes of any data returned from a read or recv from will be the initial 30 bytes of the IDP packet, as described by

```c
struct idp {
    u_short idp_sum;
    u_short idp_len;
    u_char idp_tc;
    u_char idp_pt;
    struct ns_addr idp_dna;
    struct ns_addr idp_sna;
};
```

This allows the user to determine the packet type, and whether the packet was a multi-cast packet or directed specifically at the local host. When requested, gives the current state of the option, (NSP_RAWIN or 0).

**[SO_HEADERS_ON_OUTPUT]**

When set, the first 30 bytes of any data sent will be the initial 30 bytes of the
IDP packet. This allows the user to determine the packet type, and whether the packet should be multi-cast packet or directed specifically at the local host. You can also misrepresent the sender of the packet. When requested, gives the current state of the option. (NSP_RAWOUT or 0).

[SO_DEFAULT_HEADERS]
The user provides the kernel an IDP header, from which it gleans the Packet Type. When requested, the kernel will provide an IDP header, showing the default packet type, and local and foreign addresses, if connected.

[SO_ALL_PACKETS]
When set, this option defeats automatic processing of Error packets, and Sequence Protocol packets.

[SO_SEQNO] When requested, this returns a sequence number which is not likely to be repeated until the machine crashes or a very long time has passed. It is useful in constructing Packet Exchange Protocol packets.

SEE ALSO
send(2), recv(2), intro(4N), ns(4F)
NAME

ik - Ikonas frame buffer, graphics device interface

SYNOPSIS

device ik0 at uba? csr 0172460 vector ikintr

DESCRIPTION

Ik provides an interface to an Ikonas frame buffer graphics device. Each minor device is a
different frame buffer interface board. When the device is opened, its interface registers are
mapped, via virtual memory, into the user processes address space. This allows the user pro­
cess very high bandwidth to the frame buffer with no system call overhead.

Bytes written or read from the device are DMA'ed from or to the interface. The frame buffer
XY address, its addressing mode, etc. must be set up by the user process before calling write
or read.

Other communication with the driver is via ioctls. The IK_GETADDR ioctl returns the vir­
tual address where the user process can find the interface registers. The IK_WAITINT ioctl
suspending the user process until the ikonas device has interrupted (for whatever reason — the
user process has to set the interrupt enables).

FILES

/dev/ik

DIAGNOSTICS

None.

BUGS

An invalid access (e.g., longword) to a mapped interface register can cause the system to crash
with a machine check. A user process could possibly cause infinite interrupts hence bringing
things to a crawl.
NAME
il – Interlan NI1010 10 Mb/s Ethernet interface

SYNOPSIS
device il0 at uba0 csr 164000 vector ilrint ilcint

DESCRIPTION
The il interface provides access to a 10 Mb/s Ethernet network through an Interlan 1010 or 1010A controller.

Each of the host's network addresses is specified at boot time with an SIOCSIFADDR ioctl. The il interface employs the address resolution protocol described in arp(4P) to dynamically map between Internet and Ethernet addresses on the local network.

The interface normally tries to use a "trailer" encapsulation to minimize copying data on input and output. The use of trailers is negotiated with ARP. This negotiation may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS
il%d: input error. The hardware indicated an error in reading a packet off the cable or an illegally sized packet.

il%d: can't handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

il%d: setaddr didn't work. The interface was unable to reprogram its physical ethernet address. This may happen with very early models of the interface. This facility is used only when the controller is not the first network interface configured for XNS. The oldest interface tested (2.7.1.0.1.45) has never failed in this way.

The following messages indicate a probable hardware error performing the indicated operation during autoconfiguration or initialization. The status field in the control and status register (the low-order four bits) should indicate the nature of the failure. See the hardware manual for details.

il%d: reset failed, csr=%b.

il%d: status failed, csr=%b.

il%d: hardware diag failed, csr=%b.

il%d: verifying setaddr, csr=%b.

il%d: stray xmit interrupt, csr=%b.

il%d: can't initialize.

SEE ALSO
intro(4N), inet(4F), arp(4P)
NAME
imp - 1822 network interface

SYNOPSIS
pseudo-device imp [ count ]

DESCRIPTION
The imp interface, as described in BBN Report 1822, provides access to an intelligent
message processor normally used when participating in the Department of Defense ARPA
network. The network interface communicates through a device controller, usually an ACC
LH/DH or HDH or a DEC IMP-11A, with the IMP. The interface is “reliable” and “flow-
controlled” by the host-IMP protocol.

To configure IMP support, at least one of acc(4), css(4) or hdh(4) must be included. The
optional count specifies the total number of IMP connections. The network number on which
the interface resides is specified at boot time using the SIOCSIFADDR ioctl. The host
number is discovered through receipt of NOOP messages from the IMP.

The network interface is always in one of four states: up, down, initializing, or going down.
When the system is booted, the interface is marked down. If the hardware controller is
successfully probed, the interface enters the initializing state and transmits three NOOP messages
to the IMP. It then waits for the IMP to respond with two or more NOOP messages in reply.
When it receives these messages it enters the up state. The “going down” state is entered
only when notified by the IMP of an impending shutdown. Packets may be sent through the
interface only while it is in the up state. Outgoing packets are dropped with the error ENET-
DOWN returned to the caller if the interface is in any other state.

DIAGNOSTICS
imp%d: not configured. A hardware interface could not be attached during autoconfiguration
because too few IMP-pseudo-devices were configured.

imp%d: leader error. The IMP reported an error in a leader (1822 message header). This
causes the interface to be reset and any packets queued up for transmission to be purged.

imp%d: going down in 30 seconds.
imp%d: going down for hardware PM.
imp%d: going down for reload software.
imp%d: going down for emergency reset. The Network Control Center (NCC) is manipulating
the IMP. By convention these messages are reported to all hosts on an IMP.

imp?: host %x, lost %d rfms. The IMP had messages outstanding to the host listed, but no
RFNM (Request for Next Message) messages were received from the IMP in 127 seconds.
The software state for that host is reinitialized.

imp%d: interface reset. The host has received an interface reset message from the IMP.

imp%d: address reset to x%x (%d/%d). The host has received a NOOP message which caused
it to reset its notion of its current address. The Internet address is printed in hexadecimal,
with the host and IMP numbers following. This indicates that the address originally set by
ifconfig(8) was incorrect, that the IMP has undergone an identity crisis, or that communica-
tion between the IMP and the host is being garbled.

imp%d: data error. The IMP noted an error in data transmitted. The host-IMP interface is
reset and the host enters the init state (awaiting NOOP messages).

imp%d: interface reset. The reset process has been completed.

imp%d: marked down. After receiving a “going down in 30 seconds” message, and waiting 30
seconds, the host has marked the IMP unavailable. Before packets may be sent to the IMP
again, the IMP must notify the host, through a series of NOOP messages, that it is back up.
imp%od: can't handle af%od. The interface was handed a message with addresses formatting in an unsuitable address family; the packet was dropped.

SEE ALSO
intro(4N), inet(4F), acc(4), css(4), hdh(4), implog(8), implogd(8)
NAME
imp – IMP raw socket interface

SYNOPSIS
#include <sys/socket.h>
#include <netinet/in.h>
#include <netimp/iCimp.h>
s = socket(AF_IMPUNK, SOCK_RAW, proto);

DESCRIPTION
The raw imp socket provides direct access to the imp(4) network interface. Users send packets through the interface using the send(2) calls, and receive packets with the recv(2), calls. All outgoing packets must have an 182296-bit leader on the front. Likewise, packets received by the user will have this leader on the front. The 1822 leader and the legal values for the various fields are defined in the include file <netimp/if_imp.h>. The raw imp interface automatically installs the length and destination address in the 1822 leader of all outgoing packets; these need not be filled in by the user.

If the protocol selected, proto, is zero, the socket will receive all IMP messages except RFNM and incompleted which are not input data for a kernel protocol. If proto is non-zero, only messages for the specified link type will be received.

DIAGNOSTICS
An operation on a socket may fail with one of the following errors:

[EISCONN] when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;

[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket hasn’t been connected;

[ENOBUFFS] when the system runs out of memory for an internal data structure;

[ENOBUFFS] eight messages to the destination host are outstanding, and another eight are already queued for output;

[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO
intro(4N), inet(4F), imp(4)
NAME
inet - Internet protocol family

SYNOPSIS
#include <sys/types.h>
#include <netinet/in.h>

DESCRIPTION
The Internet protocol family is a collection of protocols layered atop the Internet Protocol (IP) transport layer, and utilizing the Internet address format. The Internet family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types; the SOCK_RAW interface provides access to the IP protocol.

ADDRESSING
Internet addresses are four byte quantities, stored in network standard format (on the VAX these are word and byte reversed). The include file <netinet/in.h> defines this address as a discriminated union.

Sockets bound to the Internet protocol family utilize the following addressing structure,

struct sockaddr_in {
    short sin_family;
    u_short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};

Sockets may be created with the local address INADDR_ANY to effect "wildcard" matching on incoming messages. The address in a connect(2) or sendto(2) call may be given as INADDR_ANY to mean "this host." The distinguished address INADDR_BROADCAST is allowed as a shorthand for the broadcast address on the primary network if the first network configured supports broadcast.

PROTOCOLS
The Internet protocol family is comprised of the IP transport protocol, Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). TCP is used to support the SOCK_STREAM abstraction while UDP is used to support the SOCK_DGRAM abstraction. A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The ICMP message protocol is accessible from a raw socket.

The 32-bit Internet address contains both network and host parts. It is frequency-encoded; the most-significant bit is clear in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses use the high-order 16 bits as the network field, and Class C addresses have a 24-bit network part. Sites with a cluster of local networks and a connection to the DARPA Internet may chose to use a single network number for the cluster; this is done by using subnet addressing. The local (host) portion of the address is further subdivided into subnet and host parts. Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following ioctl(2) commands on a datagram socket in the Internet domain; they have the same form as the SIOCIFADDR command (see intro(4N)).

SIOCSIFNETMASK Set interface network mask. The network mask defines the network part of the address; if it contains more of the address than the address type would indicate, then subnets are in use.

SIOCGIFNETMASK Get interface network mask.
SEE ALSO
   ioctl(2), socket(2), intro(4N), tcp(4P), udp(4P), ip(4P), icmp(4P)
   An Introductory 4.3BSD Interprocess Communication Tutorial (PS1:7).
   An Advanced 4.3BSD Interprocess Communication Tutorial (PS1:8).

CAVEAT
   The Internet protocol support is subject to change as the Internet protocols develop. Users
   should not depend on details of the current implementation, but rather the services exported.
NAME
ip – Internet Protocol

SYNOPSIS
#include <sys/socket.h>
#include <netinet/in.h>
s = socket(AF_INET, SOCK_RAW, proto);

DESCRIPTION
IP is the transport layer protocol used by the Internet protocol family. Options may be set at the IP level when using higher-level protocols that are based on IP (such as TCP and UDP). It may also be accessed through a “raw socket” when developing new protocols, or special purpose applications.

A single generic option is supported at the IP level, IP_OPTIONS, that may be used to provide IP options to be transmitted in the IP header of each outgoing packet. Options are set with setsockopt(2) and examined with getsockopt(2). The format of IP options to be sent is that specified by the IP protocol specification, with one exception: the list of addresses for Source Route options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.

Raw IP sockets are connectionless, and are normally used with the sendto and recvfrom calls, though the connect(2) call may also be used to fix the destination for future packets (in which case the read(2) or recv(2) and write(2) or send(2) system calls may be used).

If proto is 0, the default protocol IPPROTO_RAW is used for outgoing packets, and only incoming packets destined for that protocol are received. If proto is non-zero, that protocol number will be used on outgoing packets and to filter incoming packets.

Outgoing packets automatically have an IP header prepended to them (based on the destination address and the protocol number the socket is created with). Incoming packets are received with IP header and options intact.

DIAGNOSTICS
A socket operation may fail with one of the following errors returned:
[EISCONN] when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket hasn’t been connected;
[ENOBUFS] when the system runs out of memory for an internal data structure;
[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for which no network interface exists.

The following errors specific to IP may occur when setting or getting IP options:
[EINVAL] An unknown socket option name was given.
[EINVAL] The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

SEE ALSO
getsockopt(2), send(2), recv(2), intro(4N), icmp(4P), inet(4F)
NAME
ix - Interlan Np100 10 Mb/s Ethernet interface

SYNOPSIS
device np0 at uba0 csr 166000 vector npintr

DESCRIPTION
The ix interface provides access to a 10 Mb/s Ethernet network through an Interlan Np100 controller used as a link-layer interface.

This interface is unusual in that it requires loading firmware into the controller before it may be used as a network interface. This is accomplished by opening a character special device, and writing data to it. A program to load the image is provided in /usr/src/new/np100. The sequence of commands would be:

```
# /npload np.image [/dev/np<board #> if other than np00]
# sleep 10
# ifconfig ix0 ...
```

Each of the host’s network addresses is specified at boot time with an SIOCSIFADDR ioctl.

The ix interface employs the address resolution protocol described in arp(4P) to dynamically map between Internet and Ethernet addresses on the local network.

The interface normally tries to use a “trailer” encapsulation to minimize copying data on input and output. The use of trailers is negotiated with ARP. This negotiation may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCFLAGS ioctl.

DIAGNOSTICS
ix%d: Req failed, cmd %x, stat %x, ust error %x,%x. The firmware in the controller refused to honor a request from in initializing packet level communications. The board may need to be reset and reloaded. Or, you may not have allowed enough time between loading the board and issuing the request to begin unix network operation.

ix%d: can’t initialize. The interface was unable to obtain unibus resources required for operation.

ix%d: failed to reinitialize DLA module. The interface got sick after attempting to reprogram its physical ethernet address. Try reloading the firmware. The attempt is made only when this interfaces is not the first one configured for XNS.

ix%d: can’t handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

ix%d: stray xmit interrupt, npreq=%x. This may happen if the board is reloaded while network processes are still running.

ixrint: cqe error %x, %x, %x. This will result if an ifconfig(8c) request is made at an inopportune time, such as not allowing enough time after loading the firmware. After 100 such errors are logged, the unix network driver will shut itself down, saying:

ixrint: shutting down unix dla. The recourse is to reload the firmware and allow more time.

SEE ALSO
intro(4N), inet(4F), arp(4P), np(4).
NAME
kg – KL-11/DL-11W line clock

SYNOPSIS
device kg0 at uba0 csr 0176500 vector kglock

DESCRIPTION
A kl-11 or dl-11w can be used as an alternate real time clock source. When configured, certain system statistics and, optionally, system profiling work will be collected each time the clock interrupts. For optimum accuracy in profiling, the dl-11w should be configured to interrupt at the highest possible priority level. The kg device driver automatically calibrates itself to the line clock frequency.

SEE ALSO
kgmon(8), config(8)
NAME
lo – software loopback network interface

SYNOPSIS
pseudo-device loop

DESCRIPTION
The loop interface is a software loopback mechanism which may be used for performance
analysis, software testing, and/or local communication. As with other network interfaces, the
loopback interface must have network addresses assigned for each address family with which
it is to be used. These addresses may be set or changed with the SIOCSIFADDR ioctl. The
loopback interface should be the last interface configured, as protocols may use the order of
configuration as an indication of priority. The loopback should never be configured first
unless no hardware interfaces exist.

DIAGNOSTICS
lo%d: can't handle af%d. The interface was handed a message with addresses formatted in an
unsuitable address family; the packet was dropped.

SEE ALSO
intro(4N), inet(4F), ns(4F)

BUGS
Previous versions of the system enabled the loopback interface automatically, using a non-
standard Internet address (127.1). Use of that address is now discouraged; a reserved host
address for the local network should be used instead.
NAME
lp – line printer

SYNOPSIS
device lp0 at uba0 csr 0177514 vector lpintr

DESCRIPTION
   *lp provides the interface to any of the standard DEC line printers on an LP-11 parallel interface. When it is opened or closed, a suitable number of page ejects is generated. Bytes written are printed.

   The unit number of the printer is specified by the minor device after removing the low 3 bits, which act as per-device parameters. Currently only the lowest of the low three bits is interpreted: if it is set, the device is treated as having a 64-character set, rather than a full 96-character set. In the resulting half-ASCII mode, lower case letters are turned into upper case and certain characters are escaped according to the following table:

   \[
   \begin{array}{|c|c|}
   \hline
   & \text{\textbackslash{~}} \\
   \hline
   \text{\textbackslash{,}} & . \\
   \hline
   \text{\textbackslash{\|}} & \text{\textbackslash{\|}} \\
   \text{\textbackslash{\&}} & \text{\textbackslash{\&}} \\
   \hline
   \end{array}
   \]

   The driver correctly interprets carriage returns, backspaces, tabs, and form feeds. Lines longer than the maximum page width are truncated. The default page width is 132 columns. This may be overridden by specifying, for example, "flags 256".

FILES
/dev/lp

SEE ALSO
lpr(1)

DIAGNOSTICS
None.
NAME
  mem, kmem – main memory

DESCRIPTION
  Mem is a special file that is an image of the main memory of the computer. It may be used, for example, to examine (and even to patch) the system.

  Byte addresses in mem are interpreted as physical memory addresses. References to non-existent locations cause errors to be returned.

  The file kmem is the same as mem except that kernel virtual memory rather than physical memory is accessed. Only kernel virtual addresses that are mapped to memory are allowed. The file kUmem also refers to kernel virtual memory, but may be used to access areas mapped to UNIBUS address space and other I/O areas. It forces all accesses to use word (short integer) accesses. Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

  On VAX 11/780 the I/O space begins at physical address 20000000(16); on an 11/750 I/O space addresses are of the form fxxxxx(16). On all VAX’en per-process data for the current process is UPAGES long, and ends at virtual address 80000000(16).

FILES
  /dev/mem
  /dev/kmem
  /dev/kUmem
NAME
   mt – TM78/TU-78 MASSBUS magtape interface

SYNOPSIS
   master mt0 at mba? drive ?
   tape mu0 at mt0 slave 0

DESCRIPTION
   The tm78/tu-78 combination provides a standard tape drive interface as described in mtio(4).
   Only 1600 and 6250 bpi are supported; the TU-78 runs at 125 ips and autoloads tapes.

SEE ALSO
   mt(1), tar(1), tp(1), mtio(4), tm(4), ts(4), ut(4)

DIAGNOSTICS
   mu%d: no write ring. An attempt was made to write on the tape drive when no write ring was
   present; this message is written on the terminal of the user who tried to access the tape.
   mu%d: not online. An attempt was made to access the tape while it was offline; this message
   is written on the terminal of the user who tried to access the tape.
   mu%d: can't change density in mid-tape. An attempt was made to write on a tape at a
different density than is already recorded on the tape. This message is written on the terminal
of the user who tried to switch the density.
   mu%d: hard error bn%d mbsr=+%b er=+%x ds=+%b. A tape error occurred at block bn; the mt
error register and drive status register are printed in octal with the bits symbolically decoded.
Any error is fatal on non-raw tape; when possible the driver will have retried the operation
which failed several times before reporting the error.
   mu%d: blank tape. An attempt was made to read a blank tape (a tape without even end-of-file
marks).
   mu%d: offline. During an i/o operation the device was set offline. If a non-raw tape was used
in the access it is closed.

BUGS
   If any non-data error is encountered on non-raw tape, it refuses to do anything more until
closed.

   Because 800 bpi tapes are not supported, the numbering of minor devices is inconsistent with
   triple-density tape units. Unit 0 is drive 0, 1600 bpi.
NAME
mtio - UNIX magtape interface

DESCRIPTION
The files mt0, ..., mt15 refer to the UNIX magtape drives, which may be on the MASSBUS using the TM03 formatter ht(4), or TM78 formatter, mt(4), or on the UNIBUS using either the TM11 or TS11 formatters tm(4), TU45 compatible formatters, ut(4), or ts(4). The following description applies to any of the transport/controller pairs. The files mt0, ..., mt7 are 800bpi (or the transport’s lowest density), mt8, ..., mt15 are 1600bpi (or the transport’s second density), and mt16, ..., mt23 are 6250bpi (or the transport’s third density). (But note that only 1600 bpi is available with the TS11.) The files mt0, ..., mt3, mt8, ..., mt11, and mt16, ..., mt19 are rewound when closed; the others are not. When a file open for writing is closed, two end-of-files are written. If the tape is not to be rewound it is positioned with the head between the two tapemarks.

A standard tape consists of a series of 1024 byte records terminated by an end-of-file. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it uses most of the tape in record gaps.

The mt files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the ‘raw’ interface is appropriate. The associated files are named rmt0, ..., rmt23, but the same minor-device considerations as for the regular files still apply. A number of other ioctl operations are available on raw magnetic tape. The following definitions are from <sys/mtio.h>:

/*
 * Structures and definitions for mag tape io control commands
 */

/* structure for MTIOCTOP - mag tape op command */
struct mtop {
    short mt_op;      /* operations defined below */
    daddr_tmt_count;  /* how many of them */
};

/* operations */
#define MTWEOF 0      /* write an end-of-file record */
#define MTFSF 1       /* forward space file */
#define MTBSF 2       /* backward space file */
#define MTFSR 3       /* forward space record */
#define MTBSR 4       /* backward space record */
#define MTREW 5       /* rewind */
#define MTOFFFL 6     /* rewind and put the drive offline */
#define MTNOP 7       /* no operation, sets status only */
#define MTCACHE 8     /* enable controller cache */
#define MTNOCACHE 9   /* disable controller cache */

/* structure for MTIOCGET - mag tape get status command */
struct mtget {
    short mt_type;   /* type of magtape device */
    /* the following two registers are grossly device dependent */
    short mt_dsreg;  /* “drive status” register */
    short mt_erreg;  /* “error” register */

4.2 Berkeley Distribution May 16, 1986
FILES
/dev/mt?
/dev/rmt?

SEE ALSO
mt(1), tar(1), tp(1), ht(4), tm(4), ts(4), mt(4), ut(4)

BUGS
The status should be returned in a device independent format.
The special file naming should be redone in a more consistent and understandable manner.
NAME

np - Interlan Np100 10 Mb/s Ethernet interface

SYNOPSIS

device np0 at uba0 csr 166000 vector npintr

DESCRIPTION

The Np device provides access to an Interlan Np100 Ethernet interface for control functions. This interface is unusual in that it requires loading firmware into the controller before it may be used as a network link-level interface. This is accomplished by opening a character special device, and writing data to it. It is also possible to do post-mortem debugging of firmware failures by reading the local memory of the device.

A program to load the image is provided in /usr/src/new/np100. The sequence of commands would be:

```
# ./npload np.image [/dev/np00]
# sleep 10
# ifconfig ix0 ...
```

Multiple control processes are allowed by opening separate minor devices; secondary interfaces are specified by shifting the interface number by 4 bits.

The device also responds to commands passed through the driver by the following ioctls:

NPRESET
kils off all active network processes.

NPSTART
begins execution of the board at the specified address (usually Ox400).

NPNETBOOT
downloads the image from a server on the network. [Contact MICOM-INTERLAN for details.]

DIAGNOSTICS

np%d: Bad Maintenance command: %x! An invalid ioctl was passed to the np driver.

np%d: Panic Np100 bad buffer chain. An error occurred in an read or write operation causing it to run out of buffers before it finished the operation. This indicates a kernel failure rather than a device failure.

NP100 unit %d not found! A failure occurred during initialization, such that the unibus address expected for the board was found to be bad. Probably indicates hardware problems with the board, as do the following:

NP100 Unit %d timed out!

NP100 Unit %d Failed diagnostics!
Status from CSR0: %x.

Panic from NP100 unit %d\nPanic Message: %s. An occurrence on the board was deemed serious enough to have the vax print it out.

NP100 unit #%d available!. The board was successfully loaded and started.

np%d: Bad Req: %x.. The board made a maintenance request to the vax that it did not understand.
np%d: No more room on Command Queue!. The np driver allowed an internal resource to be exhausted. This should never happen.

There are 110 other diagnostic messages that can be enabled by setting bits in a debugging mask. Consult the driver for details.

SEE ALSO
intro(4N), inet(4F), arp(4P), ix(4)
NAME
ns – Xerox Network Systems(tm) protocol family

SYNOPSIS
options NS
options NSIP
pseudo-device ns

DESCRIPTION
The NS protocol family is a collection of protocols layered atop the Internet Datagram Protocol (IDP) transport layer, and using the Xerox NS address formats. The NS family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, SOCK_SEQPACKET, and SOCK_RAW socket types; the SOCK_RAW interface is a debugging tool, allowing you to trace all packets entering, (or with toggling kernel variable, additionally leaving) the local host.

ADDRESSING
NS addresses are 12 byte quantities, consisting of a 4 byte Network number, a 6 byte Host number and a 2 byte port number, all stored in network standard format. (on the VAX these are word and byte reversed; on the Sun they are not reversed). The include file <netns/ns.h> defines the NS address as a structure containing unions (for quicker comparisons).

Sockets in the Internet protocol family use the following addressing structure:

```
struct sockaddr_ns {
    short sns_family;
    struct ns_addr sns_addr;
    char sns_zero[2];
};
```

where an ns_addr is composed as follows:

```
union ns_host {
    u_char c_host[6];
    u_short s_host[3];
};
union ns_net {
    u_char c_net[4];
    u_short s_net[2];
};
struct ns_addr {
    union ns_net x_net;
    union ns_host x_host;
    u_short x_port;
};
```

Sockets may be created with an address of all zeroes to effect "wildcard" matching on incoming messages. The local port address specified in a bind(2) call is restricted to be greater than NSPORT_RESERVED (=3000, in <netns/ns.h>) unless the creating process is running as the super-user, providing a space of protected port numbers.

PROTOCOLS
The NS protocol family supported by the operating system is comprised of the Internet Datagram Protocol (IDP) idp(4P), Error Protocol (available through IDP), and Sequenced Packet Protocol (SPP) spp(4P).
SPP is used to support the SOCK_STREAM and SOCK_SEQPACKET abstraction, while IDP is used to support the SOCK_DGRAM abstraction. The Error protocol is responded to by the kernel to handle and report errors in protocol processing; it is, however, only accessible to user programs through heroic actions.

SEE ALSO
intro(3), byteorder(3N), gethostbyname(3N), getnetent(3N), getprotoent(3N), getservent(3N), ns(3N), intro(4N), spp(4P), idp(4P), nsip(4)
Internet Transport Protocols, Xerox Corporation document XSIS-028112
An Advanced 4.3BSD Interprocess Communication Tutorial
NAME
nsip – software network interface encapsulating ns packets in ip packets.

SYNOPSIS
options NSIP
#include <netns/ns_if.h>

DESCRIPTION
The nsip interface is a software mechanism which may be used to transmit Xerox NS(tm) packets through otherwise uncooperative networks. It functions by prepending an IP header, and resubmitting the packet through the unix IP machinery.

The super-user can advise the operating system of a willing partner by naming an IP address to be associated with an NS address. Presently, only specific hosts pairs are allowed, and for each host pair, an artificial point-to-point interface is constructed. At some future date, IP broadcast addresses or hosts may be paired with NS networks or hosts.

Specifically, a socket option of SO_NSIP_ROUTE is set on a socket of family AF_NS, type SOCK_DGRAM, passing the following structure:

```c
struct nsip_req {
    struct sockaddr rq_ns; /* must be ns format destination */
    struct sockaddr rq_ip; /* must be ip format gateway */
    short rq_flags;
};
```

DIAGNOSTICS
nsip%ci: can’t handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

SEE ALSO
intro(4N), ns(4F)

BUGS
It is absurd to have a separate pseudo-device for each pt-to-pt link. There is no way to change the IP address for an NS host once the the encapsulation interface is set up. The request should honor flags of RTF_GATEWAY to indicate remote networks, and the absence of RTF_UP should be a clue to remove that partner. This was intended to postpone the necessity of rewriting reverse ARP for the en device, and to allow passing XNS packets through an Arpanet-Milnet gateway, to facilitate testing between some co-operating universities.
NAME
null – data sink

DESCRIPTION
Data written on a null special file is discarded.
Reads from a null special file always return 0 bytes.

FILES
/dev/null
NAME
pel - DEC CSS PCL-11 B Network Interface

SYNOPSIS
device pcl0 at uba? csr 164200 vector pclxint pclrint

DESCRIPTION
The pcl device provides an IP-only interface to the DEC CSS PCL-11 time division multiplexed network bus. The controller itself is not accessible to users.

The host's address is specified with the SIOCSIFADDR ioctl. The interface will not transmit or receive any data before its address is defined.

As the PCL-11 hardware is only capable of having 15 interfaces per network, a single-byte host-on-network number is used, with range [1..15] to match the TDM bus addresses of the interfaces.

The interface currently only supports the Internet protocol family and only provides "natural" (header) encapsulation.

DIAGNOSTICS
pcl%d: can't init. Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).

pcl%d: can't handle af%d. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

pcl%d: stray xmit interrupt. An interrupt occurred when no output had previously been started.

pcl%d: master. The TDM bus had no station providing "bus master" timing signals, so this interface has assumed the "master" role. This message should only appear at most once per UNIBUS INIT on a single system. Unless there is a hardware failure, only one station may be master at time.

pcl%d: send error, tcr=%b, tsr=%b. The device indicated a problem sending data on output. If a "receiver offline" error is detected, it is not normally logged unless the option PCL_TESTING has been selected, as this causes a lot of console chatter when sending to a down machine. However, this option is quite useful when debugging problems with the PCL interfaces.

pcl%d: rcv error, rcr=%b rsr=%b. The device indicated a problem receiving data on input.

pcl%d: bad len=%d. An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never happen as the maximum size of a PCL message has been agreed upon to be 1008 bytes (same as ArpaNet message).

SEE ALSO
intro(4N), inet(4F)
NAME
ps – Evans and Sutherland Picture System 2 graphics device interface

SYNOPSIS
device ps0 at uba? csr 0172460 vector psclockintr pssystemintr

DESCRIPTION
The ps driver provides access to an Evans and Sutherland Picture System 2 graphics device. Each minor device is a new PS2. When the device is opened, its interface registers are mapped, via virtual memory, into a user process’s address space. This allows the user process very high bandwidth to the device with no system call overhead.

DMA to and from the PS2 is not supported. All read and write system calls will fail. All data is moved to and from the PS2 via programmed I/O using the device’s interface registers.

Commands are fed to and from the driver using the following ioctls:

PSIOGETADDR
Returns the virtual address through which the user process can access the device’s interface registers.

PSIAUTOREFRESH
Start auto refreshing the screen. The argument is an address in user space where the following data resides. The first longword is a *count* of the number of static refresh buffers. The next *count* longwords are the addresses in refresh memory where the refresh buffers lie. The driver will cycle through these refresh buffers displaying them one by one on the screen.

PSIAUTOMAP
Start automatically passing the display file through the matrix processor and into the refresh buffer. The argument is an address in user memory where the following data resides. The first longword is a *count* of the number of display files to operate on. The next *count* longwords are the address of these display files. The final longword is the address in refresh buffer memory where transformed coordinates are to be placed if the driver is not in double buffer mode (see below).

PSIODOUBLEBUFFER
Cause the driver to double buffer the output from the map that is going to the refresh buffer. The argument is again a user space address where the real arguments are stored. The first argument is the starting address of refresh memory where the two double buffers are located. The second argument is the length of each double buffer. The refresh mechanism displays the current double buffer, in addition to its static refresh lists, when in double buffer mode.

PSIOSINGLEREFRESH
Single step the refresh process. That is, the driver does not continually refresh the screen.

PSIOSINGLEMAP
Single step the matrix process. The driver does not automatically feed display files through the matrix unit.

PSIOSINGLEBUFFER
Turn off double buffering.

PSIOTIMEREFRESH
The argument is a count of the number of refresh interrupts to take before turning off the screen. This is used to do time exposures.

PSIOWAITREFRESH
Suspend the user process until a refresh interrupt has occurred. If in TIMEREorthand

4.2 Berkeley Distribution May 21, 1986
mode, suspend until count refreshes have occurred.

**PSIOSTOPREFRESH**
Wait for the next refresh, stop all refreshes, and then return to user process.

**PSIOWAITMAP**
Wait until a map done interrupt has occurred.

**PSIOSTOPMAP**
Wait for a map done interrupt, do not restart the map, and then return to the user.

**FILES**
/dev/ps

**DIAGNOSTICS**

- **ps device intr.** An interrupt was received from the device. This shouldn't happen, check your device configuration for overlapping interrupt vectors.

- **ps dma intr.** An interrupt was received from the device. This shouldn't happen, check your device configuration for overlapping interrupt vectors.

**BUGS**

An invalid access (e.g., longword) to a mapped interface register can cause the system to crash with a machine check. A user process could possibly cause infinite interrupts hence bringing things to a crawl.
NAME
pty - pseudo terminal driver

SYNOPSIS
pseudo-device pty [ count ]

DESCRIPTION
The pty driver provides support for a device-pair termed a pseudo terminal. A pseudo terminal is a pair of character devices, a master device and a slave device. The slave device provides processes an interface identical to that described in tty(4). However, whereas all other devices which provide the interface described in tty(4) have a hardware device of some sort behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, anything written on the master device is given to the slave device as input and anything written on the slave device is presented as input on the master device.

In configuring, if an optional "count" is given in the specification, that number of pseudo terminal pairs are configured; the default count is 32.

The following ioctl calls apply only to pseudo terminals:

TIOCSTOP
   Stops output to a terminal (e.g. like typing "S". Takes no parameter.

TIOCSTART
   Restarts output (stopped by TIOCSTOP or by typing "S". Takes no parameter.

TIOCPKT
   Enable/disable packet mode. Packet mode is enabled by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. When applied to the master side of a pseudo terminal, each subsequent read from the terminal will return data written on the slave part of the pseudo terminal preceded by a zero byte (symbolically defined as TIOCPKT_DATA), or a single byte reflecting control status information. In the latter case, the byte is an inclusive-or of zero or more of the bits:

TIOCPKT_FLUSHREAD
   whenever the read queue for the terminal is flushed.

TIOCPKT_FLUSHWRITE
   whenever the write queue for the terminal is flushed.

TIOCPKT_STOP
   whenever output to the terminal is stopped a la "S".

TIOCPKT_START
   whenever output to the terminal is restarted.

TIOCPKT_DOSTOP
   whenever t_stopc is "S" and t_startc is "Q".

TIOCPKT_NOSTOP
   whenever the start and stop characters are not "S"/"Q.

While this mode is in use, the presence of control status information to be read from the master side may be detected by a select for exceptional conditions.

This mode is used by login(1C) and rlogind(8C) to implement a remote-echoed, locally "S"/"Q flow-controlled remote login with proper back-flushing of output; it can be used by other similar programs.

TIOCUCNTL
   Enable/disable a mode that allows a small number of simple user ioctl commands to
be passed through the pseudo-terminal, using a protocol similar to that of TIOCPKT. The TIOCUCNTL and TIOCPKT modes are mutually exclusive. This mode is enabled from the master side of a pseudo terminal by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. Each subsequent read from the master side will return data written on the slave part of the pseudo terminal preceded by a zero byte, or a single byte reflecting a user control operation on the slave side. A user control command consists of a special ioctl operation with no data; the command is given as UIOCCMD(n), where n is a number in the range 1-255. The operation value n will be received as a single byte on the next read from the master side. The ioctl UIOCCMD(0) is a no-op that may be used to probe for the existence of this facility. As with TIOCPKT mode, command operations may be detected with a select for exceptional conditions.

TIOCREMOTE
A mode for the master half of a pseudo terminal, independent of TIOCPKT. This mode causes input to the pseudo terminal to be flow controlled and not input edited (regardless of the terminal mode). Each write to the control terminal produces a record boundary for the process reading the terminal. In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an end-of-file character. TIOCREMOTE can be used when doing remote line editing in a window manager, or whenever flow controlled input is required.

FILES
/dev/tty[p-r][0-9a-f] slave pseudo terminals
/dev/pt[y][p-r][0-9a-f] master pseudo terminals

DIAGNOSTICS
None.
NAME
qe – DEC DEQNA Q-bus 10 Mb/s Ethernet interface

SYNOPSIS
device qe0 at uba? csr 174440 vector qeintr

DESCRIPTION
The qe interface provides access to a 10 Mb/s Ethernet network through the DEC DEQNA Q-bus controller.

Each of the host’s network addresses is specified at boot time with an SIOCSIFADDR ioctl. The qe interface employs the address resolution protocol described in arp(4P) to map dynamically between Internet and Ethernet addresses on the local network.

The interface normally tries to use a “trailer” encapsulation to minimize copying data on input and output. The use of trailers is negotiated with ARP. This negotiation may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIF-FLAGS ioctl.

SEE ALSO
inet(4F), intro(4N), arp(4P)
NAME
rx - DEC RX02 floppy disk interface

SYNOPSIS
controller fx0 at uba0 csr 0177170 vector rxintr
disk rx0 at fx0 drive 0
disk rx1 at fx0 drive 1

DESCRIPTION
The rx device provides access to a DEC RX02 floppy disk unit with M8256 interface module (RX211 configuration). The RX02 uses 8-inch, single-sided, soft-sectored floppy disks (with pre-formatted industry-standard headers) in either single or double density.

Floppy disks handled by the RX02 contain 77 tracks, each with 26 sectors (for a total of 2,002 sectors). The sector size is 128 bytes for single density, 256 bytes for double density. Single density disks are compatible with the RX01 floppy disk unit and with IBM 3740 Series Diskette 1 systems.

In addition to normal ('block' and 'raw') i/o, the driver supports formatting of disks for either density and the ability to invoke a 2 for 1 interleaved sector mapping compatible with the DEC operating system RT-11.

The minor device number is interpreted as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Sector interleaving (1 disables interleaving)</td>
</tr>
<tr>
<td>1</td>
<td>Logical sector 1 is on track 1 (0 no, 1 yes)</td>
</tr>
<tr>
<td>2</td>
<td>Not used, reserved</td>
</tr>
<tr>
<td>Other</td>
<td>Drive number</td>
</tr>
</tbody>
</table>

The two drives in a single RX02 unit are treated as two disks attached to a single controller. Thus, if there are two RX02's on a system, the drives on the first RX02 are "rx0" and "rx1", while the drives on the second are "rx2" and "rx3".

When the device is opened, the density of the disk currently in the drive is automatically determined. If there is no floppy in the device, open will fail.

The interleaving parameters are represented in raw device names by the letters 'a' through 'd'. Thus, unit 0, drive 0 is called by one of the following names:

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Device name</th>
<th>Starting track</th>
</tr>
</thead>
<tbody>
<tr>
<td>interleaved</td>
<td>/dev/rrx0a</td>
<td>0</td>
</tr>
<tr>
<td>direct</td>
<td>/dev/rrx0b</td>
<td>0</td>
</tr>
<tr>
<td>interleaved</td>
<td>/dev/rrx0c</td>
<td>1</td>
</tr>
<tr>
<td>direct</td>
<td>/dev/rrx0d</td>
<td>1</td>
</tr>
</tbody>
</table>

The mapping used on the 'c' device is compatible with the DEC operating system RT-11. The 'b' device accesses the sectors of the disk in strictly sequential order. The 'a' device is the most efficient for disk-to-disk copying. This mapping is always used by the block device.

I/O requests must start on a sector boundary, involve an integral number of complete sectors, and not go off the end of the disk.

NOTES
Even though the storage capacity on a floppy disk is quite small, it is possible to make filesystems on double density disks. For example, the command

% mksf /dev/rrx0 1001 13 1 4096 512 32 0 4

makes a file system on the double density disk in rx0 with 436 kbytes available for file storage. Using tar(1) gives a more efficient utilization of the available space for file storage. Single density diskettes do not provide sufficient storage capacity to hold file systems.
A number of ioctl(2) calls apply to the rx devices, and have the form

```
#include <vaxuba/rxreg.h>
ioctl(fildes, code, arg)
int *arg;
```

The applicable codes are:

**RXIOC_FORMAT**
Format the diskette. The density to use is specified by the *arg* argument, zero gives single density while non-zero gives double density.

**RXIOC_GETDENS**
Return the density of the diskette (zero or non-zero as above).

**RXIOC_WDDMK**
On the next write, include a *deleted data address mark* in the header of the first sector.

**RXIOC_RDDMK**
Return non-zero if the last sector read contained a *deleted data address mark* in its header, otherwise return 0.

**ERRORS**
The following errors may be returned by the driver:

- **[ENODEV]** Drive not ready; usually because no disk is in the drive or the drive door is open.
- **[ENXIO]** Nonexistent drive (on open); offset is too large or not on a sector boundary or byte count is not a multiple of the sector size (on read or write); or bad (undefined) ioctl code.
- **[EIO]** A physical error other than “not ready”, probably bad media or unknown format.
- **[EBUSY]** Drive has been opened for exclusive access.
- **[EBADF]** No write access (on format), or wrong density; the latter can only happen if the disk is changed without closing the device (i.e., calling close(2)).

**FILES**
/dev/rx?
/dev/rrx?[a-d]

**SEE ALSO**
rxformat(8V), newfs(8), mkfs(8), tar(1), arff(8V)

**DIAGNOSTICS**

```
rx%d: hard error, trk %d psec %d cs=%%b, db=%%b, err=%%x, %x, %x, %x. An unrecoverable error was encountered. The track and physical sector numbers, the device registers and the extended error status are displayed.
rx%d: state %d (reset). The driver entered a bogus state. This should not happen.
```

**BUGS**
A floppy may not be formatted if the header info on sector 1, track 0 has been damaged. Hence, it is not possible to format completely degaussed disks or disks with other formats than the two known by the hardware.

If the drive subsystem is powered down when the machine is booted, the controller won't interrupt.
NAME
spp – Xerox Sequenced Packet Protocol

SYNOPSIS
#include <sys/socket.h>
#include <netns/ns.h>
s = socket(AF_NS, SOCK_STREAM, 0);
#include <netns/sp.h>
s = socket(AF_NS, SOCK_SEQPACKET, 0);

DESCRIPTION
The SPP protocol provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the SOCK_STREAM abstraction. SPP uses the standard NS(tm) address formats.

Sockets utilizing the SPP protocol are either “active” or “passive”. Active sockets initiate connections to passive sockets. By default SPP sockets are created active; to create a passive socket the listen(2) system call must be used after binding the socket with the bind(2) system call. Only passive sockets may use the accept(2) call to accept incoming connections. Only active sockets may use the connect(2) call to initiate connections.

Passive sockets may “underspecify” their location to match incoming connection requests from multiple networks. This technique, termed “wildcard addressing”, allows a single server to provide service to clients on multiple networks. To create a socket which listens on all networks, the NS address of all zeroes must be bound. The SPP port may still be specified at this time; if the port is not specified the system will assign one. Once a connection has been established the socket’s address is fixed by the peer entity’s location. The address assigned the socket is the address associated with the network interface through which packets are being transmitted and received. Normally this address corresponds to the peer entity’s network.

If the SOCK_SEQPACKET socket type is specified, each packet received has the actual 12 byte sequenced packet header left for the user to inspect:

```
struct sphdr {
    u_char     sp_cc;     /* connection control */
#define SP_EM 0x10
    u_char     sp_dt;     /* end of message */
    u_short    sp_sid;
    u_short    sp_did;
    u_short    sp_seq;
    u_short    sp_ack;
    u_short    sp_alo;
};
```

This facilitates the implementation of higher level Xerox protocols which make use of the data stream type field and the end of message bit. Conversely, the user is required to supply a 12 byte header, the only part of which inspected is the data stream type and end of message fields.

For either socket type, packets received with the Attention bit sent are interpreted as out of band data. Data sent with send(..., ..., MSG_OOB) cause the attention bit to be set.

DIAGNOSTICS
A socket operation may fail with one of the following errors returned:

[EISCONN] when trying to establish a connection on a socket which already has one;
when the system runs out of memory for an internal data structure;
[ETIMEDOUT] when a connection was dropped due to excessive retransmissions;
[ECONNRESET] when the remote peer forces the connection to be closed;
[ECONNREFUSED] when the remote peer actively refuses connection establishment (usually
because no process is listening to the port);
[EADDRINUSE] when an attempt is made to create a socket with a port which has
already been allocated;
[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for
which no network interface exists.

SOCKET OPTIONS
SO_DEFAULT_HEADERS
when set, this determines the data stream type and whether the end of
message bit is to be set on every ensuing packet.

SO_MTU
This specifies the maximum amount of user data in a single packet.
The default is 576 bytes - sizeof(struct spidp). This quantity affects
windowing – increasing it without increasing the amount of buffering in the
socket will lower the number of unread packets accepted. Anything
larger than the default will not be forwarded by a bona fide XEROX
product internetwork router. The data argument for the setsockopt call
must be an unsigned short.

SEE ALSO
intro(4N), ns(4F)

BUGS
There should be some way to reflect record boundaries in a stream. For stream mode, there
should be an option to get the data stream type of the record the user process is about to
receive.
NAME
  tb – line discipline for digitizing devices

SYNOPSIS
  pseudo-device tb

DESCRIPTION
This line discipline provides a polled interface to many common digitizing devices which are
connected to a host through a serial line. When these devices stream data at high speed, the
use of the line discipline is critical in minimizing the number of samples that would otherwise
be lost due to buffer exhaustion in the tty(4) handler.

The line discipline is enabled by a sequence:

```
#include <sys/tablet.h>
int ldisc = TBLDISC, fildes;
ioctl(fildes, TIOCSETD, &ldisc);
```

A typical application program then polls the digitizing device by reading a binary data struc-
ture which contains: the current X and Y positions (in the device coordinate space), up-down
status of the buttons or pen stylus, proximity information (when available), and a count of the
number of samples received from the input device since it was opened. In addition, devices
such as the GTCO append tilt and pressure information to the end of the aforementioned
structure. For the Polhemus 3-D digitizer the structure read is completely different. Refer to
the include file for a complete description.

While in tablet mode, normal teletype input and output functions take place. Thus, if an 8
bit output data path is desired, it is necessary to prepare the output line by putting it into
RAW mode using ioctl(2). This must be done before changing the discipline with
TIOCSETD, as most ioctl(2) calls are disabled while in tablet line-discipline mode.

The line discipline supports ioctl(2) requests to get/set the operating mode, and to get/set the
tablet type and operating mode by or-ing the two values together.

The line discipline supports digitizing devices which are compatible with Hitachi, GTCO, or
Polhemus protocol formats. For Hitachi there are several formats with that used in the newer
model HDG-1111B the most common.

SEE ALSO
  tty(4)

DIAGNOSTICS
  None.
NAME
tcp - Internet Transmission Control Protocol

SYNOPSIS
#include <sys/socket.h>
#include <netinet/in.h>
s = socket(AF_INET, SOCK_STREAM, 0);

DESCRIPTION
The TCP protocol provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the SOCK_STREAM abstraction. TCP uses the standard Internet address format and, in addition, provides a per-host collection of "port addresses". Thus, each address is composed of an Internet address specifying the host and network, with a specific TCP port on the host identifying the peer entity.

Sockets utilizing the tcp protocol are either "active" or "passive". Active sockets initiate connections to passive sockets. By default TCP sockets are created active; to create a passive socket the listen(2) system call must be used after binding the socket with the bind(2) system call. Only passive sockets may use the accept(2) call to accept incoming connections. Only active sockets may use the connect(2) call to initiate connections.

Passive sockets may "underspecify" their location to match incoming connection requests from multiple networks. This technique, termed "wildcard addressing", allows a single server to provide service to clients on multiple networks. To create a socket which listens on all networks, the Internet address INADDR_ANY must be bound. The TCP port may still be specified at this time; if the port is not specified the system will assign one. Once a connection has been established the socket's address is fixed by the peer entity's location. The address assigned the socket is the address associated with the network interface through which packets are being transmitted and received. Normally this address corresponds to the peer entity's network.

TCP supports one socket option which is set with setsockopt(2) and tested with getsockopt(2). Under most circumstances, TCP sends data when it is presented; when outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgement is received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays. Therefore, TCP provides a boolean option, TCP_NODELAY (from <netinet/tcp.h>, to defeat this algorithm. The option level for the setsockopt call is the protocol number for TCP, available from getprotobyname(3N).

Options at the IP transport level may be used with TCP; see ip(4P). Incoming connection requests that are source-routed are noted, and the reverse source route is used in responding.

DIAGNOSTICS
A socket operation may fail with one of the following errors returned:

[EISCONN] when trying to establish a connection on a socket which already has one;
[ENOBUFS] when the system runs out of memory for an internal data structure;
[ETIMEDOUT] when a connection was dropped due to excessive retransmissions;
[ECONNRESET] when the remote peer forces the connection to be closed;
[ECONNREFUSED] when the remote peer actively refuses connection establishment (usually because no process is listening to the port);
[EADDRINUSE] when an attempt is made to create a socket with a port which has already been allocated;

4.2 Berkeley Distribution May 16, 1986
[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO
getsockopt(2), socket(2), intro(4N), inet(4F), ip(4P)
NAME
tm – TM-11/TE-10 magtape interface

SYNOPSIS
controller tm0 at uba? csr 0172520 vector tmintr
tape te0 at tm0 drive 0

DESCRIPTION
The tm-11/te-10 combination provides a standard tape drive interface as described in mtio(4). Hardware implementing this on the VAX is typified by the Emulex TC-11 controller operating with a Kennedy model 9300 tape transport, providing 800 and 1600 bpi operation at 125 ips.

SEE ALSO
mt(1), tar(1), tp(1), mtio(4), ht(4), ts(4), mt(4), ut(4)

DIAGNOSTICS

te%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.
te%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.
te%d: can’t switch density in mid-tape. An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.
te%d: hard error bn%d er=%b. A tape error occurred at block bn; the tm error register is printed in octal with the bits symbolically decoded. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.
te%d: lost interrupt. A tape operation did not complete within a reasonable time, most likely because the tape was taken off-line during rewind or lost vacuum. The controller should, but does not, give an interrupt in these cases. The device will be made available again after this message, but any current open reference to the device will return an error as the operation in progress aborts.

BUGS
If any non-data error is encountered on non-raw tape, it refuses to do anything more until closed.
NAME
tmscp – DEC TMSCP magtape interface

SYNOPSIS
controller tmscp0 at uba? csr 0174500 vector tmscpintr
tape tms0 at tmscp0 drive 0

DESCRIPTION
Tape controllers compatible with the DEC Tape Mass Storage Control Protocol (TMSCP)
arquitectura such as the TU81 and the TK50 provide a standard tape drive interface as
described in mtio(4). The controller communicates with the host through a packet oriented
protocol. Consult the file <vax/tmscp.h> for a detailed description of this protocol.

DIAGNOSTICS
tmscp controller failed to init. The controller initialization procedure failed. This probably
indicates a hardware problem.
tmscp%d: sa 0%o, state %d. (Additional status information given after a hard I/O error.) The
values of the controller status register and the internal driver state are printed.
tmscp%d: random interrupt ignored. An unexpected interrupt was received (e.g. when no i/o
was pending). The interrupt is ignored.
tmscp%d: interrupt in unknown state %d ignored. An interrupt was received when the driver
was in an unknown internal state. Indicates a hardware problem or a driver bug.
tmscp%d: fatal error (0%o). The controller detected a “fatal error” in the status returned to
the host. The contents of the status register are displayed.
OFFLINE. (Additional status information given after a hard I/O error.) A hard I/O error
occurred because the drive was not on-line.

The following errors are interpretations of TMSCP error messages returned by the controller
to the host. Each is preceded by either tmscp%d: hard error or tmscp%d: soft error.
controller error, event 0%o.
host memory access error, event 0%o, addr 0%o.
tape transfer error, unit %d, grp 0x%x, event 0%o.
STI error, unit %d, event 0%o.
STI Drive Error Log, unit %d, event 0%o.
STI Formatter Error Log, unit %d, event 0%o.
unknown error, unit %d, format 0%o, event 0%o.

SEE ALSO
mt(1), tar(1), tp(1), mtio(4), tm(4), ts(4), ut(4), dmesg(8)
NAME

ts - TS-11 magtape interface

SYNOPSIS

controller zs0 at uba? csr 0172520 vector tsintr
tape ts0 at zs0 drive 0

DESCRIPTION

The ts-11 combination provides a standard tape drive interface as described in mtio(4). The ts-11 operates only at 1600 bpi, and only one transport is possible per controller.

SEE ALSO

mt(1), tar(1), tp(1), mtio(4), ht(4), tm(4), mt(4), ut(4)

DIAGNOSTICS

ts%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

ts%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

ts%d: hard error bs%d xs0=%b. A hard error occurred on the tape at block bn; status register 0 is printed in octal and symbolically decoded as bits.

BUGS

If any non-data error is encountered on non-raw tape, it refuses to do anything more until closed.

The device lives at the same address as a tm-11 tm(4); as it is very difficult to get this device to interrupt, a generic system assumes that a ts is present whenever no tm-11 exists but the csr responds and a ts-11 is configured. This does no harm as long as a non-existent ts-11 is not accessed.
NAME
tty - general terminal interface

SYNOPSIS
#include <sgtty.h>

DESCRIPTION
This section describes both a particular special file /dev/tty and the terminal drivers used for conversational computing.

Line disciplines.
The system provides different line disciplines for controlling communications lines. In this version of the system there are two disciplines available for use with terminals:

old The old (Version 7) terminal driver. This is sometimes used when using the standard shell sh(1).
new The standard Berkeley terminal driver, with features for job control; this must be used when using csh(1).

Line discipline switching is accomplished with the TIOCSETD ioctl:

int ldisc = LDISC;
ioctl(fd, TIOCSETD, &ldisc);

where LDISC is OTTYDISC for the standard tty driver and NTTYDISC for the "new" driver. The standard (currently old) tty driver is discipline 0 by convention. Other disciplines may exist for special purposes, such as use of communications lines for network connections. The current line discipline can be obtained with the TIOCGETD ioctl. Pending input is discarded when the line discipline is changed.

All of the low-speed asynchronous communications ports can use any of the available line disciplines, no matter what hardware is involved. The remainder of this section discusses the "old" and "new" disciplines.

The control terminal.
When a terminal file is opened, it causes the process to wait until a connection is established. In practice, user programs seldom open these files; they are opened by getty(8) or rlogind(8C) and become a user's standard input and output file.

If a process which has no control terminal opens a terminal file, then that terminal file becomes the control terminal for that process. The control terminal is thereafter inherited by a child process during a fork(2), even if the control terminal is closed.

The file /dev/tty is, in each process, a synonym for a control terminal associated with that process. It is useful for programs that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand a file name for output, when typed output is desired and it is tiresome to find out which terminal is currently in use.

A process can remove the association it has with its controlling terminal by opening the file /dev/tty and issuing an

ioctl(fd, TIOCNOTTY, 0);

This is often desirable in server processes.

Process groups.
Command processors such as csh(1) can arbitrate the terminal between different jobs by placing related jobs in a single process group and associating this process group with the terminal. A terminal's associated process group may be set using the TIOCSPPGRP ioctl(2):
ioctl(cfdes, TIOCSPGRP, &pgrp);

or examined using TIOCGPGRP, which returns the current process group in pgrp. The new terminal driver aids in this arbitration by restricting access to the terminal by processes which are not in the current process group; see Job access control below.

Modes.

The terminal drivers have three major modes, characterized by the amount of processing on the input and output characters:

cooked The normal mode. In this mode lines of input are collected and input editing is done. The edited line is made available when it is completed by a newline, or when the \texttt{\textasciitilde brkc} character (normally undefined) or \texttt{\textasciitilde eofc} character (normally an EOT, control-D, hereafter 'D) is entered. A carriage return is usually made synonymous with newline in this mode, and replaced with a newline whenever it is typed. All driver functions (input editing, interrupt generation, output processing such as delay generation and tab expansion, etc.) are available in this mode.

CBREAK This mode eliminates the character, word, and line editing input facilities, making the input character available to the user program as it is typed. Flow control, literal-next and interrupt processing are still done in this mode. Output processing is done.

RAW This mode eliminates all input processing and makes all input characters available as they are typed; no output processing is done either.

The style of input processing can also be very different when the terminal is put in non-blocking I/O mode; see the FNDELAY flag described in \texttt{fcntl(2)}. In this case a \texttt{read(2)} from the control terminal will never block, but rather return an error indication (EWOULD BLOCK) if there is no input available.

A process may also request that a SIGIO signal be sent it whenever input is present and also whenever output queues fall below the low-water mark. To enable this mode the FASYNC flag should be set using \texttt{fcntl(2)}.

Input editing.

A UNIX terminal ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely choked, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently this limit is 256 characters. In RAW mode, the terminal driver throws away all input and output without notice when the limit is reached. In CBREAK or cooked mode it refuses to accept any further input and, if in the new line discipline, rings the terminal bell.

Input characters are normally accepted in either even or odd parity with the parity bit being stripped off before the character is given to the program. By clearing either the EVEN or ODD bit in the flags word it is possible to have input characters with that parity discarded (see the \textit{Summary} below.)

In all of the line disciplines, it is possible to simulate terminal input using the TIOCSTI ioctl, which takes, as its third argument, the address of a character. The system pretends that this character was typed on the argument terminal, which must be the control terminal except for the super-user (this call is not in standard version 7 UNIX).

Input characters are normally echoed by putting them in an output queue as they arrive. This may be disabled by clearing the ECHO bit in the flags word using the \texttt{stty(3C)} call or the TIOCSETN or TIOCSETP ioctl (see the \textit{Summary} below).
In cooked mode, terminal input is processed in units of lines. A program attempting to read will normally be suspended until an entire line has been received (but see the description of SIGTTIN in Job access control and of FIONREAD in Summary, both below.) No matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, line editing is normally done, with the erase character sg_erase (by default, DELETE) logically erasing the last character typed and the sg_kill character (default, "U": control-U) logically erasing the entire current input line. These characters never erase beyond the beginning of the current input line or an eof. These characters may be entered literally by preceding them with '\'; the '\' will normally be erased when the character is typed.

The drivers normally treat either a carriage return or a newline character as terminating an input line, replacing the return with a newline and echoing a return and a line feed. If the CRMOD bit is cleared in the local mode word then the processing for carriage return is disabled, and it is simply echoed as a return, and does not terminate cooked mode input.

In the new driver there is a literal-next character (normally "V") which can be typed in both cooked and CBREAK mode preceding any character to prevent its special meaning to the terminal handler. This is to be preferred to the use of "\" escaping erase and kill characters, but "\" is retained with its old function in the new line discipline.

The new terminal driver also provides two other editing characters in normal mode. The word-erase character, normally "W", erases the preceding word, but not any spaces before it. For the purposes of "W", a word is defined as a sequence of non-blank characters, with tabs counted as blanks. Finally, the reprint character, normally "R", retypes the pending input beginning on a new line. Retyping occurs automatically in cooked mode if characters which would normally be erased from the screen are fouled by program output.

**Input echoing and redisplay**

The terminal driver has several modes (not present in standard UNIX Version 7 systems) for handling the echoing of terminal input, controlled by bits in a local mode word.

**Hardcopy terminals.** When a hardcopy terminal is in use, the LPRTERA bit is normally set in the local mode word. Characters which are logically erased are then printed out backwards preceded by "\" and followed by "\" in this mode.

**CRT terminals.** When a CRT terminal is in use, the LCRTBS bit is normally set in the local mode word. The terminal driver then echoes the proper number of erase characters when input is erased; in the normal case where the erase character is a "H" this causes the cursor of the terminal to back up to where it was before the logically erased character was typed. If the input has become fouled due to interspersed asynchronous output, the input is automatically retyped.

**Erasing characters from a CRT.** When a CRT terminal is in use, the LCRTBLA bit may be set to cause input to be erased from the screen with a "backspace-space-backspace" sequence when character or word deleting sequences are used. A LCRTKIL bit may be set as well, causing the input to be erased in this manner on line kill sequences as well.

**Echoing of control characters.** If the LCTLECH bit is set in the local state word, then non-printing (control) characters are normally echoed as "X" (for some X) rather than being echoed unmodified; delete is echoed as "\".

The normal modes for use on CRT terminals are speed dependent. At speeds less than 1200 baud, the LCRTBLA and LCRTKILL processing is painfully slow, and stty(1) normally just sets LCRTBS and LCTLECH; at speeds of 1200 baud or greater all of these bits are normally set. Stty(1) summarizes these option settings and the use of the new terminal driver as "newcrt."
Output processing.

When one or more characters are written, they are actually transmitted to the terminal as soon as previously-written characters have finished typing. (As noted above, input characters are normally echoed by putting them in the output queue as they arrive.) When a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold the program is resumed. Even parity is normally generated on output. The EOT character is not transmitted in cooked mode to prevent terminals that respond to it from hanging up; programs using RAW or CBREAK mode should be careful.

The terminal drivers provide necessary processing for cooked and CBREAK mode output including delay generation for certain special characters and parity generation. Delays are available after backspaces H, form feeds L, carriage returns M, tabs I and newlines J. The driver will also optionally expand tabs into spaces, where the tab stops are assumed to be set every eight columns, and optionally convert newlines to carriage returns followed by newline. These functions are controlled by bits in the tty flags word; see Summary below.

The terminal drivers provide for mapping between upper and lower case on terminals lacking lower case, and for other special processing on deficient terminals.

Finally, in the new terminal driver, there is a output flush character, normally O, which sets the LFLUSHO bit in the local mode word, causing subsequent output to be flushed until it is cleared by a program or more input is typed. This character has effect in both cooked and CBREAK modes and causes pending input to be retyped if there is any pending input. An ioctl to flush the characters in the input or output queues, TIOCFLUSH, is also available.

Upper case terminals and Hazeltines

If the LCASE bit is set in the tty flags, then all upper-case letters are mapped into the corresponding lower-case letter. The upper-case letter may be generated by preceding it by \'. Upper case letters are preceded by a \' when output. In addition, the following escape sequences can be generated on output and accepted on input:

```
\"  \"  ( )
```

To deal with Hazeltine terminals, which do not understand that \ has been made into an ASCII character, the LTILDE bit may be set in the local mode word; in this case the character \ will be replaced with the character \ on output.

Flow control.

There are two characters (the stop character, normally S, and the start character, normally Q) which cause output to be suspended and resumed respectively. Extra stop characters typed when output is already stopped have no effect, unless the start and stop characters are made the same, in which case output resumes.

A bit in the flags word may be set to put the terminal into TANDEM mode. In this mode the system produces a stop character (default S) when the input queue is in danger of overflowing, and a start character (default Q) when the input has drained sufficiently. This mode is useful when the terminal is actually another machine that obeys those conventions.

Line control and breaks.

There are several ioctl calls available to control the state of the terminal line. The TIOCSBRK ioctl will set the break bit in the hardware interface causing a break condition to exist; this can be cleared (usually after a delay with sleep(3)) by TIOCCBRK. Break conditions in the input are reflected as a null character in RAW mode or as the interrupt character in cooked or CBREAK mode. The TIOCCDTR ioctl will clear the data terminal ready condition; it can be set again by TIOCSDTR.
When the carrier signal from the dataset drops (usually because the user has hung up his terminal) a SIGHUP hangup signal is sent to the processes in the distinguished process group of the terminal; this usually causes them to terminate. The SIGHUP can be suppressed by setting the LNOHANG bit in the local state word of the driver. Access to the terminal by other processes is then normally revoked, so any further reads will fail, and programs that read a terminal and test for end-of-file on their input will terminate appropriately.

It is possible to ask that the phone line be hung up on the last close with the TIOCHPCL ioctl; this is normally done on the outgoing lines and dialups.

**Interrupt characters.**

There are several characters that generate interrupts in cooked and CBREAK mode; all are sent to the processes in the control group of the terminal, as if a TIOCGPGRP ioctl were done to get the process group and then a killpg(2) system call were done, except that these characters also flush pending input and output when typed at a terminal (à la TIOCFflush). The characters shown here are the defaults; the field names in the structures (given below) are also shown. The characters may be changed.

- `t_intrc (ETX)` generates a SIGINT signal. This is the normal way to stop a process which is no longer interesting, or to regain control in an interactive program.
- `t_quitc (FS)` generates a SIGQUIT signal. This is used to cause a program to terminate and produce a core image, if possible, in the file core in the current directory.
- `t_suspc (EM)` generates a SIGTSTP signal, which is used to suspend the current process group.
- `t_dsuspc (SUB)` generates a SIGTSTP signal as `t_suspc` does, but the signal is sent when a program attempts to read the `Y`, rather than when it is typed.

**Job access control.**

When using the new terminal driver, if a process which is not in the distinguished process group of its control terminal attempts to read from that terminal its process group is sent a SIGTTIN signal. This signal normally causes the members of that process group to stop. If, however, the process is ignoring SIGTTIN, has SIGTTIN blocked, or is in the middle of process creation using vfork(2), the read will return -1 and set errno to EIO.

When using the new terminal driver with the LSTOP bit set in the local modes, a process is prohibited from writing on its control terminal if it is not in the distinguished process group for that terminal. Processes which are holding or ignoring SIGTTOU signals or which are in the middle of a vfork(2) are excepted and allowed to produce output. **Terminal/window sizes.** In order to accommodate terminals and workstations with variable-sized windows, the terminal driver provides a mechanism for obtaining and setting the current terminal size. The driver does not use this information internally, but only stores it and provides a uniform access mechanism. When the size is changed, a SIGWINCH signal is sent to the terminal’s process group so that knowledgeable programs may detect size changes. This facility was added in 4.3BSD and is not available in earlier versions of the system.

**Summary of modes.**

Unfortunately, due to the evolution of the terminal driver, there are 4 different structures which contain various portions of the driver data. The first of these (sgttyb) contains that part of the information largely common between version 6 and version 7 UNIX systems. The second contains additional control characters added in version 7. The third is a word of local state added in 4BSD, and the fourth is another structure of special characters added for the new driver. In the future a single structure may be made available to programs which need to access all this information; most programs need not concern themselves with all this state.
Basic modes: sgtty.

The basic ioctls use the structure defined in <sgtty.h>:

```
struct sgttyb {
  char sg_ispeed;
  char sg_ospeed;
  char sg_erase;
  char sg_kill;
  short sg_flags;
};
```

The sg_ispeed and sg_ospeed fields describe the input and output speeds of the device according to the following table, which corresponds to the DEC DH-11 interface. If other hardware is used, impossible speed changes are ignored. Symbolic values in the table are as defined in <sgtty.h>.

<table>
<thead>
<tr>
<th>Code (decimal)</th>
<th>Speed (baud)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>0</td>
</tr>
<tr>
<td>B50</td>
<td>1</td>
</tr>
<tr>
<td>B75</td>
<td>2</td>
</tr>
<tr>
<td>B110</td>
<td>3</td>
</tr>
<tr>
<td>B134</td>
<td>4</td>
</tr>
<tr>
<td>B150</td>
<td>5</td>
</tr>
<tr>
<td>B200</td>
<td>6</td>
</tr>
<tr>
<td>B300</td>
<td>7</td>
</tr>
<tr>
<td>B600</td>
<td>8</td>
</tr>
<tr>
<td>B1200</td>
<td>9</td>
</tr>
<tr>
<td>B1800</td>
<td>10</td>
</tr>
<tr>
<td>B2400</td>
<td>11</td>
</tr>
<tr>
<td>B4800</td>
<td>12</td>
</tr>
<tr>
<td>B9600</td>
<td>13</td>
</tr>
<tr>
<td>EXTA</td>
<td>14</td>
</tr>
<tr>
<td>EXTB</td>
<td>15</td>
</tr>
</tbody>
</table>

Code conversion and line control required for IBM 2741’s (134.5 baud) must be implemented by the user’s program. The half-duplex line discipline required for the 202 dataset (1200 baud) is not supplied; full-duplex 212 datasets work fine.

The sg_erase and sg_kill fields of the argument structure specify the erase and kill characters respectively. (Defaults are DELETE and ‘U.)

The sg_flags field of the argument structure contains several bits that determine the system’s treatment of the terminal:

```
ALLDELAY 0177400 Delay algorithm selection
BSDELAY 0100000 Select backspace delays (not implemented):
  BS0 0
  BS1 0100000
VTDELAY 0040000 Select form-feed and vertical-tab delays:
  FF0 0
  FF1 0040000
CRDELAY 0030000 Select carriage-return delays:
  CR0 0
  CR1 0010000
  CR2 0020000
  CR3 0030000
TBDELAY 0006000 Select tab delays:
  TAB0 0
```
TTY (4)

UNIX Programmer’s Manual

TTY (4)

TAB1 0002000
TAB2 0004000
XTABS 0006000
NLDELAY 0001400 Select new-line delays:
NL0 0
NL1 0000400
NL2 0001000
NL3 0001400
EVENP 0000200 Even parity allowed on input
ODDP 0000100 Odd parity allowed on input
RAW 0000040 Raw mode: wake up on all characters, 8-bit interface
CRMOD 0000020 Map CR into LF; output LF as CR-LF
ECHO 0000010 Echo (full duplex)
LCASE 0000004 Map upper case to lower on input and lower to upper on output
CBREAK 0000002 Return each character as soon as typed
TANDEM 0000001 Automatic flow control

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay.

Backspace delays are currently ignored but might be used for Terminet 300’s.

If a form-feed/vertical tab delay is specified, it lasts for about 2 seconds.

Carriage-return delay type 1 lasts about .08 seconds and is suitable for the Terminet 300. Delay type 2 lasts about .16 seconds and is suitable for the VT05 and the TI 700. Delay type 3 is suitable for the concept-100 and pads lines to be at least 9 characters at 9600 baud.

New-line delay type 1 is dependent on the current column and is tuned for Teletype model 37’s. Type 2 is useful for the VT05 and is about .10 seconds. Type 3 is unimplemented and is 0.

Tab delay type 1 is dependent on the amount of movement and is tuned to the Teletype model 37. Type 3, called XTABS, is not a delay at all but causes tabs to be replaced by the appropriate number of spaces on output.

The flags for even and odd parity control parity checking on input and generation on output in cooked and CBREAK mode (unless LPASS8 is enabled, see below). Even parity is generated on output unless ODDP is set and EVENP is clear, in which case odd parity is generated. Input characters with the wrong parity, as determined by EVENP and ODDP, are ignored in cooked and CBREAK mode.

RAW disables all processing save output flushing with LFLUSHO; full 8 bits of input are given as soon as it is available; all 8 bits are passed on output. A break condition in the input is reported as a null character. If the input queue overflows in raw mode all data in the input and output queues are discarded; this applies to both new and old drivers.

CRMOD causes input carriage returns to be turned into new-lines, and output and echoed new-lines to be output as a carriage return followed by a line feed.

CBREAK is a sort of half-cooked (rare?) mode. Programs can read each character as soon as typed, instead of waiting for a full line; all processing is done except the input editing: character and word erase and line kill, input reprint, and the special treatment of \ and EOT are disabled.

TANDEM mode causes the system to produce a stop character (default ‘$’) whenever the input queue is in danger of overflowing, and a start character (default ‘Q’) when the input queue has drained sufficiently. It is useful for flow control when the ‘terminal’ is really another computer which understands the conventions.
Note: The same "stop" and "start" characters are used for both directions of flow control; the \texttt{t_stopc} character is accepted on input as the character that stops output and is produced on output as the character to stop input, and the \texttt{t_startc} character is accepted on input as the character that restarts output and is produced on output as the character to restart input.

Basic ioctl

A large number of \texttt{ioctl(2)} calls apply to terminals. Some have the general form:

\begin{verbatim}
#include <sgtty.h>
ioctl(filedes, code, arg)
struct sgttyb *arg;
\end{verbatim}

The applicable codes are:

- \texttt{TIOCGETP}: Fetch the basic parameters associated with the terminal, and store in the pointed-to \texttt{sgttyb} structure.
- \texttt{TIOCSETP}: Set the parameters according to the pointed-to \texttt{sgttyb} structure. The interface delays until output is quiescent, then throws away any unread characters, before changing the modes.
- \texttt{TIOCSETN}: Set the parameters like \texttt{TIOCSETP} but do not delay or flush input. Input is not preserved, however, when changing to or from RAW.

With the following codes \texttt{arg} is ignored.

- \texttt{TIOCEXCL}: Set "exclusive-use" mode: no further opens are permitted until the file has been closed.
- \texttt{TIOCNXCL}: Turn off "exclusive-use" mode.
- \texttt{TIOCHPCL}: When the file is closed for the last time, hang up the terminal. This is useful when the line is associated with an ACU used to place outgoing calls.

With the following codes \texttt{arg} is a pointer to an int.

- \texttt{TIOCGETD}: \texttt{arg} is a pointer to an int into which is placed the current line discipline number.
- \texttt{TIOCSETD}: \texttt{arg} is a pointer to an int whose value becomes the current line discipline number.
- \texttt{TIOCFLUSH}: If the int pointed to by \texttt{arg} has a zero value, all characters waiting in input or output queues are flushed. Otherwise, the value of the int is for the FREAD and FWRITE bits defined in \texttt{<sys/file.h>}; if the FREAD bit is set, all characters waiting in input queues are flushed, and if the FWRITE bit is set, all characters waiting in output queues are flushed.

The remaining calls are not available in vanilla version 7 UNIX. In cases where arguments are required, they are described; \texttt{arg} should otherwise be given as 0.

- \texttt{TIOCSTI}: the argument points to a character which the system pretends had been typed on the terminal.
- \texttt{TIOCSBRK}: the break bit is set in the terminal.
- \texttt{TIOCCBRK}: the break bit is cleared.
- \texttt{TIOCSDTR}: data terminal ready is set.
- \texttt{TIOCCDTR}: data terminal ready is cleared.
- \texttt{TIOCSTOP}: output is stopped as if the "stop" character had been typed.
- \texttt{TIOCSTART}: output is restarted as if the "start" character had been typed.
TTY(4)

UNIX Programmer's Manual

TTY(4)

TIOCGPGRP.arg is a pointer to an int into which is placed the process group ID of the process group for which this terminal is the control terminal.

TIOCSGPRP.arg is a pointer to an int which is the value to which the process group ID for this terminal will be set.

TIOCOUTQ returns in the int pointed to by arg the number of characters queued for output to the terminal.

FIONREAD returns in the int pointed to by arg the number of characters immediately readable from the argument descriptor. This works for files, pipes, and terminals.

Tchars
The second structure associated with each terminal specifies characters that are special in both the old and new terminal interfaces: The following structure is defined in <sys/ioctl.h>, which is automatically included in <sgtty.h>:

```c
struct tchars {
    char t_intrc; /* interrupt */
    char t_quitc; /* quit */
    char t_startc; /* start output */
    char t_stopc; /* stop output */
    char t.eofc; /* end-of-file */
    char t_brkc; /* input delimiter (like nl) */
};
```

The default values for these characters are ‘C, ‘\\, ‘Q, ‘S, ‘D, and -1. A character value of -1 eliminates the effect of that character. The t_brkc character, by default -1, acts like a newline in that it terminates a 'line,' is echoed, and is passed to the program. The 'stop' and 'start' characters may be the same, to produce a toggle effect. It is probably counterproductive to make other special characters (including erase and kill) identical. The applicable ioctl calls are:

TIOCGGETC Get the special characters and put them in the specified structure.

TIOCSGETC Set the special characters to those given in the structure.

Local mode
The third structure associated with each terminal is a local mode word. The bits of the local mode word are:

- LCRTBS 000001 Backspace on erase rather than echoing erase
- LPRTERA 000002 Printing terminal erase mode
- LCRTERA 000004 Erase character echoes as backspace-space-backspace
- LTILDE 000010 Convert ~ to \ on output (for Hazeltine terminals)
- LMDMBUF 000020 Stop/start output when carrier drops
- LLITOUT 000040 Suppress output translations
- LTOSTOP 000100 Send SIGTTOU for background output
- LFLUSHO 000200 Output is being flushed
- LNOHANG 000400 Don't send hangup when carrier drops
- LETXACK 001000 Diablo style buffer hacking (unimplemented)
- LCRTKIL 002000 BS-space-BS erase entire line on line kill
- LPASS8 004000 Pass all 8 bits through on input, in any mode
- LCTLECH 010000 Echo input control chars as "X, delete as "?
- LPENDIN 020000 Retype pending input at next read or input character
- LDEECTQ 040000 Only 'Q restarts output after 'S, like DEC systems
- LNOFLSH 100000 Inhibit flushing of pending I/O when an interrupt character is typed.
The applicable *ioctl* functions are:

**TIOCLBIS**  
*arg* is a pointer to an int whose value is a mask containing the bits to be set in the local mode word.

**TIOCLBIC**  
*arg* is a pointer to an int whose value is a mask containing the bits to be cleared in the local mode word.

**TIOCLSET**  
*arg* is a pointer to an int whose value is stored in the local mode word.

**TIOCLGET**  
*arg* is a pointer to an int into which the current local mode word is placed.

**Local special chars**

The final control structure associated with each terminal is the *ltchars* structure which defines control characters for the new terminal driver. Its structure is:

```c
struct ltchars {
    char t_suspc; /* stop process signal */
    char t_dsuspc; /* delayed stop process signal */
    char t_rprntc; /* reprint line */
    char t_flush; /* flush output (toggles) */
    char t_werase; /* word erase */
    char t_nextc; /* literal next character */
};
```

The default values for these characters are 'Z, 'Y, 'R, 'O, 'W, and 'V. A value of -1 disables the character.

The applicable *ioctl* functions are:

**TIOCSLTC**  
*arg* is a pointer to an *ltchars* structure which defines the new local special characters.

**TIOCGLTC**  
*arg* is a pointer to an *ltchars* structure into which is placed the current set of local special characters.

**Window/terminal sizes**

Each terminal has provision for storage of the current terminal or window size in a *winsize* structure, with format:

```c
struct winsize {
    unsigned short ws_row; /* rows, in characters */
    unsigned short ws_col; /* columns, in characters */
    unsigned short ws_xpixel; /* horizontal size, pixels */
    unsigned short ws_ypixel; /* vertical size, pixels */
};
```

A value of 0 in any field is interpreted as "undefined;" the entire structure is zeroed on final close.

The applicable *ioctl* functions are:

**TIOCGWINSZ**

*arg* is a pointer to a struct *winsize* into which will be placed the current terminal or window size information.

**TIOCWSWINsz**

*arg* is a pointer to a struct *winsize* which will be used to set the current terminal or window size information. If the new information is different than the old information, a SIGWINCH signal will be sent to the terminal's process group.
FILES
/dev/tty
/dev/ttye
/dev/console

SEE ALSO
csh(1), stty(1), tset(1), ioctl(2), sigvec(2), stty(3C), getty(8)
NAME
tu - VAX-11/730 and VAX-11/750 TU58 console cassette interface

SYNOPSIS
options MRSP (for VAX-11/750's with an MRSP prom)

DESCRIPTION
The *tu* interface provides access to the VAX 11/730 and 11/750 TU58 console cassette drive(s).

The interface supports only block i/o to the TU58 cassettes. The devices are normally manipulated with the *arff*(8V) program using the "f" and "m" options.

The device driver is automatically included when a system is configured to run on an 11/730 or 11/750.

The TU58 on an 11/750 uses the Radial Serial Protocol (RSP) to communicate with the cpu over a serial line. This protocol is inherently unreliable as it has no flow control measures built in. On an 11/730 the Modified Radial Serial Protocol is used. This protocol incorporates flow control measures which insure reliable data transfer between the cpu and the device. Certain 11/750's have been modified to use the MRSP prom used in the 11/730. To reliably use the console TU58 on an 11/750 under UNIX, the MRSP prom is required. For those 11/750's without an MRSP prom, an unreliable but often useable interface has been developed. This interface uses an assembly language "pseudo-dma" routine to minimize the receiver interrupt service latency. To include this code in the system, the configuration must not specify the system will run on an 11/730 or use an MRSP prom. This unfortunately makes it impossible to configure a single system which will properly handle TU58's on both an 11/750 and an 11/730 (unless both machines have MRSP proms).

FILES
/dev/tu0
/dev/tu1 (only on a VAX-11/730)

SEE ALSO
arff(8V)

DIAGNOSTICS
tu%%: no bp, active %d. A transmission complete interrupt was received with no outstanding i/o request. This indicates a hardware problem.

tu%%d protocol error, state=%%s, op=%%x, cnt=%%d, block=%%d. The driver entered an illegal state. The information printed indicates the illegal state, operation currently being executed, the i/o count, and the block number on the cassette.

tu%%d receive state error, state=%%s, byte=%%x. The driver entered an illegal state in the receiver finite state machine. The state is shown along with the control byte of the received packet.

tu%%d: read stalled. A timer watching the controller detected no interrupt for an extended period while an operation was outstanding. This usually indicates that one or more receiver interrupts were lost and the transfer is restarted (11/750 only).

tu%%: hard error bn%%d, pk_mod %o. The device returned a status code indicating a hard error. The actual error code is shown in octal. No retries are attempted by the driver.

BUGS
The VAX-11/750 console interface without MRSP prom is uneable while the system is multi-user; it should be used only with the system running single-user and, even then, with caution.
NAME
uda – UDA-50 disk controller interface

SYNOPSIS
controller uda0 at uba0 csr 0172150 vector udntr
disk ra0 at uda0 drive 0

DESCRIPTION
This is a driver for the DEC UDA-50 disk controller and for other compatible controllers. The UDA-50 communicates with the host through a packet oriented protocol termed the Mass Storage Control Protocol (MSCP). Consult the file <vax/mscp.h> for a detailed description of this protocol.

Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, etc. The standard device names begin with “ra” followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-7.

The block files access the disk via the system’s normal buffering mechanism and may be read and written without regard to physical disk records. There is also a ‘raw’ interface which provides for direct transmission between the disk and the user’s read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra ‘r.’

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise seek calls should specify a multiple of 512 bytes.

DISK SUPPORT
This driver configures the drive type of each drive when it is first opened. A partition table in the driver is required for each type of disk. The origin and size (in sectors) of the pseudodisks on each drive are shown below. Not all partitions begin on cylinder boundaries, as on other drives, because previous drivers used one partition table for all drive types. Variants of the partition tables are common; check the driver and the file /etc/disktab(disktab(5)) for other possibilities.

RC25 partitions

+-----------------+-------+------------------+
<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra?a</td>
<td>0</td>
<td>15884</td>
</tr>
<tr>
<td>ra?b</td>
<td>15884</td>
<td>10032</td>
</tr>
<tr>
<td>ra?c</td>
<td>0</td>
<td>50902</td>
</tr>
<tr>
<td>ra?g</td>
<td>25916</td>
<td>24986</td>
</tr>
</tbody>
</table>

RD52 partitions

+-----------------+-------+------------------+
<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra?a</td>
<td>0</td>
<td>15884</td>
</tr>
<tr>
<td>ra?b</td>
<td>15884</td>
<td>9766</td>
</tr>
<tr>
<td>ra?c</td>
<td>0</td>
<td>60480</td>
</tr>
<tr>
<td>ra?g</td>
<td>25650</td>
<td>34830</td>
</tr>
</tbody>
</table>

RD53 partitions

+-----------------+-------+------------------+
<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra?a</td>
<td>0</td>
<td>15884</td>
</tr>
<tr>
<td>ra?b</td>
<td>15884</td>
<td>33440</td>
</tr>
<tr>
<td>ra?c</td>
<td>0</td>
<td>138672</td>
</tr>
<tr>
<td>ra?g</td>
<td>49324</td>
<td>89348</td>
</tr>
<tr>
<td>ra?h</td>
<td>15884</td>
<td>122788</td>
</tr>
</tbody>
</table>

RA60 partitions

+-----------------+-------+------------------+
<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra?a</td>
<td>0</td>
<td>15884</td>
</tr>
</tbody>
</table>
The ra?a partition is normally used for the root file system, the ra?b partition as a paging area, and the ra?c partition for pack-pack copying (it maps the entire disk).

FILES
/dev/ra[0-9][a-f]/dev/rra[0-9][a-f]

DIAGNOSTICS
uda: ubinfo %x. (VAX 11/750 only.) When allocating UNIBUS resources, the driver found it already had resources previously allocated. This indicates a bug in the driver.

udasa %o, state %d. (Additional status information given after a hard i/o error.) The values of the UDA-50 status register and the internal driver state are printed.

uda%d: random interrupt ignored. An unexpected interrupt was received (e.g. when no i/o was pending). The interrupt is ignored.
uda%d: interrupt in unknown state %d ignored. An interrupt was received when the driver was in an unknown internal state. Indicates a hardware problem or a driver bug.

uda%d: fatal error (%o). The UDA-50 indicated a "fatal error" in the status returned to the host. The contents of the status register are displayed.

OFFLINE. (Additional status information given after a hard i/o error.) A hard i/o error occurred because the drive was not on-line.

status %o. (Additional status information given after a hard i/o error.) The status information returned from the UDA-50 is tacked onto the end of the hard error message printed on the console.

uda: unknown packet. An MSCP packet of unknown type was received from the UDA-50. Check the cabling to the controller.

The following errors are interpretations of MSCP error messages returned by the UDA-50 to the host.

uda%d: %s error, controller error, event 0%0.
uda%d: %s error, host memory access error, event 0%0, addr 0%0.
uda%d: %s error, disk transfer error, unit %d.
uda%d: %s error, SDI error, unit %d, event 0%0.
uda%d: %s error, small disk error, unit %d, event 0%0, cyl %d.
uda%d: %s error, unknown error, unit %d, format 0%0, event 0%0.

BUGS

The partition tables attempt to combine compatibility with previous drivers and functionality; this is impossible. The best solution would be to read the partition tables off the drive.
NAME
udp – Internet User Datagram Protocol

SYNOPSIS
#include <sys/socket.h>
#include <netinet/in.h>
s = socket(AF_INET, SOCK_DGRAM, 0);

DESCRIPTION
UDP is a simple, unreliable datagram protocol which is used to support the SOCK_DGRAM abstraction for the Internet protocol family. UDP sockets are connectionless, and are normally used with the sendto and recvfrom calls, though the connect(2) call may also be used to fix the destination for future packets (in which case the recv(2) or read(2) and send(2) or write(2) system calls may be used).

UDP address formats are identical to those used by TCP. In particular UDP provides a port identifier in addition to the normal Internet address format. Note that the UDP port space is separate from the TCP port space (i.e. a UDP port may not be “connected” to a TCP port).

In addition broadcast packets may be sent (assuming the underlying network supports this) by using a reserved “broadcast address”; this address is network interface dependent.

Options at the IP transport level may be used with UDP; see ip(4P).

DIAGNOSTICS
A socket operation may fail with one of the following errors returned:
[EISCONN] when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket hasn’t been connected;
[ENOBUFFS] when the system runs out of memory for an internal data structure;
[EADDRINUSE] when an attempt is made to create a socket with a port which has already been allocated;
[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO
getsockopt(2), recv(2), send(2), socket(2), intro(4N), inet(4F), ip(4P)
NAME
up - unibus storage module controller/drives

SYNOPSIS
controller sc0 at uba? csr 0176700 vector upintr
disk up0 at sc0 drive 0

DESCRIPTION
This is a generic UNIBUS storage module disk driver. It is specifically designed to work with the Emulex SC-21 and SC-31 controllers. It can be easily adapted to other controllers (although bootstrapping will not necessarily be directly possible.)

Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, etc. The standard device names begin with "up" followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-7.

The block files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise seek calls should specify a multiple of 512 bytes.

DISK SUPPORT
The driver interrogates the controller's holding register to determine the type of drive attached. The driver recognizes seven different drives: CDC 9762, CDC 9766, AMPEX DM980, AMPEX 9300, AMPEX Capricorn, FUJITSU 160, and FUJITSU Eagle (the Eagle is not supported by the SC-21). The origin and size of the pseudo-disks on each drive are as follows:

**CDC 9762 partitions**

<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
<th>cyls</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp?a</td>
<td>0</td>
<td>15884</td>
<td>0-99</td>
</tr>
<tr>
<td>hp?b</td>
<td>16000</td>
<td>33440</td>
<td>100-309</td>
</tr>
<tr>
<td>hp?c</td>
<td>0</td>
<td>131680</td>
<td>0-822</td>
</tr>
<tr>
<td>hp?d</td>
<td>49600</td>
<td>15884</td>
<td>309-408</td>
</tr>
<tr>
<td>hp?e</td>
<td>65440</td>
<td>55936</td>
<td>409-758</td>
</tr>
<tr>
<td>hp?f</td>
<td>121440</td>
<td>10080</td>
<td>759-822</td>
</tr>
<tr>
<td>hp?g</td>
<td>49600</td>
<td>82080</td>
<td>309-822</td>
</tr>
</tbody>
</table>

**CDC 9766 300M drive partitions:**

<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
<th>cyl</th>
</tr>
</thead>
<tbody>
<tr>
<td>up?a</td>
<td>0</td>
<td>15884</td>
<td>0-26</td>
</tr>
<tr>
<td>up?b</td>
<td>16416</td>
<td>33440</td>
<td>27-81</td>
</tr>
<tr>
<td>up?c</td>
<td>0</td>
<td>500384</td>
<td>0-822</td>
</tr>
<tr>
<td>up?d</td>
<td>341696</td>
<td>15884</td>
<td>562-588</td>
</tr>
<tr>
<td>up?e</td>
<td>358112</td>
<td>55936</td>
<td>589-680</td>
</tr>
<tr>
<td>up?f</td>
<td>414048</td>
<td>861760</td>
<td>681-822</td>
</tr>
<tr>
<td>up?g</td>
<td>341696</td>
<td>158528</td>
<td>562-822</td>
</tr>
<tr>
<td>up?h</td>
<td>49856</td>
<td>291346</td>
<td>82-561</td>
</tr>
</tbody>
</table>

**AMPEX DM980 partitions**

<table>
<thead>
<tr>
<th>disk</th>
<th>start</th>
<th>length</th>
<th>cyls</th>
</tr>
</thead>
<tbody>
<tr>
<td>hp?a</td>
<td>0</td>
<td>15884</td>
<td>0-99</td>
</tr>
<tr>
<td>hp?b</td>
<td>16000</td>
<td>33440</td>
<td>100-309</td>
</tr>
</tbody>
</table>
It is unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter. The up?a partition is normally used for the root file system, the up?b partition as a paging area, and the up?c partition for pack-pack copying (it maps the entire disk). On 160M drives the up?g partition maps the rest of the pack. On other drives both up?g and up?h are used to map the remaining cylinders.
FILES
/dev/up[0-7][a-h]  block files
/dev/rup[0-7][a-h]  raw files

SEE ALSO
hk(4), hp(4), uda(4)

DIAGNOSTICS
up%d%c: hard error sn%d cs2=%b er1=%b er2=%b. An unrecoverable error occurred during transfer of the specified sector in the specified disk partition. The contents of the cs2, er1 and er2 registers are printed in octal and symbolically with bits decoded. The error was either unrecoverable, or a large number of retry attempts (including offset positioning and drive recalibration) could not recover the error.

up%d: write locked. The write protect switch was set on the drive when a write was attempted. The write operation is not recoverable.

up%d: not ready. The drive was spun down or off line when it was accessed. The i/o operation is not recoverable.

up%d: not ready (flakey). The drive was not ready, but after printing the message about being not ready (which takes a fraction of a second) was ready. The operation is recovered if no further errors occur.

up%d%c: soft ecc sn%d. A recoverable ECC error occurred on the specified sector of the specified disk partition. This happens normally a few times a week. If it happens more frequently than this the sectors where the errors are occurring should be checked to see if certain cylinders on the pack, spots on the carriage of the drive or heads are indicated.

sc%d: lost interrupt. A timer watching the controller detecting no interrupt for an extended period while an operation was outstanding. This indicates a hardware or software failure. There is currently a hardware/software problem with spinning down drives while they are being accessed which causes this error to occur. The error causes a UNIBUS reset, and retry of the pending operations. If the controller continues to lose interrupts, this error will recur a few seconds later.

BUGS
In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

A program to analyze the logged error information (even in its present reduced form) is needed.

The partition tables for the file systems should be read off of each pack, as they are never quite what any single installation would prefer, and this would make packs more portable.
NAME
ut - UNIBUS TU45 tri-density tape drive interface

SYNOPSIS
controller ut0 at uba0 csr 0172440 vector utintr
tape tj0 at ut0 drive 0

DESCRIPTION
The ut interface provides access to a standard tape drive interface as describe in mtio(4). Hardware implementing this on the VAX is typified by the System Industries SI 9700 tape subsystem. Tapes may be read or written at 800, 1600, and 6250 bpi.

SEE ALSO
mt(1), mtio(4)

DIAGNOSTICS

tj%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

tj%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

tj%d: can't change density in mid-tape. An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

ut%d: soft error bn%db cs1=%b er=%b cs2=%b ds=%b. The formatter indicated a corrected error at a density other than 800bpi. The data transferred is assumed to be correct.

ut%d: hard error bn%db cs1=%b er=%b cs2=%b ds=%b. A tape error occurred at block bn. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.

tj%d: lost interrupt. A tape operation did not complete within a reasonable time, most likely because the tape was taken off-line during rewind or lost vacuum. The controller should, but does not, give an interrupt in these cases. The device will be made available again after this message, but any current open reference to the device will return an error as the operation in progress aborts.

BUGS
If any non-data error is encountered on non-raw tape, it refuses to do anything more until closed.
NAME
uu – TU58/DECtape II UNIBUS cassette interface

SYNOPSIS
options UUDMA
device uu0 at uba0 csr 0176500 vector uurintr uuxintr

DESCRIPTION
The uu device provides access to dual DEC TU58 tape cartridge drives connected to the
UNIBUS via a DL11-W interface module.

The interface supports only block i/o to the TU58 cassettes. The drives are normally manipu-
lated with the arff(8V) program using the “m” and “f” options.

The driver provides for an optional write and verify (read after write) mode that is activated
by specifying the “a” device.

The TU58 is treated as a single device by the system even though it has two separate drives,
“uu0” and “uu1”. If there is more than one TU58 unit on a system, the extra drives are
named “uu2”, “uu3” etc.

NOTES
Assembly language code to assist the driver in handling the receipt of data (using a pseudo-
dma approach) should be included when using this driver; specify “options UUDMA” in the
configuration file.

ERRORS
The following errors may be returned:
[ENXIO] Nonexistent drive (on open); offset is too large or bad (undefined) ioctl code.
[EIO] Open failed, the device could not be reset.
[EBUSY] Drive in use.

FILES
/dev/uu?
/dev/uu?a

SEE ALSO
tu(4), arff(8V)

DIAGNOSTICS
uu%d: timeout. A transmission complete interrupt was received with no outstanding
i/o request. This indicates a hardware problem.

uu%d protocol error, state=%s, op=%x, cnt=%d, block=%d. The driver entered an illegal
state. The information printed indicates the illegal state, the operation currently being exe-
cuted, the i/o count, and the block number on the cassette.

uu%d: break received, transfer restarted. The TU58 was sending a continuous break signal
and had to be reset. This may indicate a hardware problem, but the driver will attempt to
recover from the error.

uu%d receive state error, state=%s, byte=%x. The driver entered an illegal state in the
receiver finite state machine. The state is shown along with the control byte of the received
packet.

uu%d: read stalled. A timer watching the controller detected no interrupt for an extended
period while an operation was outstanding. This usually indicates that one or more receiver
interrupts were lost and the transfer is restarted.

uu%d: hard error bn%d, pk_mod %. The device returned a status code indicating a hard
error. The actual error code is shown in octal. No retries are attempted by the driver.
NAME
va – Benson-Varian interface

SYNOPSIS
controller va0 at uba0 csr 0164000 vector vaint
disk vz0 at va0 drive 0

DESCRIPTION
(NO'TE: the configuration description, while counter-intuitive, is actually as shown above.)
The Benson-Varian printer/plotter is normally used with the line printer system. This
description is designed for those who wish to drive the Benson-Varian directly.

In print mode, the Benson-Varian uses a modified ASCII character set. Most control charac-
ters print various non-ASCII graphics such as daggers, sigmas, copyright symbols, etc. Only
LF and FF are used as format effectors. LF acts as a newline, advancing to the beginning of
the next line, and FF advances to the top of the next page.

In plot mode, the Benson-Varian prints one raster line at a time. An entire raster line of bits
(2112 bits = 264 bytes) is sent, and then the Benson-Varian advances to the next raster line.

Note: The Benson-Varian must be sent an even number of bytes. If an odd number is sent,
the last byte will be lost. Nulls can be used in print mode to pad to an even number of bytes.

To use the Benson-Varian yourself, you must realize that you cannot open the device,
/dev/va0 if there is a daemon active. You can see if there is an active daemon by doing a
lpq(1) and seeing if there are any files being printed. Printing should be turned off using
lpc(8).

To set the Benson-Varian into plot mode include the file <sys/vcmd.h> and use the following
ioctl(2) call
ioctl(fileno(va), VSETSTATE, plotmd);
where plotmd is defined to be
int plotmd[] = { VPLOT, 0, 0 };
and va is the result of a call to fopen on stdin. When you finish using the Benson-Varian in
plot mode you should advance to a new page by sending it a FF after putting it back into
print mode, i.e. by
int prtmd[] = { VPRINT, 0, 0 };
...           
fflush(va);
ioctl(fileno(va), VSETSTATE, prtmd);
write(fileno(va), "\n\n0", 2);

FILES
/dev/va0

SEE ALSO
vfont(5), lpr(1), lpd(8), vp(4)

DIAGNOSTICS
The following error numbers are significant at the time the device is opened.
[ENXIO] The device is already in use.
[EIO] The device is offline.
The following message may be printed on the console.
va%d: npr timeout. The device was not able to get data from the UNIBUS within the timeout
period, most likely because some other device was hogging the bus. (But see BUGS below).
BUGS

The 1's (one's) and l's (lower-case el's) in the Benson-Varian's standard character set look very similar; caution is advised.

The interface hardware is rumored to have problems which can play havoc with the UNIBUS. We have intermittent minor problems on the UNIBUS where our va lives, but haven't ever been able to pin them down completely.
NAME
vp - Versatec interface

SYNOPSIS
device vp0 at uba0 csr 0177510 vector vpintr vpintr

DESCRIPTION
The Versatec printer/plotter is normally used with the line printer system. This description
is designed for those who wish to drive the Versatec directly.

To use the Versatec yourself, you must realize that you cannot open the device, /dev/vp0 if
there is a daemon active. You can see if there is a daemon active by doing a lpq(1), and see­
ing if there are any files being sent. Printing should be turned off using lpc(8).

To set the Versatec into plot mode you should include <sys/vcmd.h> and use the ioctl(2) call
ioctl(fileno(vp), VSETSTATE, plotmd);

where plotmd is defined to be
int plotmd[] = { VPLOT, 0, 0 };

and vp is the result of a call to fopen on stdio. When you finish using the Versatec in plot
mode you should eject paper by sending it a EOT after putting it back into print mode, i.e. by

int prtmd[] = { VPRINT, 0, 0 };

... flush(vp);
ioctl(fileno(vp), VSETSTATE, prtmd);
write(fileno(vp), "\04", 1);

FILES
/dev/vp0

SEE ALSO
vfont(5), lpr(1), lpd(8), vtroff(1), va(4)

DIAGNOSTICS
The following error numbers are significant at the time the device is opened.
[ENXIO] The device is already in use.
[EIO] The device is offline.

BUGS
The configuration part of the driver assumes that the device is set up to vector print mode
through 0174 and plot mode through 0200. As the configuration program can't be sure which
vector interrupted at boot time, we specify that it has two interrupt vectors, and if an inter­
rupt comes through 0200 it is reset to 0174. This is safe for devices with one or two vectors
at these two addresses. Other configurations with 2 vectors may require changes in the driver.
NAME

vv - Proteon proNET 10 Megabit ring

SYNOPSIS

device vv0 at uba0 csr 0161000 vector vrint vvxint

DESCRIPTION

The vv interface provides access to a 10 Mb/s Proteon proNET ring network.

The network address of the interface must be specified with an SIOCSIFADDR ioctl before data can be transmitted or received. It is only permissible to change the network address while the interface is marked "down".

The host's hardware address is discovered by putting the interface in digital loopback mode (not joining the ring) and sending a broadcast packet from which the hardware address is extracted.

Transmit timeouts are detected through use of a watchdog routine. Lost input interrupts are checked for when packets are sent out.

If the installation is running CTL boards which use the old broadcast address of 0 instead of the new address of 0xff, the define OLD_BROADCAST should be specified in the driver.

The driver can use "trailer" encapsulation to minimize copying data on input and output. This may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS

vv%d: host %. The software announces the host address discovered during autoconfiguration.

vv%d: can't initialize. The software was unable to discover the address of this interface, so it deemed "dead" will not be enabled.

vv%d: error vvocr=%b. The hardware indicated an error on the previous transmission.

vv%d: output timeout. The token timer has fired and the token will be recreated.

vv%d: error vvice=%b. The hardware indicated an error in reading a packet off the ring.

en%d: can't handle af%. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

vv%d: vs_olen=%d. The ring output routine has been handed a message with a preposterous length. This results in an immediate panic: vs_olen.

SEE ALSO

intro(4N), inet(4F)

BUGS

The encapsulation of trailer packets in the 4.2BSD version of this driver was incorrect (the packet type was in VAX byte order). As a result, the trailer encapsulation in this version is not compatible with the 4.2BSD VAX version.
L-DEVICES(5)        UNIX Programmer's Manual        L-DEVICES(5)

NAME
L-devices – UUCP device description file

DESCRIPTION
The L-devices file is consulted by the UUCP daemon uucico(8C) under the direction of
L.sys(5) for information on the devices that it may use. Each line describes exactly one devi-

A line in L-devices has the form:
Caller Device Call_Unit Class Dialer [Expect Send]....

Each item can be separated by any number of blanks or tabs. Lines beginning with a ‘#’ char-

 Caller denotes the type of connection, and must be one of the following:
 ACU  Automatic call unit, e.g., autodialing modems such as the Hayes Smartmodem 1200 or Novation “Smart Cat”.
 DIR  Direct connect; hardwired line (usually RS-232) to a remote system.
 DK   AT&T Datakit.
 MICOM  Micom Terminal switch.
 PAD  X.25 PAD connection.
 PCP  GTE Telenet PC Pursuit.
 SYTEK   Sytek high-speed dedicated modem port connection.
 TCP  Berkeley TCP/IP or 3Com UNET connection. These are mutually exclusive. Note that listing TCP connections in L-devices is superfluous; uucico does not even bother to look here since it has all the information it needs in L.sys(5).

 Device is a device file in /dev/ that is opened to use the device. The device file must be owned by UUCP, with access modes of 0600 or better. (See chmod(2)).

 Call_Unit is an optional second device file name. True automatic call units use a separate device file for data and for dialing; the Device field specifies the data port, while the Call_unit field specifies the dialing port. If the Call_unit field is unused, it must not be left empty.
 Insert a dummy entry as a placeholder, such as “0” or “unused.”

 Class is an integer number that specifies the line baud (for dialers and direct lines) or the port number (for network connections).

 The Class may be preceded by a non-numeric prefix. This is to differentiate among devices that have identical Caller and baud, but are distinctly different. For example, “1200” could refer to all Bell 212-compatible modems, “V1200” to Racal-Vadic modems, and “C1200” to CCITT modems, all at 1200 baud. Similarly, “W1200” could denote long distance lines, while “L1200” could refer to local phone lines.

 Dialer applies only to ACU devices. This is the “brand” or type of the ACU or modem.

 DF02  DEC DF02 or DF03 modems.
 DF112 Dec DF112 modems. Use a Dialer field of DF112T to use tone dialing, or DF112P for pulse dialing.
 att   AT&T 2224 2400 baud modem.
 cds224 Concord Data Systems 224 2400 baud modem.

4.3 Berkeley Distribution April 24, 1986
dn11  DEC DN11 Unibus dialer.

hayes  Hayes Smartmodem 1200 and compatible autodialing modems. Use a Dialer field of hayestone to use tone dialing, or hayespulse for pulse dialing. It is also permissible to include the letters ‘T’ and ‘P’ in the phone number (in L.sys) to change to tone or pulse midway through dialing. (Note that a leading ‘T’ or ‘P’ will be interpreted as a dialcode!)

hayes2400  Hayes Smartmodem 2400 and compatible modems. Use a Dialer field of hayes2400tone to use tone dialing, or hayes2400pulse for pulse dialing.

novation  Novation “Smart Cat” autodialing modem.

penril  Penril Corp “Hayes compatible” modems (they really aren’t or they would use the hayes entry.)

rvmacs  Racal-Vadic 820 dialer with 831 adapter in a MACS configuration.

va212  Racal-Vadic 212 autodialing modem.

va811s  Racal-Vadic 811s dialer with 831 adapter.

va820  Racal-Vadic 820 dialer with 831 adapter.

vadic  Racal-Vadic 3450 and 3451 series autodialing modems.

ventel  Ventel 212+ autodialing modem.

vmacs  Racal-Vadic 811 dialer with 831 adapter in a MACS configuration.

Expect/Send is an optional Expect/Send script for getting through a smart port selector, or for issuing special commands to the modem. The syntax is identical to that of the Expect/Send script of L.sys. The difference is that the L-devices script is used before the connection is made, while the L.sys script is used after.

FILES
/usr/lib/uucp/L-devices
/usr/lib/uucp/UUAIDS/L-devices  L-devices example

SEE ALSO
uucp(1C), uux(1C), L.sys(5), uucico(8C)
NAME
L-dialcodes - UUCP phone number index file

DESCRIPTION
The `L-dialcodes` file defines the mapping of strings from the phone number field of `L.sys(5)` to actual phone numbers.

Each line in L-dialcodes has the form:

```
alpha_string  phone_number
```

The two items can be separated by any number of blanks or tabs. Lines beginning with a `#` character are comments.

A phone number in `L.sys` can be preceded by an arbitrary alphabetic character string; the string is matched against the list of `alpha_strings` in `L-dialcodes`. If a match is found, `phone_number` is substituted for it. If no match is found, the string is discarded.

`L-dialcodes` is commonly used either of two ways:

1. The alphabetic strings are used as prefixes to denote area codes, zones, and other commonly used sequences. For example, if `L-dialcodes` included the following lines:

   ```
   chi  1312
   mv   1415
   ```

   In `L.sys` you could enter:

   ```
   chivax Any ACU 1200 chi5551234 ogin:-ogin: nuucp
   mvpyr Any ACU 1200 mv5556001 ogin:-ogin: Uuucp
   ```

   instead of

   ```
   chivax Any ACU 1200 13125551234 ogin:-ogin: nuucp
   mvpyr Any ACU 1200 14155556001 ogin:-ogin: Uuucp
   ```

2. All phone numbers are placed in `L-dialcodes`, one for each remote site. `L.sys` then refers to these by name. For example, if `L-dialcodes` contains the following lines:

   ```
   chivax 13125551234
   mvpyr 14155556001
   ```

   then `L.sys` could have:

   ```
   chivax Any ACU 1200 chivax ogin:-ogin: nuucp
   mvpyr Any ACU 1200 mvpyr ogin:-ogin: Uuucp
   ```

   This scheme allows a site administrator to give users read access to the table of phone numbers, while still protecting the login/password sequences in `L.sys`.

FILES
/usr/lib/uucp/L-dialcodes
/usr/lib/uucp/UUAIDS/L-dialcodes  L-dialcodes example

SEE ALSO
uucp(1C), uux(1C), L.sys(5), uucico(8C).
NAME
L.aliases – UUCP hostname alias file

DESCRIPTION
The L.aliases file defines mapping (aliasing) of system names for uucp. This is intended for compensating for systems that have changed names, or do not provide their entire machine name (like most USG systems). It is also useful when a machine's name is not obvious or commonly misspelled.

Each line in L.aliases is of the form:

    real_name alias_name

Any amount of whitespace may separate the two items. Lines beginning with a "#" character are comments.

All occurrences of alias_name are mapped to real_name by uucico(8C), uucp(1), and uux(1). The mapping occurs regardless of whether the name was typed in by a user or provided by a remote site. An exception is the -s option of uucico; only the site's real hostname (the name in L.sys(5)) will be accepted there.

Aliased system names should not be placed in L.sys; they will not be used.

FILES
/usr/lib/uucp/L.aliases /usr/lib/uucp/UUAIDS/L.aliases  L.aliases example

SEE ALSO
uucp(1C), uux(1C), L.sys(5), uucico(8C)
NAME

L.cmds – UUCP remote command permissions file

DESCRIPTION

The *L.cmds* file contains a list of commands, one per line, that are permitted for remote execution via *uux*(1C).

The default search path is */bin:/usr/bin:/usr/ucb*. To change the path, include anywhere in the file a line of the form:

```
PATH=/bin:/usr/bin:/usr/ucb
```

Normally, an acknowledgment is mailed back to the requesting site after the command completes. If a command name is suffixed with *Error*, then an acknowledgment will be mailed only if the command fails. If the command is suffixed with *No*, then no acknowledgment will ever be sent. (These correspond with the *-z* and *-n* options of *uux*, respectively.)

For most sites, *L.cmds* should only include the lines:

```
rm
rnu
```

News sites should add:

```
PATH=/bin:/usr/bin:/usr/ucb:/usr/new
rm,new,Error
```

While file names supplied as arguments to *uux* commands will be checked against the list of accessible directory trees in *USERFILE*(5), this check can be easily circumvented and should not be depended upon. In other words, it is unwise to include any commands in *L.cmds* that accept local file names. In particular, *sh*(1) and *csh*(1) are extreme risks.

It is common (but hazardous) to include *uucp*(1C) in *L.cmds*; see the NOTES section of *USERFILE*.

FILES

*/usr/lib/uucp/L.cmds*
*/usr/lib/uucp/UUAIDS/L.cmds*  *L.cmds* example.

SEE ALSO

*uucp*(1C), *uux*(1C), *USERFILE*(5), *uucico*(8C), *uuxqt*(8C)
NAME
L.sys – UUCP remote host description file

DESCRIPTION
The L.sys file is consulted by the UUCP daemon uucico(8C) for information on remote systems. L.sys includes the system name, appropriate times to call, phone numbers, and a login and password for the remote system. L.sys is thus a privileged file, owned by the UUCP Administrator; it is accessible only to the Administrator and to the superuser.

Each line in L.sys describes one connection to one remote host, and has the form:
System Times Caller Class Device/Phone_Number [Expect Send]....

Fields can be separated by any number of blanks or tabs. Lines beginning with a ‘#’ character are comments; long lines can be continued by appending a ‘\’ character to the end of the line.

The first five fields (System through Device/Phone_Number) specify the hardware mechanism that is necessary to make a connection to a remote host, such as a modem or network. Uucico searches from the top down through L.sys to find the desired System; it then opens the L-devices(5) file and searches for the first available device with the same Caller, Class, and (possibly) Device. (“Available” means that the device is ready and not being used for something else.) Uucico attempts a connection using that device; if the connection cannot be made (for example, a dialer gets a busy signal), uucico tries the next available device. If this also fails, it returns to L.sys to look for another line for the same System. If none is found, uucico gives up.

System is the hostname of the remote system. Every machine with which this system communicates via UUCP should be listed, regardless of who calls whom. Systems not listed in L.sys will not be permitted a connection. The local hostname should not appear here for security reasons.

Times is a comma-separated list of the times of the day and week that calls are permitted to this System. Times is most commonly used to restrict long distance telephone calls to those times when rates are lower. List items are constructed as:
keywordhhmm-hhmm/grade;retry_time

Keyword is required, and must be one of:

Any Any time, any day of the week.
Wk Any weekday. In addition, Mo, Tu, We, Th, Fr, Sa, and Su can be used for Monday through Sunday, respectively.

Evening When evening telephone rates are in effect, from 1700 to 0800 Monday through Friday, and all day Saturday and Sunday. Evening is the same as Wk1700-0800,Sa,Su.

Night When nighttime telephone rates are in effect, from 2300 to 0800 Monday through Friday, all day Saturday, and from 2300 to 1700 Sunday. Night is the same as Any2300-0800,Sa,Su0800-1700.

NonPeak This is a slight modification of Evening. It matches when the USA X.25 carriers have their lower rate period. This is 1800 to 0700 Monday through Friday, and all day Saturday and Sunday. NonPeak is the same as Any1800-0700,Sa,Su.

Never Never call; calling into this System is forbidden or impossible. This is intended for polled connections, where the remote system calls into the local machine periodically. This is necessary when one of the machines is lacking either dial-in or dial-out modems.
The optional hhmm-hhmm subfield provides a time range that modifies the keyword. hhmm refers to hours and minutes in 24-hour time (from 0000 to 2359). The time range is permitted to "wrap" around midnight, and will behave in the obvious way. It is invalid to follow the Evening, NonPeak, and Night keywords with a time range.

The grade subfield is optional; if present, it is composed of a '/' (slash) and single character denoting the grade of the connection, from 0 to 9, A to Z, or a to z. This specifies that only requests of grade grade or better will be transferred during this time. (The grade of a request or job is specified when it is queued by uucp or uux.) By convention, mail is sent at grade C, news is sent at grade d, and uucp copies are sent at grade n. Unfortunately, some sites do not follow these conventions, so it is not 100% reliable.

The retry_time subfield is optional; it must be preceded by a ';' (semicolon) and specifies the time, in minutes, before a failed connection may be tried again. (This restriction is in addition to any constraints imposed by the rest of the Time field.) By default, the retry time starts at 10 minutes and gradually increases at each failure, until after 26 tries uucico gives up completely (MAX RETRIES). If the retry time is too small, uucico may run into MAX RETRIES too soon.

Caller is the type of device used:

ACU Automatic call unit or auto-dialing modem such as the Hayes Smartmodem 1200 or Novation "Smart Cat". See L-devices for a list of supported modems.

DIR Direct connect; hardwired line (usually RS-232) to a remote system.

MICOM Micom Terminal Switch.

PAD X.25 PAD connection.

PCP GTE Telenet PC Pursuit. See L-devices for configuration details.

SYTEK Sytek high-speed dedicated modem port connection.

TCP Berkeley TCP/IP or 3Com UNET connection. These are mutually exclusive. TCP ports do not need entries in L-devices since all the necessary information is contained in L.sys. If several alternate ports or network connections should be tried, use multiple L.sys entries.

Class is usually the speed (baud) of the device, typically 300, 1200, or 2400 for ACU devices and 9600 for direct lines. Valid values are device dependent, and are specified in the L-devices file.

On some devices, the baud may be preceded by a non-numeric prefix. This is used in L-devices to distinguish among devices that have identical Caller and baud, but yet are distinctly different. For example, 1200 could refer to all Bell 212-compatible modems, V1200 to Racal-Vadic modems, and C1200 to CCITT modems, all at 1200 baud.

On TCP connections, Class is the port number (an integer number) or a port name from /etc/services that is used to make the connection. For standard Berkeley TCP/IP, UUCP normally uses port number 540.

Device/Phone_Number varies based on the Caller field. For ACU devices, this is the phone number to dial. The number may include: digits 0 through 9; # and * for dialing those symbols on tone telephone lines; - (hyphen) to pause for a moment, typically two to four seconds; = (equal sign) to wait for a second dial tone (implemented as a pause on many modems). Other characters are modem dependent; generally standard telephone punctuation characters (such as the slash and parentheses) are ignored, although uucico does not guarantee this.

The phone number can be preceded by an alphabetic string; the string is indexed and converted through the L-dialcodes(5) file.
For DIR devices, the Device/Phone_Number field contains the name of the device in /dev that is used to make the connection. There must be a corresponding line in L-devices with identical Caller, Class, and Device fields.

For TCP and other network devices, Device/Phone_Number holds the true network name of the remote system, which may be different from its UUCP name (although one would hope not).

*Expect* and *Send* refer to an arbitrarily long set of strings that alternately specify what to *expect* and what to *send* to login to the remote system once a physical connection has been established. A complete set of expect/send strings is referred to as an *expect/send script*. The same syntax is used in the L-devices file to interact with the dialer prior to making a connection; there it is referred to as a *chat script*. The complete format for one *expect/send* pair is:

```
expect-timeout-send-expect-timeout send
```

*Expect* and *Send* are character strings. *Expect* is compared against incoming text from the remote host; *send* is sent back when *expect* is matched. By default, the *send* is followed by a '\r' (carriage return). If the *expect* string is not matched within *timeout* seconds (default 45), then it is assumed that the match failed. The 'expect-send-expect' notation provides a limited loop mechanism; if the first *expect* string fails to match, then the *send* string between the hyphens is transmitted, and *uucico* waits for the second *expect* string. This can be repeated indefinitely. When the last *expect* string fails, *uucico* hangs up and logs that the connection failed.

The timeout can (optionally) be specified by appending the parameter "nn" to the *expect* string, when *nn* is the timeout time in seconds.

Backslash escapes that may be imbedded in the *expect* or *send* strings include:

- `\b` Generate a 3/10 second BREAK.
- `\bn` Where *n* is a single-digit number; generate an *n/10* second BREAK.
- `\c` Suppress the \r at the end of a *send* string.
- `\d` Delay; pause for 1 second. (*Send* only.)
- `\r` Carriage Return.
- `\s` Space.
- `\n` Newline.
- `\xxx` Where *xxx* is an octal constant; denotes the corresponding ASCII character.

As a special case, an empty pair of double-quotes "" in the *expect* string is interpreted as "expect nothing"; that is, transmit the *send* string regardless of what is received. Empty double-quotes in the *send* string cause a lone '\r' (carriage return) to be sent.

One of the following keywords may be substituted for the *send* string:

- **BREAK** Generate a 3/10 second BREAK.
- **BREAK*n** Generate an *n/10* second BREAK.
- **CR** Send a Carriage Return (same as '"').
- **EOT** Send an End-Of-Transmission character, ASCII \004.
  Note that this will cause most hosts to hang up.
- **NL** Send a Newline.
- **PAUSE** Pause for 3 seconds.
- **PAUSE*n** Pause for *n* seconds.
- **P_ODD** Use odd parity on future send strings.
- **P_ONE** Use parity one on future send strings.
- **P_EVEN** Use even parity on future send strings. (Default)
- **P_ZERO** Use parity zero on future send strings.
Finally, if the `expect` string consists of the keyword **ABORT**, then the string following is used to arm an abort trap. If that string is subsequently received any time prior to the completion of the entire `expect/send` script, then `uucico` will abort, just as if the script had timed out. This is useful for trapping error messages from port selectors or front-end processors such as "Host Unavailable" or "System is Down."

For example:

```
" " 'login:-login: nuucp ssword: ufeedme
```

This is executed as, "When the remote system answers, `expect` nothing. `Send` a carriage return. `Expect` the remote to transmit the string 'login:'. If it doesn't within 45 seconds, `send` another carriage return. When it finally does, `send` it the string 'nuucp'. Then `expect` the string 'ssword:'; when that is received, `send` 'ufeedme'."

### FILES

```
/usr/lib/uucp/L.sys
/usr/lib/uucp/UUAIDS/L.sys L.sys example
```

### SEE ALSO

`uucp(1C), uux(1C), L-devices(5), services(5), uucico(8C)`

### BUGS

"**ABORT**" in the `send/expect` script is expressed "backwards," that is, it should be written "`expect ABORT`" but instead it is "**ABORT expect**".

Several of the backslash escapes in the `send/expect` strings are confusing and/or different from those used by AT&T and Honey-Danber UUCP. For example, \b requests a BREAK, while practically everywhere else \b means backspace. \t for tab and \f for formfeed are not implemented. \s is a kludge; it would be more sensible to be able to delimit strings with quotation marks.
NAME
 USERFILE – UUCP pathname permissions file

DESCRIPTION
 The USERFILE file specifies the file system directory trees that are accessible to local users and to remote systems via UUCP.

Each line in USERFILE is of the form:

[loginname],[system] [ c ] pathname [ pathname ] [ pathname ]

The first two items are separated by a comma; any number of spaces or tabs may separate the remaining items. Lines beginning with a ‘#’ character are comments. A trailing ‘\’ indicates that the next line is a continuation of the current line.

Loginname is a login (from /etc/passwd) on the local machine.

System is the name of a remote machine, the same name used in L.sys(5).

c denotes the optional callback field. If a c appears here, a remote machine that calls in will be told that callback is requested, and the conversation will be terminated. The local system will then immediately call the remote host back.

Pathname is a pathname prefix that is permissible for this login and/or system.

When uucico(8C) runs in master role or uucp(1C) or uux(1C) are run by local users, the permitted pathnames are those on the first line with a loginname that matches the name of the user who executed the command. If no such line exists, then the first line with a null (missing) loginname field is used. (Beware: uucico is often run by the superuser or the UUCP administrator through cron(8).)

When uucico runs in slave role, the permitted pathnames are those on the first line with a system field that matches the hostname of the remote machine. If no such line exists, then the first line with a null (missing) system field is used.

Uuxqt(8) works differently; it knows neither a login name nor a hostname. It accepts the pathnames on the first line that has a null system field. (This is the same line that is used by uucico when it cannot match the remote machine’s hostname.)

A line with both loginname and system null, for example

   , /usr/spool/uucppub

can be used to conveniently specify the paths for both “no match” cases if lines earlier in USERFILE did not define them. (This differs from older Berkeley and all USG versions, where each case must be individually specified. If neither case is defined earlier, a “null” line only defines the “unknown login” case.)

To correctly process loginname on systems that assign several logins per UID, the following strategy is used to determine the current loginname:

1) If the process is attached to a terminal, a login entry exists in /etc/utmp, and the UID for the utmp name matches the current real UID, then loginname is set to the utmp name.

2) If the USER environment variable is defined and the UID for this name matches the current real UID, then loginname is set to the name in USER.

3) If both of the above fail, call getpwuid(3) to fetch the first name in /etc/passwd that matches the real UID.

4) If all of the above fail, the utility aborts.

FILES
 /usr/lib/uucp/USERFILE
/usr/lib/uucp/UUAIDS/USERFILE USERFILE example

SEE ALSO
uucp(1C), uux(1C), L.cmds(5), L.sys(5), uucico(8C), uuxqt(8C)

NOTES
The UUCP utilities (uucico, uucp, uux, and uuxqt) always have access to the UUCP spool files in /usr/spool/uucp, regardless of pathnames in USERFILE.

If uucp is listed in L.cmds(5), then a remote system will execute uucp on the local system with the USERFILE privileges for its login, not its hostname.

Uucico freely switches between master and slave roles during the course of a conversation, regardless of the role it was started with. This affects how USERFILE is interpreted.

WARNING
USERFILE restricts access only on strings that the UUCP utilities identify as being pathnames. If the wrong holes are left in other UUCP control files (notably L.cmds), it can be easy for an intruder to open files anywhere in the file system. Arguments to uucp(1C) are safe, since it assumes all of its non-option arguments are files. Uux(1C) cannot make such assumptions; hence, it is more dangerous.

BUGS
The UUCP Implementation Description explicitly states that all remote login names must be listed in USERFILE. This requirement is not enforced by Berkeley UUCP, although it is by USG UUCP.

Early versions of 4.2BSD uuxqt(8) erroneously check UUCP spool files against the USERFILE pathname permissions. Hence, on these systems it is necessary to specify /usr/spool/uucp as a valid path on the USERFILE line used by uuxqt. Otherwise, all uux(1C) requests are rejected with a "PERMISSION DENIED" message.
NAME
a.out — assembler and link editor output

SYNOPSIS
#include <a.out.h>

DESCRIPTION
A.out is the output file of the assembler as(1) and the link editor ld(1). Both programs make
a.out executable if there were no errors and no unresolved external references. Layout information as given in the include file for the VAX-11 is:

/*
 * Header prepended to each a.out file.
 */
struct exec {
    long a_magic;  /* magic number */
    unsigned a_text;  /* size of text segment */
    unsigned a_data;  /* size of initialized data */
    unsigned a_bss;  /* size of uninitialized data */
    unsigned a_syms;  /* size of symbol table */
    unsigned a_entry;  /* entry point */
    unsigned a_trsize;  /* size of text relocation */
    unsigned a_drsize;  /* size of data relocation */
};

#define OMAGIC0407 /* old impure format */
#define NMAGIC0410 /* read-only text */
#define ZMAGIC0413 /* demand load format */

/*
 * Macros which take exec structures as arguments and tell whether
 * the file has a reasonable magic number or offsets to text | symbols | strings.
 */
#define N_BADMAG(x) 
    (((x).a_magic)!=OMAGIC && ((x).a_magic)!=NMAGIC && (x).a_magic)!=ZMAGIC)
#define N_TXTOFF(x) 
    ((x).a_magic==ZMAGIC ? 1024: sizeof (struct exec))
#define N_SYMOFF(x) 
    (N_TXTOFF(x) + (x).a_text+(x).a_data + (x).a_trsize+(x).a_drsize)
#define N_STROFF(x) 
    (N_SYMOFF(x) + (x).a_syms)

The file has five sections: a header, the program text and data, relocation information, a symbol table and a string table (in that order). The last three may be omitted if the program was loaded with the '-s' option of ld or if the symbols and relocation have been removed by strip(1).

In the header the sizes of each section are given in bytes. The size of the header is not included in any of the other sizes.

When an a.out file is executed, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized), and a stack. The text segment begins at 0 in the core image; the header is not loaded. If the magic number in the header is OMAGIC (0407), it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. This is the oldest kind of executable program and is rarely used. If the magic number
is NMAGIC (0410) or ZMAGIC (0413), the data segment begins at the first 0 mod 1024 byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same file, they will share the text segment. For ZMAGIC format, the text segment begins at a 0 mod 1024 byte boundary in the a.out file, the remaining bytes after the header in the first block are reserved and should be zero. In this case the text and data sizes must both be multiples of 1024 bytes, and the pages of the file will be brought into the running image as needed, and not pre-loaded as with the other formats. This is especially suitable for very large programs and is the default format produced by ld(1).

The stack will occupy the highest possible locations in the core image, growing downwards from USRSTACK (from <machine/vmparam.h>). The stack is automatically extended as requested by brk(2).

After the header in the file follow the text, data, text relocation data relocation, symbol table and string table in that order. The text begins at the byte 1024 in the file for ZMAGIC format or just after the header for the other formats. The N_TXTOFF macro returns this absolute file position when given the name of an exec structure as argument. The data segment is contiguous with the text and immediately followed by the text relocation and then the data relocation information. The symbol table follows all this; its position is computed by the N_SYMOFF macro. Finally, the string table immediately follows the symbol table at a position which can be gotten easily using N_STROFF. The first 4 bytes of the string table are not used for string storage, but rather contain the size of the string table; this size INCLUDES the 4 bytes, the minimum string table size is thus 4.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file as follows:

```c
/*
 * Format of a symbol table entry.
 */

struct nlist {
    union {
        char     *n_name; /* for use when in-core */
        long     n_strx; /* index into file string table */
    } n_un;
    unsigned char n_type; /* type flag, i.e. N_TEXT etc; see below */
    char      n_other;
    short     n_desc; /* see <stab.h> */
    unsigned  n_value; /* value of this symbol (or offset) */
};
#define N_hash n_desc /* used internally by ld */

/*
 * Simple values for n_type.
 */
#define N_UNDF 0x0  /* undefined */
#define N_ABS   0x2  /* absolute */
#define N_TEXT  0x4  /* text */
#define N_DATA  0x6  /* data */
#define N_BSS   0x8  /* bss */
#define N_COMM  0x12 /* common (internal to ld) */
#define N_FN    0x1f /* file name symbol */
#define N_EXT   0x1  /* external bit, or'ed in */
#define N_TYPE  0x1e /* mask for all the type bits */
```
• Other permanent symbol table entries have some of the N_STAB bits set.
• These are given in <stab.h>

#define N_STAB OxeO

• Format for namelist values.

#define N_FORMAT "%08x"

In the a.out file a symbol's n_un.n_strx field gives an index into the string table. A n_strx value of 0 indicates that no name is associated with a particular symbol table entry. The field n_un.n_name can be used to refer to the symbol name only if the program sets this up using n_strx and appropriate data from the string table.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader ld as the name of a common region whose size is indicated by the value of the symbol.

The value of a byte in the text or data which is not a portion of a reference to an undefined external symbol is exactly that value which will appear in memory when the file is executed. If a byte in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the bytes in the file.

If relocation information is present, it amounts to eight bytes per relocatable datum as in the following structure:

/*
• Format of a relocation datum.
*/
struct relocation_info {
    int r_address;        /* address which is relocated */
    unsigned r_symbolnum:24, /* local symbol ordinal */
    r_pcrel:1,           /* was relocated pc relative already */
    r_length:2,          /* 0=byte, 1=word, 2=long */
    rExtern:1,           /* does not include value of sym referenced */
    :4;                  /* nothing, yet */
};

There is no relocation information if a_trsize+a_drsz==0. If rExtern is 0, then r_symbolnum is actually a n_type for the relocation (i.e. N_TEXT meaning relative to segment text origin.)

SEE ALSO
adb(1), as(1), ld(1), nm(1), dbx(1), stab(5), strip(1)

BUGS
Not having the size of the string table in the header is a loss, but expanding the header size would have meant stripped executable file incompatibility, and we couldn't hack this just now.
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 Regents of the University of California.
 * All rights reserved. The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 * @(#)acct.h 6.4 (Berkeley) 10/28/85
 */

/* 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 I/O 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

#ifndef KERNEL
struct acct acctbuf;
struct inode *acctp;

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
aliases - aliases file for sendmail

SYNOPSIS
/usr/lib/aliases

DESCRIPTION
This file describes user id aliases used by /usr/lib/sendmail. It is formatted as a series of lines of the form

   name: name_1, name_2, name_3, ...

The name is the name to alias, and the name_n are the aliases for that name. Lines beginning with white space are continuation lines. Lines beginning with ‘#’ are comments.

Aliasing occurs only on local names. Loops can not occur, since no message will be sent to any person more than once.

After aliasing has been done, local and valid recipients who have a “.forward” file in their home directory have messages forwarded to the list of users defined in that file.

This is only the raw data file; the actual aliasing information is placed into a binary format in the files /usr/lib/aliases.dir and /usr/lib/aliases.pag using the program newaliases(1). A newaliases command should be executed each time the aliases file is changed for the change to take effect.

SEE ALSO
newaliases(1), dbm(3X), sendmail(8)
SENDMAIL Installation and Operation Guide.
SENDMAIL An Internetwork Mail Router.

BUGS
Because of restrictions in dbm(3X) a single alias cannot contain more than about 1000 bytes of information. You can get longer aliases by “chaining”; that is, make the last name in the alias be a dummy name which is a continuation alias.
NAME
    ar - archive (library) file format

SYNOPSIS
    #include <ar.h>

DESCRIPTION
    The archive command ar combines several files into one. Archives are used mainly as
    libraries to be searched by the link-editor ld.

    A file produced by ar has a magic string at the start, followed by the constituent files, each
    preceded by a file header. The magic number and header layout as described in the include
    file are:

        /*
           * Copyright (c) 1980 Regents of the University of California.
           * All rights reserved. The Berkeley software License Agreement
           * specifies the terms and conditions for redistribution.
           *
           * @(#)ar.h  5.1 (Berkeley) 5/30/85
           */

        #define ARMAG "!<arch>
        #define SARMAG 8
        #define ARFMAG "\n"

        struct ar_hdr {
            char  ar_name[16];
            char  ar_date[12];
            char  ar_uid[6];
            char  ar_gid[6];
            char  ar_mode[8];
            char  ar_size[10];
            char  ar_fmag[2];
        }

    The name is a blank-padded string. The ar_fmag field contains ARFMAG to help verify the
    presence of a header. The other fields are left-adjusted, blank-padded numbers. They are
decimal except for ar_mode, which is octal. The date is the modification date of the file at
the time of its insertion into the archive.

    Each file begins on a even (0 mod 2) boundary; a new-line is inserted between files if neces-
    sary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

    There is no provision for empty areas in an archive file.

    The encoding of the header is portable across machines. If an archive contains printable files,
the archive itself is printable.

SEE ALSO
    ar(1), ld(1), nm(1)

BUGS
    File names lose trailing blanks. Most software dealing with archives takes even an included
    blank as a name terminator.
NAME
core - format of memory image file

SYNOPSIS
#include <sys/param.h>

DESCRIPTION
The UNIX System writes out a memory image of a terminated process when any of various
errors occur. See sigvec(2) for the list of reasons; the most common are memory violations,
illegal instructions, bus errors, and user-generated quit signals. The memory image is called
'core' and is written in the process's working directory (provided it can be; normal access con­
trols apply).

The maximum size of a core file is limited by setrlimit(2). Files which would be larger than
the limit are not created.

The core file consists of the u. area, whose size (in pages) is defined by the UPAGES manifest
in the <sys/param.h> file. The u. area starts with a user structure as given in <sys/user.h>. The remainder of the core file consists first of the data pages and then the stack pages of the
process image. The amount of data space image in the core file is given (in pages) by the
variable u_dsize in the u. area. The amount of stack image in the core file is given (in pages)
by the variable u_ssize in the u. area. The size of a “page” is given by the constant NBPG
(also from <sys/param.h>).

In general the debugger adb(1) is sufficient to deal with core images.

SEE ALSO
adb(1), dbx(1), sigvec(2), setrlimit(2)
NAME
dbx – dbx symbol table information

DESCRIPTION
The compiler symbol information generated for \texttt{dbx(1)} uses the same structure as described in \textit{stab(5)}, with additional type and scope information appended to a symbol’s name. The assembler directive used to describe symbol information has the following format:

\begin{verbatim}
stabs "string",kind,0,size,value
\end{verbatim}

\textit{String} contains the name, source language type, and scope of the symbol, \textit{kind} specifies the memory class (e.g., external, static, parameter, local, register), and \textit{size} specifies the byte size of the object, if relevant. The third field (0 above) is unused. For a global variable or a type, \textit{value} is unused; for a local variable or parameter, it is the offset from the frame pointer, for a register variable, it is the associated register number.

The different kinds of stab entries are interpreted by \texttt{dbx} as follows:

\begin{itemize}
  \item \texttt{N\_GSYM} The symbol is a global variable (e.g., \texttt{.comm} variable). The variable’s address can be found from the corresponding \texttt{ld(1)} symbol entry, thus the value field for \texttt{N\_GSYM} symbols is ignored. For example, a global variable “x” will have both an \texttt{N\_GSYM} entry and an \texttt{ld(1)} entry (e.g., \texttt{N\_BSS + N\_EXT}). See \textit{a.out(5)} for details about these other entries. of
  \item \texttt{N\_FUN} The symbol is a procedure or function. The size field contains the line number of the entry point. The value field contains the address of the entry point (in the text segment).
  \item \texttt{N\_STSYM} The symbol is a statically allocated variable for which an initial value has been specified. The value field contains the address of the variable (in the data segment).
  \item \texttt{N\_LCSYM} The symbol is statically allocated, but not initialized.
  \item \texttt{N\_RSYM} The symbol is a register variable whose value is kept in the register denoted by the value field.
  \item \texttt{N\_PSYM} The symbol is a parameter whose value is pushed on the stack before the call. The value field contains the offset from the argument base pointer (on the VAX, the ap register).
  \item \texttt{N\_LSYM} The symbol is a local variable whose value is stored in the most recently defined procedure’s stack frame. The value is the (often negative) offset from the frame pointer (on the VAX, the fp register).
  \item \texttt{N\_PC, N\_MOD2} The symbol defines separate compilation information for pre-linking checking for Berkeley Pascal and DEC Modula-2 programs respectively. For Pascal, the value field contains the line number that the symbol is defined on. The value field is not used for Modula-2.
\end{itemize}

Most of the source level information about a symbol is stored in the string field of the stab entry. Since strings are kept in a separate string table in the \textit{a.out} file, they can be arbitrarily long. Thus there are no restrictions on the kind or length of information in the string field, and it was not necessary to modify the assembler or loader when extending or modifying the format of this information.
Below is a grammar describing the syntax of the symbol string. Except in the case of a constant whose value is a string, there are no blanks in a symbol string.

NAME: \([a-zA-Z_][a-zA-Z_0-9]*\)
INTEGER: \([-][0-9][0-9]*\)
REAL: \([+-][0-9]*([0-9]*|([eE][+-][0-9]*))\)
STRING: ".."
BSTRING: ".."

String:
  NAME `:` Class
  `:` Class

Class:
  `c` `=` Constant `;`
  Variable
  Procedure
  Parameter
  NamedType
  `X` ExportInfo – export or import information (for N_MOD2 only)

Constant:
  `i` INTEGER
  `r` REAL
  `c` OrdValue
  `b` OrdValue
  `s` STRING
  `e` Typeld `;`, OrdValue
  `S` Typeld `;`, NumElements `;`, NumBits `;`, BSTRING

OrdValue:
  INTEGER

NumElements:
  INTEGER

NumBits:
  INTEGER

Variable:
  Typeld – local variable of type Typeld
  `r` Typeld – register variable of type Typeld
  `S` Typeld – module variable of type Typeld (static global in C)
  `V` Typeld – own variable of type Typeld (static local in C)
  `G` Typeld – global variable of type Typeld

Procedure:
  Proc – top level procedure
  Proc `;`, NAME `;`, NAME – local to first NAME,
  – second NAME is corresponding id symbol
  Proc `P` – global procedure
Parameter:
 parch parameter of type TypeId
  'v' TypeId  -- reference parameter of type TypeId

NamedType:
  't' TypeId  -- type name for type TypeId
  'T' TypeId  -- C structure tag name for struct TypeId

TypeId:
  INTEGER  -- Unique (per compilation) number of type
  INTEGER '=' TypeDef  -- Definition of type number
  INTEGER '=' TypeAttrs TypeDef

- Type attributes are extra information associated with a type,
  such as alignment constraints or pointer checking semantics.
- Dbx interprets some of these, but will ignore rather than complain
  about any it does not recognize. Therefore this is a way to add
  extra information for pre-linking checking.

TypeAttrs:
  '@' TypeAttrList ';

TypeAttrList:
  TypeAttrList ',' TypeAttr
  TypeAttr

TypeAttr:
  'a' INTEGER  -- align boundary
  's' INTEGER  -- size in bits
  'p' INTEGER  -- pointer class (e.g., checking)
  BSTRING  -- something else

TypeDef:
  INTEGER
  Subrange
  Array
  Record
  'e' EnumList ';
  'a' TypeId  -- pointer to TypeId
  'S' TypeId  -- set of TypeId
  'd' TypeId  -- file of TypeId
  ProcedureType
  'i' NAME ':' NAME ';
  'o' NAME ';
  'i' NAME ':' NAME ',', TypeId ';
  'o' NAME ' ,', TypeId ';

4.3 Berkeley Distribution January 12, 1986
Subrange:
‘r’ TypeId ‘;’ INTEGER ‘;’ INTEGER

Array:
‘a’ TypeId ‘;’ TypeId — array [TypeId] of TypeId
‘A’ TypeId — open array of TypeId
‘D’ INTEGER ‘;’ TypeId — N-dim. dynamic array
‘E’ INTEGER ‘;’ TypeId — N-dim. subarray

ProcedureType:
‘f’ TypeId ‘;’ — C function type
‘f’ TypeId ‘;’ NumParams ‘;’ TParamList ‘;’
‘p’ NumParams ‘;’ TParamList ‘;’

NumParams:
INTEGER

Record:
‘s’ ByteSize FieldList ‘;’ — structure/record
‘u’ ByteSize FieldList ‘;’ — C union

ByteSize:
INTEGER

FieldList :
Field
FieldList Field

Field:
NAME ‘;’ TypeId ‘;’ BitOffset ‘;’ BitSize ‘;’

BitSize:
INTEGER

BitOffset:
INTEGER

EnumList:
Enum
EnumList Enum

Enum:
NAME ‘;’ OrdValue ‘;’

ParamList:
Param
ParamList Param

Param:
NAME ‘;’ TypeId ‘;’ PassBy ‘;’

PassBy:
INTEGER
TParam:
  TypeId ' ,' PassBy ' ,'

TParamList:
  TParam
  TParamList TParam

Export:
  INTEGER ExportInfo

ExportInfo:
  't' TypeId
  'f' TypeId ' ,' NumParams ' ,' ParamList ' ;'
  'p' NumParams ' ,' ParamList ' ;'
  'v' TypeId
  'c' '=' Constant

A '?' indicates that the symbol information is continued in the next stab entry. This directive can only occur where a ';' would otherwise separate the fields of a record or constants in an enumeration. It is useful when the number of elements in one of these lists is large.

SEE ALSO
  dbx(1), stab(5), a.out(5)
NAME
dir – format of directories

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

DESCRIPTION
A directory behaves exactly like an ordinary file, save that no user may write into a directory.
The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry; see fs(5). The structure of a directory entry as given in the include file is:

/ *
  * A directory consists of some number of blocks of DIRBLKSIZ
  * bytes, where DIRBLKSIZ is chosen such that it can be transferred
  * to disk in a single atomic operation (e.g. 512 bytes on most machines).
  *
  * Each DIRBLKSIZ byte block contains some number of directory entry
  * structures, which are of variable length. Each directory entry has
  * a struct direct at the front of it, containing its inode number,
  * the length of the entry, and the length of the name contained in
  * the entry. These are followed by the name padded to a 4 byte boundary
  * with null bytes. All names are guaranteed null terminated.
  * The maximum length of a name in a directory is MAXNAMLEN.
  *
  * The macro DIRSIZ(dp) gives the amount of space required to represent
  * a directory entry. Free space in a directory is represented by
  * entries which have dp->d_reclen > DIRSIZ(dp). All DIRBLKSIZ bytes
  * in a directory block are claimed by the directory entries. This
  * usually results in the last entry in a directory having a large
  * dp->d_reclen. When entries are deleted from a directory, the
  * space is returned to the previous entry in the same directory
  * block by increasing its dp->d_reclen. If the first entry of
  * a directory block is free, then its dp->d_ino is set to 0.
  * Entries other than the first in a directory do not normally have
  * dp->d_ino set to 0.
  */
#ifdef KERNEL
#define DIRBLKSIZ DEV_BSIZE
#else
#define DIRBLKSIZ 512
#endif

#define MAXNAMLEN 255

/ *
  * The DIRSIZ macro gives the minimum record length which will hold
  * the directory entry. This requires the amount of space in struct direct
  * without the d_name field, plus enough space for the name with a terminating
  * null byte (dp->d_namlen+1), rounded up to a 4 byte boundary.
  */
#undef DIRSIZ
#define DIRSIZ(dp) ((sizeof (struct direct) - (MAXNAMLEN+1)) + (((dp)->d_namlen+1 + 3) & 3))
struct direct {
    u_long    d_ino;
    short    d_reclen;
    short    d_namlen;
    char     d_name[MAXNAMLEN + 1];
/* typically shorter */
};

struct _dirdesc {
    int       dd_fd;
    long      dd_loc;
    long      dd_size;
    char      dd_buf[DIRBLKSIZ];
};

By convention, the first two entries in each directory are for ‘.’ and ‘.’. The first is an entry for the directory itself. The second is for the parent directory. The meaning of ‘.’ is modified for the root directory of the master file system (“/”), where ‘..’ has the same meaning as ‘.’.

SEE ALSO
    fs(5)
NAME
disktab - disk description file

SYNOPSIS
#include <disktab.h>

DESCRIPTION
Disktab is a simple date base which describes disk geometries and disk partition characteristics. The format is patterned after the termcap(5) terminal data base. Entries in disktab consist of a number of ':' separated fields. The first entry for each disk gives the names which are known for the disk, separated by '|' characters. The last name given should be a long name fully identifying the disk.

The following list indicates the normal values stored for each disk entry.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ns</td>
<td>num</td>
<td>Number of sectors per track</td>
</tr>
<tr>
<td>nt</td>
<td>num</td>
<td>Number of tracks per cylinder</td>
</tr>
<tr>
<td>nc</td>
<td>num</td>
<td>Total number of cylinders on the disk</td>
</tr>
<tr>
<td>ba</td>
<td>num</td>
<td>Block size for partition 'a' (bytes)</td>
</tr>
<tr>
<td>bd</td>
<td>num</td>
<td>Block size for partition 'd' (bytes)</td>
</tr>
<tr>
<td>be</td>
<td>num</td>
<td>Block size for partition 'e' (bytes)</td>
</tr>
<tr>
<td>bf</td>
<td>num</td>
<td>Block size for partition 'f' (bytes)</td>
</tr>
<tr>
<td>bg</td>
<td>num</td>
<td>Block size for partition 'g' (bytes)</td>
</tr>
<tr>
<td>bh</td>
<td>num</td>
<td>Block size for partition 'h' (bytes)</td>
</tr>
<tr>
<td>fa</td>
<td>num</td>
<td>Fragment size for partition 'a' (bytes)</td>
</tr>
<tr>
<td>fd</td>
<td>num</td>
<td>Fragment size for partition 'd' (bytes)</td>
</tr>
<tr>
<td>fe</td>
<td>num</td>
<td>Fragment size for partition 'e' (bytes)</td>
</tr>
<tr>
<td>ff</td>
<td>num</td>
<td>Fragment size for partition 'f' (bytes)</td>
</tr>
<tr>
<td>fg</td>
<td>num</td>
<td>Fragment size for partition 'g' (bytes)</td>
</tr>
<tr>
<td>fh</td>
<td>num</td>
<td>Fragment size for partition 'h' (bytes)</td>
</tr>
<tr>
<td>pa</td>
<td>num</td>
<td>Size of partition 'a' in sectors</td>
</tr>
<tr>
<td>pb</td>
<td>num</td>
<td>Size of partition 'b' in sectors</td>
</tr>
<tr>
<td>pc</td>
<td>num</td>
<td>Size of partition 'c' in sectors</td>
</tr>
<tr>
<td>pd</td>
<td>num</td>
<td>Size of partition 'd' in sectors</td>
</tr>
<tr>
<td>pe</td>
<td>num</td>
<td>Size of partition 'e' in sectors</td>
</tr>
<tr>
<td>pf</td>
<td>num</td>
<td>Size of partition 'f' in sectors</td>
</tr>
<tr>
<td>pg</td>
<td>num</td>
<td>Size of partition 'g' in sectors</td>
</tr>
<tr>
<td>ph</td>
<td>num</td>
<td>Size of partition 'h' in sectors</td>
</tr>
<tr>
<td>se</td>
<td>num</td>
<td>Sector size in bytes</td>
</tr>
<tr>
<td>sf</td>
<td>bool</td>
<td>supports bad144-style bad sector forwarding</td>
</tr>
<tr>
<td>so</td>
<td>bool</td>
<td>partition offsets in sectors</td>
</tr>
<tr>
<td>ty</td>
<td>str</td>
<td>Type of disk (e.g. removable, winchester)</td>
</tr>
</tbody>
</table>

Disktab entries may be automatically generated with the diskpart program.

FILES
/etc/disktab

SEE ALSO
newfs(8), diskpart(8), getdiskbyname(3)

BUGS
This file shouldn't exist, the information should be stored on each disk pack.
NAME
dump, dumpdates – incremental dump format

SYNOPSIS
#include <sys/types.h>
#include <sys/inode.h>
#include <protocols/dumprestore.h>

DESCRIPTION
Tapes used by dump and restore(8) contain:
a header record
two groups of bit map records
a group of records describing directories
a group of records describing files

The format of the header record and of the first record of each description as given in the include file <protocols/dumprestore.h> is:
#define NTREC 10
#define MLEN 16
#define MSIZ 4096
#define TS_TAPE 1
#define TS_INODE 2
#define TS_BITS 3
#define TS_ADDR 4
#define TS_END 5
#define TS_CLRI 6
#define MAGIC (int) 60011
#define CHECKSUM (int) 84446

struct spcl {
    int c_type;
    time_t c_date;
    time_t c_ddate;
    int c_volume;
    intaddr c_tapea;
    ino_t c_inumber;
    int c_magic;
    int c_checksum;
    struct dinode c_dinode;
    int c_count;
    char c_addr[BSIZE];
} spcl;

struct idates {
    char id_name[16];
    char id_incno;
    time_t id_ddate;
};

#define DUMPOUTFMT "%-16s %c %s" /* for printf */
#define DUMPINFMT "%-16s %c %r
" /* name, incno, ctime(date) */

NTREC is the number of 1024 byte records in a physical tape block. MLEN is the number of
bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the c_type field to indicate what sort of header this is. The types
and their meanings are as follows:

- **TS_TAPE**: Tape volume label
- **TS_INODE**: A file or directory follows. The c_dinode field is a copy of the disk inode and
  contains bits telling what sort of file this is.
- **TS_BITS**: A bit map follows. This bit map has a one bit for each inode that was dumped.
- **TS_ADDR**: A subrecord of a file description. See c_addr below.
- **TS_END**: End of tape record.
- **TS_CLRI**: A bit map follows. This bit map contains a zero bit for all inodes that were
  empty on the file system when dumped.
- **MAGIC**: All header records have this number in c_magic.
- **CHECKSUM**: Header records checksum to this value.

The fields of the header structure are as follows:

- **c_type**: The type of the header.
- **c_date**: The date the dump was taken.
- **c_ddate**: The date the file system was dumped from.
- **c_volume**: The current volume number of the dump.
- **c_tapea**: The current number of this (1024-byte) record.
- **c_inumber**: The number of the inode being dumped if this is of type TS_INODE.
- **c_magic**: This contains the value MAGIC above, truncated as needed.
- **c_checksum**: This contains whatever value is needed to make the record sum to CHECK-
  SUM.
- **c_dinode**: This is a copy of the inode as it appears on the file system; see fs(5).
- **c_count**: The count of characters in c_addr.
- **c_addr**: An array of characters describing the blocks of the dumped file. A character is
  zero if the block associated with that character was not present on the file sys-
  tem, otherwise the character is non-zero. If the block was not present on the
  file system, no block was dumped; the block will be restored as a hole in the
  file. If there is not sufficient space in this record to describe all of the blocks in
  a file, TS_ADDR records will be scattered through the file, each one picking up
  where the last left off.

Each volume except the last ends with a tapemark (read as an end of file). The last volume
ends with a TS_END record and then the tapemark.

The structure idates describes an entry in the file /etc/dumpdates where dump history is kept.
The fields of the structure are:

- **id_name**: The dumped filesystem is ‘/dev/id_nam’.
- **id_incnr**: The level number of the dump tape; see dump(8).
- **id_ddate**: The date of the incremental dump in system format see types(5).

**FILES**

/etc/dumpdates

**SEE ALSO**
dump(8), restore(8), fs(5), types(5)
NAME
exports — NFS file systems being exported

SYNOPSIS
/etc/exports

DESCRIPTION
The file /etc/exports describes the file systems which are being exported to nfs(4) clients. It is created by the system administrator using a text editor and processed by the mount request daemon mountd(8c) each time a mount request is received.

The file consists of a list of file systems and the netgroups(5) or machine names allowed to remote mount each file system. The file system names are left justified and followed by a list of names separated by white space. The names will be looked up in /etc/netgroups and then in /etc/hosts. A file system name with no name list following means export to everyone. A "#" anywhere in the file indicates a comment extending to the end of the line it appears on. Lines beginning with white space are continuation lines.

EXAMPLE
/usr  clients  # export to my clients
/usr/local  # export to the world
/usr2  phoenix sun sundae  # export to only these machines

FILES
/etc/exports

SEE ALSO
mountd(8c)
NAME
fs, inode – format of file system volume

SYNOPSIS
#include <sys/types.h>
#include <sys/fs.h>
#include <sys/inode.h>

DESCRIPTION
Every file system storage volume (disk, nine-track tape, for instance) has a common format
for certain vital information. Every such volume is divided into a certain number of blocks.
The block size is a parameter of the file system. Sectors beginning at BBLOCK and continu­ing
for BBSIZE are used to contain primary and secondary bootstrapping programs.

The actual file system begins at sector SBLOCK with the super block that is of size SBSIZE.
The layout of the super block as defined by the include file <sys/fs.h> is:
#define FS_MAGIC 0x011954
struct fs {
    struct fs *fs_link; /* linked list of file systems */
    struct fs *fs_rlink; /* used for incore super blocks */
    daddr_t fs_sblkno; /* addr of super-block in filesys */
    daddr_t fs_cblkno; /* offset of cyl-block in filesys */
    daddr_t fs_iblkno; /* offset of inode-blocks in filesys */
    daddr_t fs_dblkno; /* offset of first data after cg */
    long fs_cgoffset; /* cylinder group offset in cylinder */
    long fs_cgmask; /* used to calc mod fs_ntrak */
    time_t fs_time; /* last time written */
    long fs_size; /* number of blocks in fs */
    long fs_dsize; /* number of data blocks in fs */
    long fs_ncg; /* number of cylinder groups */
    long fs_bsize; /* size of basic blocks in fs */
    long fs_fsize; /* size of frag blocks in fs */
    long fs_frag; /* number of frags in a block in fs */
    /* these are configuration parameters */
    long fs_minfree; /* minimum percentage of free blocks */
    long fs_rotdelay; /* num of ms for optimal next block */
    long fs_rps; /* disk revolutions per second */
    /* these fields can be computed from the others */
    long fs_bmask; /* “blkoff” calc of blk offsets */
    long fs_fmask; /* “fragoff” calc of frag offsets */
    long fs_bshift; /* “iblkno” calc of logical bknos */
    long fs_fshift; /* “numfrags” calc number of frags */
    /* these are configuration parameters */
    long fs_maxcontig; /* max number of contiguous blks */
    long fs_maxbpg; /* max number of blks per cyl group */
    /* these fields can be computed from the others */
    long fs_fragshift; /* block to frag shift */
    long fs_fsbtodb; /* fsbtodb and dbtofsb shift constant */
    long fs_sbsize; /* actual size of super block */
    long fs_csmask; /* csum block offset */
    long fs_csshift; /* csum block number */
    long fs_nindir; /* value of NINDIR */
    long fs_inopb; /* value of INOPB */
    long fs_nspf; /* value of NSPF */
    long fs_optim; /* optimization preference, see below */

4.2 Berkeley Distribution May 16, 1986
Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A file system is described by its super-block, which in turn describes the cylinder groups. The super-block is critical data and is replicated in each cylinder group to protect against catastrophic loss. This is done at file system creation time and the critical super-block data does not change, so the copies need not be referenced further unless disaster strikes.

Addresses stored in inodes are capable of addressing fragments of 'blocks'. File system blocks of at most size MAXBSIZE can be optionally broken into 2, 4, or 8 pieces, each of which is addressable; these pieces may be DEY_BSIZE, or some multiple of a DEY_BSIZE unit.

Large files consist of exclusively large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated as only as many fragments of a large block as are necessary. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determinable from information in the inode, using the "blksize(fs, ip, lbn)" macro.

The file system records space availability at the fragment level; to determine block availability, aligned fragments are examined.

The root inode is the root of the file system. Inode 0 can't be used for normal purposes and historically bad blocks were linked to inode 1, thus the root inode is 2 (inode 1 is no longer used for this purpose, however numerous dump tapes make this assumption, so we are stuck...
with it). The lost+found directory is given the next available inode when it is initially created by mksfs.

`fs_minfree` gives the minimum acceptable percentage of file system blocks that may be free. If the freelist drops below this level only the super-user may continue to allocate blocks. This may be set to 0 if no reserve of free blocks is deemed necessary, however severe performance degradations will be observed if the file system is run at greater than 90% full; thus the default value of `fs_minfree` is 10%.

Empirically the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 4, thus the default fragment size is a fourth of the block size.

`fs_optim` specifies whether the file system should try to minimize the time spent allocating blocks, or if it should attempt to minimize the space fragmentation on the disk. If the value of `fs_minfree` (see above) is less than 10%, then the file system defaults to optimizing for space to avoid running out of full sized blocks. If the value of minfree is greater than or equal to 10%, fragmentation is unlikely to be problematical, and the file system defaults to optimizing for time.

**Cylinder group related limits:** Each cylinder keeps track of the availability of blocks at different rotational positions, so that sequential blocks can be laid out with minimum rotational latency. NRPOS is the number of rotational positions which are distinguished. With NRPOS 8 the resolution of the summary information is 2ms for a typical 3600 rpm drive.

`fs_rotdelay` gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for `fs_rotdelay` is 2ms.

Each file system has a statically allocated number of inodes. An inode is allocated for each NBPI bytes of disk space. The inode allocation strategy is extremely conservative.

MAXIPG bounds the number of inodes per cylinder group, and is needed only to keep the structure simpler by having the only a single variable size element (the free bit map).

N.B.: MAXIPG must be a multiple of INOPB(fs).

MINBSIZE is the smallest allowable block size. With a MINBSIZE of 4096 it is possible to create files of size 2^32 with only two levels of indirection. MINBSIZE must be big enough to hold a cylinder group block, thus changes to (struct cg) must keep its size within MINBSIZE. MAXCPG is limited only to dimension an array in (struct cg); it can be made larger as long as that structure's size remains within the bounds dictated by MINBSIZE. Note that super blocks are never more than size SBSIZE.

The path name on which the file system is mounted is maintained in `fs_fsmnt`. MAXMNTLEN defines the amount of space allocated in the super block for this name. The limit on the amount of summary information per file system is defined by MAXCSBUFS. It is currently parameterized for a maximum of two million cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from `fs_csaddr` (size `fs_cssize`) in addition to the super block.

N.B.: `sizeof (struct csum)` must be a power of two in order for the "fs_cs" macro to work.

**Super block for a file system:** MAXBPC bounds the size of the rotational layout tables and is limited by the fact that the super block is of size SBSIZE. The size of these tables is inversely proportional to the block size of the file system. The size of the tables is increased when sector sizes are not powers of two, as this increases the number of cylinders included before the rotational pattern repeats (`fs_cpc`). The size of the rotational layout tables is derived from the number of bytes remaining in (struct fs).
MAXBPG bounds the number of blocks of data per cylinder group, and is limited by the fact that cylinder groups are at most one block. The size of the free block table is derived from the size of blocks and the number of remaining bytes in the cylinder group structure (struct cg).

**Inode**: The inode is the focus of all file activity in the UNIX file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is 'named' by its device/i-number pair. For further information, see the include file `<sys/inode.h>`.
NAME
fstab — static information about filesystems

SYNOPSIS
#include <mntent.h>

DESCRIPTION
The file /etc/fstab describes the filesystems and swapping partitions used by the local
machine. The system administrator can modify it with a text editor. It is read by com-
mands that mount, unmount, dump, restore, and check the consistency of filesystems; also
by the system when providing swap space. The file consists of a number of lines of the
form:

    fsname dir type opts freq passno

for example:

    /dev/xy0a /4.2 rw,noquota 1 2

The entries from this file are accessed using the routines in getmntent(3), which returns a
structure of the following form:

struct mntent {
    char *mnt_fsname; /* filesystem name */
    char *mnt_dir;  /* filesystem path prefix */
    char *mnt_type; /* 4.2, nfs, swap, or ignore */
    char *mnt_opts; /* rw, ro, noquota, quota, hard, soft */
    int mnt_freq;  /* dump frequency, in days */
    int mnt_passno; /* pass number on parallel fsck */
};

Fields are separated by white space; a '#' as the first non-white character indicates a com-
ment.

The mnt_dir field is the full path name of the directory to be mounted on.

The mnt_type field determines how the mnt_fsname and mnt_opts fields will be inter-
preted. Here is a list of the filesystem types currently supported, and the way each of them
interprets these fields:

4.2 mnt_fsname Must be a block special device.
nfs mnt_fsname the path on the server of the directory to be served.
swap mnt_fsname must be a block special device swap partition.

If the mnt_type is specified as ignore then the entry is ignored. This is useful to show disk
partitions not currently used.

The mnt_opts field contains a list of comma-separated option words. Some mnt_opts are
valid for all filesystem types, while others apply to a specific type only:

mnt_opts valid on all file systems (the default is rw,suid):

    rw read/write.
    ro read-only.
    suid set-uid execution allowed.
    nosuid set-uid execution not allowed.

mnt_opts specific to 4.2 file systems (the default is noquota).
quota usage limits enforced.
noquota usage limits not enforced.

mnt_opts specific to nfs (NFS) file systems (the defaults are:
fg,retry=1,timeo=7,retrans=4,port=NFS_PORT,hard
with defaults for rsize and wsize set by the kernel):
bg if the first attempt fails, retry in the background.
fg retry in foreground.
retry=n set number of failure retries to n.
rsize=n set read buffer size to n bytes.
wsize=n set write buffer size to n bytes.
timeo=n set NFS timeout to n tenths of a second.
retrans=n set number of NFS retransmissions to n.
port=n set server IP port number to n.
soft return error if server doesn't respond.
hard retry request until server responds.

The bg option causes mount to run in the background if the server's mountd(8) does not respond. mount attempts each request retry=n times before giving up. Once the filesystem is mounted, each nfs request made in the kernel waits timeo=n tenths of a second for a response. If no response arrives, the time-out is multiplied by 2 and the request is retransmitted. When retrans=n retransmissions have been sent with no reply a soft mounted filesystem returns an error on the request and a hard mounted filesystem retries the request. The number of bytes in a read or write request can be set with the rsize and wsize options.

The field mnt_freq indicates how often each partition should be dumped by the dump(8) command (and triggers that command's w option, which determines what filesystems should be dumped). Most systems set the mnt_freq field to 1, indicating that filesystems are dumped each day.

The final field, mnt_passno, is used by the consistency checking program fsck(8) to allow overlapped checking of filesystems during a reboot. All filesystems with mnt_passno of 1 are checked first simultaneously, then all filesystems with mnt_passno of 2, and so on. It is usual to make the mnt_passno of the root filesystem have the value 1, and then check one filesystem on each available disk drive in each subsequent pass, until all filesystem partitions are checked.

The /etc/fstab file is read only by programs and never written; the system administrator must maintain it manually. The order of records in /etc/fstab is important because fsck, mount, and umount process the file sequentially; filesystems must appear after filesystems they are mounted within.

FILES
/etc/fstab

SEE ALSO
getmntent(3), fsck(8), mount(8), quotacheck(8), quotaon(8)
NAME
gettytab – terminal configuration data base

SYNOPSIS
/etc/gettytab

DESCRIPTION
Gettytab is a simplified version of the termcap(5) data base used to describe terminal lines. The initial terminal login process getty(8) accesses the gettytab file each time it starts, allowing simpler reconfiguration of terminal characteristics. Each entry in the data base is used to describe one class of terminals.

There is a default terminal class, default, that is used to set global defaults for all other classes. (That is, the default entry is read, then the entry for the class required is used to override particular settings.)

CAPABILITIES
Refer to termcap(5) for a description of the file layout. The default column below lists defaults obtained if there is no entry in the table obtained, nor one in the special default table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ap</td>
<td>bool</td>
<td>false</td>
<td>terminal uses any parity</td>
</tr>
<tr>
<td>bd</td>
<td>num</td>
<td>0</td>
<td>backspace delay</td>
</tr>
<tr>
<td>bk</td>
<td>str</td>
<td>0377</td>
<td>alternate end of line character (input break)</td>
</tr>
<tr>
<td>cb</td>
<td>bool</td>
<td>false</td>
<td>use crt backspace mode</td>
</tr>
<tr>
<td>cd</td>
<td>num</td>
<td>0</td>
<td>carriage-return delay</td>
</tr>
<tr>
<td>ce</td>
<td>bool</td>
<td>false</td>
<td>use crt erase algorithm</td>
</tr>
<tr>
<td>ck</td>
<td>bool</td>
<td>false</td>
<td>use crt kill algorithm</td>
</tr>
<tr>
<td>cl</td>
<td>str</td>
<td>NULL</td>
<td>screen clear sequence</td>
</tr>
<tr>
<td>co</td>
<td>bool</td>
<td>false</td>
<td>console - add \n after login prompt</td>
</tr>
<tr>
<td>ds</td>
<td>str</td>
<td>“Y”</td>
<td>delayed suspend character</td>
</tr>
<tr>
<td>dx</td>
<td>bool</td>
<td>false</td>
<td>set DECCTEQ</td>
</tr>
<tr>
<td>ec</td>
<td>bool</td>
<td>false</td>
<td>leave echo OFF</td>
</tr>
<tr>
<td>ep</td>
<td>bool</td>
<td>false</td>
<td>terminal uses even parity</td>
</tr>
<tr>
<td>er</td>
<td>str</td>
<td>“?”</td>
<td>erase character</td>
</tr>
<tr>
<td>et</td>
<td>str</td>
<td>“D”</td>
<td>end of text (EOF) character</td>
</tr>
<tr>
<td>ev</td>
<td>str</td>
<td>NULL</td>
<td>initial environment</td>
</tr>
<tr>
<td>f0</td>
<td>num</td>
<td>unused</td>
<td>tty mode flags to write messages</td>
</tr>
<tr>
<td>f1</td>
<td>num</td>
<td>unused</td>
<td>tty mode flags to read login name</td>
</tr>
<tr>
<td>f2</td>
<td>num</td>
<td>unused</td>
<td>tty mode flags to leave terminal as</td>
</tr>
<tr>
<td>fd</td>
<td>num</td>
<td>0</td>
<td>form-feed (vertical motion) delay</td>
</tr>
<tr>
<td>fl</td>
<td>str</td>
<td>“O”</td>
<td>output flush character</td>
</tr>
<tr>
<td>hc</td>
<td>bool</td>
<td>false</td>
<td>do NOT hangup line on last close</td>
</tr>
<tr>
<td>he</td>
<td>str</td>
<td>NULL</td>
<td>hostname editing string</td>
</tr>
<tr>
<td>hn</td>
<td>str</td>
<td>hostname</td>
<td>hostname</td>
</tr>
<tr>
<td>ht</td>
<td>bool</td>
<td>false</td>
<td>terminal has real tabs</td>
</tr>
<tr>
<td>ig</td>
<td>bool</td>
<td>false</td>
<td>ignore garbage characters in login name</td>
</tr>
<tr>
<td>im</td>
<td>str</td>
<td>NULL</td>
<td>initial (banner) message</td>
</tr>
<tr>
<td>in</td>
<td>str</td>
<td>“C”</td>
<td>interrupt character</td>
</tr>
<tr>
<td>is</td>
<td>num</td>
<td>unused</td>
<td>input speed</td>
</tr>
<tr>
<td>kl</td>
<td>str</td>
<td>“U”</td>
<td>kill character</td>
</tr>
<tr>
<td>lc</td>
<td>bool</td>
<td>false</td>
<td>terminal has lower case</td>
</tr>
<tr>
<td>lm</td>
<td>str</td>
<td>login:</td>
<td>login prompt</td>
</tr>
<tr>
<td>ln</td>
<td>str</td>
<td>“V”</td>
<td>“literal next” character</td>
</tr>
<tr>
<td>lo</td>
<td>str</td>
<td>/bin/login</td>
<td>program to exec when name obtained</td>
</tr>
<tr>
<td>nd</td>
<td>num</td>
<td>0</td>
<td>newline (line-feed) delay</td>
</tr>
<tr>
<td>nl</td>
<td>bool</td>
<td>false</td>
<td>terminal has (or might have) a newline character</td>
</tr>
<tr>
<td>----</td>
<td>----------</td>
<td>-------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>nx</td>
<td>str</td>
<td>default</td>
<td>next table (for auto speed selection)</td>
</tr>
<tr>
<td>op</td>
<td>bool</td>
<td>false</td>
<td>terminal uses odd parity</td>
</tr>
<tr>
<td>os</td>
<td>num</td>
<td>unused</td>
<td>output speed</td>
</tr>
<tr>
<td>pc</td>
<td>str</td>
<td>\0</td>
<td>pad character</td>
</tr>
<tr>
<td>pe</td>
<td>bool</td>
<td>false</td>
<td>use printer (hard copy) erase algorithm</td>
</tr>
<tr>
<td>pf</td>
<td>num</td>
<td>0</td>
<td>delay between first prompt and following flush (seconds)</td>
</tr>
<tr>
<td>ps</td>
<td>bool</td>
<td>false</td>
<td>line connected to a MICOM port selector</td>
</tr>
<tr>
<td>qu</td>
<td>str</td>
<td>\</td>
<td>quit character</td>
</tr>
<tr>
<td>rp</td>
<td>str</td>
<td>&quot;R</td>
<td>line retype character</td>
</tr>
<tr>
<td>rw</td>
<td>bool</td>
<td>false</td>
<td>do NOT use raw for input, use cbreak</td>
</tr>
<tr>
<td>sp</td>
<td>num</td>
<td>unused</td>
<td>line speed (input and output)</td>
</tr>
<tr>
<td>su</td>
<td>str</td>
<td>&quot;Z</td>
<td>suspend character</td>
</tr>
<tr>
<td>tc</td>
<td>str</td>
<td>none</td>
<td>table continuation</td>
</tr>
<tr>
<td>to</td>
<td>num</td>
<td>0</td>
<td>timeout (seconds)</td>
</tr>
<tr>
<td>tt</td>
<td>str</td>
<td>NULL</td>
<td>terminal type (for environment)</td>
</tr>
<tr>
<td>ub</td>
<td>bool</td>
<td>false</td>
<td>do unbuffered output (of prompts etc)</td>
</tr>
<tr>
<td>uc</td>
<td>bool</td>
<td>false</td>
<td>terminal is known upper case only</td>
</tr>
<tr>
<td>we</td>
<td>str</td>
<td>&quot;W</td>
<td>word erase character</td>
</tr>
<tr>
<td>xc</td>
<td>bool</td>
<td>false</td>
<td>do NOT echo control chars as &quot;X</td>
</tr>
<tr>
<td>xf</td>
<td>str</td>
<td>&quot;S</td>
<td>XOFF (stop output) character</td>
</tr>
<tr>
<td>xn</td>
<td>str</td>
<td>&quot;Q</td>
<td>XON (start output) character</td>
</tr>
</tbody>
</table>

If no line speed is specified, speed will not be altered from that which prevails when getty is entered. Specifying an input or output speed will override line speed for stated direction only.

Terminal modes to be used for the output of the message, for input of the login name, and to leave the terminal set as upon completion, are derived from the boolean flags specified. If the derivation should prove inadequate, any (or all) of these three may be overridden with one of the f0, f1, or f2 numeric specifications, which can be used to specify (usually in octal, with a leading '0') the exact values of the flags. Local (new tty) flags are set in the top 16 bits of this (32 bit) value.

Should getty receive a null character (presumed to indicate a line break) it will restart using the table indicated by the nx entry. If there is none, it will re-use its original table.

Delays are specified in milliseconds, the nearest possible delay available in the tty driver will be used. Should greater certainty be desired, delays with values 0, 1, 2, and 3 are interpreted as choosing that particular delay algorithm from the driver.

The cl screen clear string may be preceded by a (decimal) number of milliseconds of delay required (a la termcap). This delay is simulated by repeated use of the pad character pc.

The initial message, and login message, im and lm may include the character sequence %h or %t to obtain the hostname or tty name respectively. (%%) obtains a single '%' character.) The hostname is normally obtained from the system, but may be set by the hn table entry. In either case it may be edited with he. The he string is a sequence of characters, each character that is neither '@' nor '#' is copied into the final hostname. A '@' in the he string, causes one character from the real hostname to be copied to the final hostname. A '#' in the he string, causes the next character of the real hostname to be skipped. Surplus '@' and '#' characters are ignored.

When getty execs the login process, given in the lo string (usually "/bin/login"), it will have set the enviroment to include the terminal type, as indicated by the tt string (if it exists). The ev string, can be used to enter additional data into the environment. It is a list of comma separated strings, each of which will presumably be of the form name=value.
If a non-zero timeout is specified, with to, then getty will exit within the indicated number of seconds, either having received a login name and passed control to login, or having received an alarm signal, and exited. This may be useful to hangup dial in lines.

Output from getty is even parity unless op is specified. Op may be specified with ap to allow any parity on input, but generate odd parity output. Note: this only applies while getty is being run, terminal driver limitations prevent a more complete implementation. Getty does not check parity of input characters in RAW mode.

SEE ALSO
login(1), termcap(5), getty(8).

BUGS
The special characters (erase, kill, etc.) are reset to system defaults by login(1). In all cases, '#' or 'H' typed in a login name will be treated as an erase character, and '@' will be treated as a kill character.

The delay stuff is a real crock. Apart form its general lack of flexibility, some of the delay algorithms are not implemented. The terminal driver should support sane delay settings.

The he capability is stupid.

Termcap format is horrid, something more rational should have been chosen.
NAME
group — group file

SYNOPSIS
/etc/group

DESCRIPTION
Group contains for each group the following information:
• group name
• encrypted password
• numerical group ID
• a comma separated list of all users allowed in the group
This is an ASCII file. The fields are separated by colons; each group is separated from the next by a new-line. If the password field is null, no password is demanded.
This file resides in the /etc directory. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.
A group file can have a line beginning with a plus (+), which means to incorporate entries from the yellow pages. There are two styles of + entries: All by itself, + means to insert the entire contents of the yellow pages group file at that point; +name means to insert the entry (if any) for name from the yellow pages at that point. If a + entry has a non-null password or group member field, the contents of that field will override what is contained in the yellow pages. The numerical group ID field cannot be overridden.

EXAMPLE
+myproject:::bill, steve
+: If these entries appear at the end of a group file, then the group myproject will have members bill and steve, and the password and group ID of the yellow pages entry for the group myproject. All the groups listed in the yellow pages will be pulled in and placed after the entry for myproject.

FILES
/etc/group /etc/yp/group

SEE ALSO
setgroups(2), initgroups(3), crypt(3), passwd(1), passwd(5)

BUGS
The passwd(1) command won't change group passwords.
NAME
hosts - host name data base

DESCRIPTION
The *hosts* file contains information regarding the known hosts on the network. For each host a single line should be present with the following information:

- official host name
- Internet address
- aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

When using the name server *named*(8), this file provides a backup when the name server is not running. For the name server, it is suggested that only a few addresses be included in this file. These include address for the local interfaces that *ifconfig*(8C) needs at boot time and a few machines on the local network.

This file may be created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts. As the database maintained at NIC is incomplete, use of the name server is recommend for sites on the DARPA Internet.

Network addresses are specified in the conventional "." notation using the *inet_addr()* routine from the Internet address manipulation library, *inet*(3N). Host names may contain any printable character other than a field delimiter, newline, or comment character.

FILES
/etc/hosts

SEE ALSO
- *gethostbyname*(3N), *ifconfig*(8C), *named*(8)
- Name Server Operations Guide for BIND
NAME
hosts.equiv — list of trusted hosts

DESCRIPTION
Hosts.equiv resides in directory /etc and contains a list of trusted hosts. When an rlogin(1) or rsh(1) request from such a host is made, and the initiator of the request is in /etc/passwd, then no further validity checking is done. That is, rlogin does not prompt for a password, and rsh completes successfully. So a remote user is "equivalenced" to a local user with the same user ID when the remote user is in hosts.equiv.

The format of hosts.equiv is a list of names, as in this example:

```
host1
host2
+@group1
-@group2
```

A line consisting of a simple host name means that anyone logging in from that host is trusted. A line consisting of +@group means that all hosts in that network group are trusted. A line consisting of -@group means that hosts in that group are not trusted. Programs scan hosts.equiv linearly, and stop at the first hit (either positive for hostname and +@ entries, or negative for -@ entries). A line consisting of a single + means that everyone is trusted.

The rhosts file has the same format as hosts.equiv. When user XXX executes rlogin or rsh, the rhosts file from XXX’s home directory is conceptually concatenated onto the end of hosts.equiv for permission checking. However, -@ entries are not sticky. If a user is excluded by a minus entry from hosts.equiv but included in rhosts, then that user is considered trusted. In the special case when the user is root, then only the /rhosts file is checked.

It is also possible to have two entries (separated by a single space) on a line of these files. In this case, if the remote host is equivalenced by the first entry, then the user named by the second entry is allowed to log in as anyone, that is, specify any name to the -1 flag (provided that name is in the /etc/passwd file, of course). Thus

```
sundown john
```

allows john to log in from sundown as anyone. The usual usage would be to put this entry in the rhosts file in the home directory for bill. Then john may log in as bill when coming from sundown. The second entry may be a netgroup, thus

```
+@group1 +@group2
```

allows any user in group2 coming from a host in group1 to log in as anyone.

FILES
/etc/hosts.equiv
/etc/yp/domain/netgroup
/etc/yp/domain/netgroup.byuser
/etc/yp/domain/netgroup.byhost

SEE ALSO
rlogin(1), rsh(1), netgroup(5)
NAME
map3270 - database for mapping ascii keystrokes into IBM 3270 keys

SYNOPSIS
/etc/map3270

DESCRIPTION
When emulating IBM-syle 3270 terminals under UNIX (see tn3270(1)), a mapping must be performed between sequences of keys hit on a user's (ascii) keyboard, and the keys that are available on a 3270. For example, a 3270 has a key labeled EEOF which erases the contents of the current field from the location of the cursor to the end. In order to accomplish this function, the terminal user and a program emulating a 3270 must agree on what keys will be typed to invoke the EEOF function.

The requirements for these sequences are:

1.) that the first character of the sequence be outside of the standard ascii printable characters;

2.) that no one sequence be an initial part of another (although sequences may share initial parts).

FORMAT
The file consists of entries for various terminals. The first part of an entry lists the names of the terminals which use that entry. These names should be the same as in /etc/termcap (see termcap(5)); note that often the terminals from various termcap entries will all use the same map3270 entry; for example, both 925 and 925vb (for 925 with visual bells) would probably use the same map3270 entry. After the names, separated by vertical bars ('|'), comes a left brace ('{'); the definitions; and, finally, a right brace ('}').

The definitions consist of a reserved keyword (see list below) which identifies the 3270 function (extended as defined below), followed by an equal sign ('='), followed by the various ways to generate this particular function, followed by a semi-colon (';'). Each way is a sequence of strings of printable ascii characters enclosed inside single quotes ('"'); various ways (options) are separated by vertical bars ('|').

Inside the single quotes, a few characters are special. A caret ('^') specifies that the next character is the "control" character of whatever the character is. So, "^A" represents control-a, i.e.: hexadecimal 1 (note that "^A" would generate the same code). To generate 'rubout', one enters "^?". To represent a control character inside a file requires using the caret to represent a control sequence; simply typing control-A will not work. Note: the ctrl-caret sequence (to generate a hexadecimal 1E) is represented as "^E" (not "^\".

In addition to the caret, a letter may be preceded by a backslash ("\"). Since this has little effect for most characters, its use is usually not recommended. For the case of a single quote (""), the backslash prevents that single quote from terminating the string. To have the backslash be part of the string, it is necessary to place two backslashes (\"\") in the file.

In addition, the following characters are special:

"\E" means an escape character;
"\n" means newline;
"\t" means tab;
"\r" means carriage return.

It is not necessary for each character in a string to be enclosed within single quotes. "\E\E\E" means three escape characters.
Comments, which may appear anywhere on a line, begin with a hash mark ("#"), and terminate at the end of that line. However, comments cannot begin inside a quoted string; a hash mark inside a quoted string has no special meaning.

3270 KEYS SUPPORTED

The following is the list of 3270 key names that are supported in this file. Note that some of the keys don't really exist on a 3270. In particular, the developers of this file have relied extensively on the work at the Yale University Computer Center with their 3270 emulator which runs in an IBM Series/1 front end. The following list corresponds closely to the functions that the developers of the Yale code offer in their product.

In the following list, the starred ("*") functions are not supported by tn3270(1). An unsupported function will cause tn3270(1) to send a bell sequence to the user's terminal.

3270 Key Name    Functional description

  [*]LPRT         local print
  DP             dup character
  FM             field mark character
  [*]CURSEL      cursor select
  RESHOW         redisplay the screen
  EINP           erase input
  EEOF           erase end of field
  DELETE         delete character
  INSRT          toggle insert mode
  TAB            field tab
  BTAB           field back tab
  COLTAB         column tab
  COLBACK        column back tab
  INDENT         indent one tab stop
  UNDENT         undent one tab stop
  NL             new line
  HOME           home the cursor
  UP             up cursor
  DOWN           down cursor
  RIGHT          right cursor
  LEFT           left cursor
  SETTAB         set a column tab
  DELTAB         delete a column tab
  SETMRG         set left margin
  SETHOM         set home position
  CLRSTAB        clear all column tabs
  [*]APLOFF      apl off
  [*]APLOFF       apl off
  [*]APLEND       treat input as ascii
  [*]PCON         xon/xoff on
  [*]PCOFF        xon/xoff off
  DISC           disconnect (suspend)
  [*]INIT         new terminal type
  [*]ALTK         alternate keyboard dvorak
  FLINP          flush input
  ERASE          erase last character
  WERASE         erase last word
  FERASE         erase field
SYNCH    we are in synch with the user
RESET    reset key-unlock keyboard
MASTER_RESET reset, unlock and redisplay
(*)XOFF  please hold output
(*)XON    please give me output
ESCAPE   enter telnet command mode
WORDTAB  tab to beginning of next word
WORDBACKTAB tab to beginning of current/last word
WORDEND  tab to end of current/next word
FIELDEND tab to last non-blank of current/next
           unprotected (writable) field.
PA1      program attention 1
PA2      program attention 2
PA3      program attention 3
CLEAR    local clear of the 3270 screen
TREQ     test request
ENTER    enter key
PFK1     program function key 1
PFK2     program function key 2
cetc.    etc.
PFK36    program function key 36

A SAMPLE ENTRY
The following entry is used by tn3270(1) when unable to locate a reasonable version in the
user's environment and in /etc/map3270:

    name { # actual name comes from TERM variable
clear = "z";
flinp = "x";
enter = "m";
delete = "d" | "?";  # note that "?" is delete (rubout)
synch = "r";
reshow = "v";
eeof = "e";
tab = "l";
btab = "b";
nl = "n";
left = "h";
right = "l";
up = "k";
down = "j";
einp = "w";
reset = "t";
xoff = "s";
xon = "q";
escape = "c";
ferase = "u";
insert = ";
    # program attention keys
pa1 = "p1"; pa2 = "p2"; pa3 = "p3";
    # program function keys
IBM 3270 KEY DEFINITIONS FOR AN ABOVE DEFINITION

The charts below show the proper keys to emulate each 3270 function when using the default key mapping supplied with `tn3270(1)` and `mset(1).`

<table>
<thead>
<tr>
<th>Command Keys</th>
<th>IBM 3270 Key</th>
<th>Default Key(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>RETURN</td>
<td></td>
</tr>
<tr>
<td>Clear</td>
<td>control-z</td>
<td></td>
</tr>
</tbody>
</table>

| Cursor Movement Keys | | |
|----------------------|----------------|
| New Line             | control-n or Home |
| Tab                  | control-i       |
| Back Tab             | control-b       |
| Cursor Left          | control-h       |
| 'Cursor Right        | control-l       |
| Cursor Up            | control-k       |
| Cursor Down          | control-j or LINE FEED |

| Edit Control Keys | | |
|-------------------|----------------|
| Delete Char       | control-d or RUB |
| Erase EOF         | control-e       |
| Erase Input       | control-w       |
| Insert Mode       | ESC Space       |
| End Insert        | ESC Space       |

| Program Function Keys | | |
|-----------------------|----------------|
| PF1                   | ESC 1          |
| PF2                   | ESC 2          |
| ...                   | ...            |
| PF10                  | ESC 0          |
| PF11                  | ESC -          |
| PF12                  | ESC =          |
| PF13                  | ESC !          |
| PF14                  | ESC @          |
| ...                   | ...            |
| PF24                  | ESC +          |

| Program Attention Keys | | |
|------------------------|----------------|
| PA1                    | control-p 1    |
| PA2                    | control-p 2    |
| PA3                    | control-p 3    |

| Local Control Keys | | |
|-------------------|----------------|
| Reset After Error | control-r      |
| Purge Input Buffer| control-x      |
| Keyboard Unlock    | control-t      |
| Redisplay Screen   | control-v      |

| Other Keys | | |
|------------|----------------|
| Erase current field | control-u    |
FILES
   /etc/map3270

SEE ALSO
   tn3270(1), mset(1), Yale ASCII Terminal Communication System II Program Description/Operator's Manual (IBM SB30-1911)

AUTHOR
   Greg Minshall

BUGS
   Tn3270 doesn't yet understand how to process all the functions available in map3270; when such a function is requested tn3270 will beep at you.
   The definition of "word" (for "word delete", "word tab") should be a run-time option. Currently it is defined as the kernel tty driver defines it (strings of non-blanks); more than one person would rather use the "vi" definition (strings of specials, strings of alphanumeric).
NAME
/etc/mtab — mounted file system table

SYNOPSIS
#include <mntent.h>

DESCRIPTION
Mtab resides in the /etc directory, and contains a table of filesystems currently mounted by the mount command. Umount removes entries from this file.

The file contains a line of information for each mounted filesystem, structurally identical to the contents of /etc/fstab, described in fstab(5). There are a number of lines of the form:

```
fsname dir type opts freq pasmo
```

for example:

```
/dev/xy0a / 4.2 rw,noquota 1 2
```

The file is accessed by programs using getmntent(3), and by the system administrator using a text editor.

FILES
/etc/mtab

SEE ALSO
getmntent(3), fstab(5), mount(8)
NAME
netgroup — list of network groups

DESCRIPTION
Netgroup defines network wide groups, used for permission checking when doing remote
mounts, remote logins, and remote shells. For remote mounts, the information in netgroup
is used to classify machines; for remote logins and remote shells, it is used to classify users.
Each line of the netgroup file defines a group and has the format

groupname member1 member2 ....

where memberi is either another group name, or a triple:

(hostname, username, domainname)

Any of three fields can be empty, in which case it signifies a wild card. Thus
universal (...) defines a group to which everyone belongs. Field names that begin with something other
than a letter, digit or underscore (such as "-_") work in precisely the opposite fashion. For
example, consider the following entries:

justmachines (analytica,-.sun)
justpeople (-.babbage.sun)

The machine analytica belongs to the group justmachines in the domain sun, but no users
belong to it. Similarly, the user babbage belongs to the group justpeople in the domain sun,
but no machines belong to it.

Network groups are contained in the yellow pages, and are accessed through these files:

/etc/yp/domainname/netgroup.dir
/etc/yp/domainname/netgroup.pag
/etc/yp/domainname/netgroup.byuser.dir
/etc/yp/domainname/netgroup.byuser.pag
/etc/yp/domainname/netgroup.byhost.dir
/etc/yp/domainname/netgroup.byhost.pag

These files can be created from /etc/netgroup using makedbm(8).

FILES

/etc/netgroup
/etc/yp/domainname/netgroup.dir
/etc/yp/domainname/netgroup.pag
/etc/yp/domainname/netgroup.byuser.dir
/etc/yp/domainname/netgroup.byuser.pag
/etc/yp/domainname/netgroup.byhost.dir
/etc/yp/domainname/netgroup.byhost.pag

SEE ALSO
getnetgrent(3), exportfs(8), makedbm(8), ypserv(8)
NAME
  networks - network name data base

DESCRIPTION
  The networks file contains information regarding the known networks which comprise the
  DARPA Internet. For each network a single line should be present with the following infor-
  mation:

  official network name
  network number
  aliases

  Items are separated by any number of blanks and/or tab characters. A "#" indicates the
  beginning of a comment; characters up to the end of the line are not interpreted by routines
  which search the file. This file is normally created from the official network data base main-
  tained at the Network Information Control Center (NIC), though local changes may be
  required to bring it up to date regarding unofficial aliases and/or unknown networks.

  Network number may be specified in the conventional "." notation using the inet_network()
  routine from the Internet address manipulation library, inet(3N). Network names may con-
  tain any printable character other than a field delimiter, newline, or comment character.

FILES
  /etc/networks

SEE ALSO
  getnetent(3N)

BUGS
  A name server should be used instead of a static file.
NAME
passwd — password file

SYNOPSIS
/etc/passwd

DESCRIPTION
The passwd file contains for each user the following information:

name  User's login name — contains no upper case characters and must not be greater than eight characters long.

password  encrypted password

numerical user ID
  This is the user's ID in the system and it must be unique.

numerical group ID
  This is the number of the group that the user belongs to.

user's real name
  In some versions of UNIX, this field also contains the user's office, extension, home phone, and so on. For historical reasons this field is called the GCOS field.

initial working directory
  The directory that the user is positioned in when they log in — this is known as the 'home' directory.

shell  program to use as Shell when the user logs in.

The user's real name field may contain '&', meaning insert the login name.

The password file is an ASCII file. Each field within each user's entry is separated from the next by a colon. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, /bin/sh is used.

The passwd file can also have line beginning with a plus (+), which means to incorporate entries from the yellow pages. There are three styles of + entries: all by itself, + means to insert the entire contents of the yellow pages password file at that point; +name means to insert the entry (if any) for name from the yellow pages at that point; +@name means to insert the entries for all members of the network group name at that point. If a + entry has a non-null password, directory, gecos, or shell field, they will override what is contained in the yellow pages. The numerical user ID and group ID fields cannot be overridden.

EXAMPLE
Here is a sample /etc/passwd file:

root:q.mJzTnu8icF::0:10:God:/bin/csh
tut:6k/7KCFRPNVXg:508:10:Bill Tuthill:/usr2/tut:/bin/csh
+john:
+@documentation:no-login:
+:::Guest

In this example, there are specific entries for users root tut, in case the yellow pages are out of order. The user will have his password entry in the yellow pages incorporated without change; anyone in the netgroup documentation will have their password field disabled, and anyone else will be able to log in with their usual password, shell, and home directory, but with a gecos field of Guest.

The password file resides in the /etc directory. Because of the encrypted passwords, it has general read permission and can be used, for example, to map numerical user ID's to names.
Appropriate precautions must be taken to lock the `/etc/passwd` file against simultaneous changes if it is to be edited with a text editor; `vipw(8)` does the necessary locking.

FILES

`/etc/passwd`

SEE ALSO

`getpwent(3), login(1), crypt(3), passwd(1), group(5), vipw(8), adduser(8)`
NAME
phones – remote host phone number data base

DESCRIPTION
The file /etc/phones contains the system-wide private phone numbers for the tip(1C) program. This file is normally unreadable, and so may contain privileged information. The format of the file is a series of lines of the form: <system-name>[ \t]*<phone-number>. The system name is one of those defined in the remote(5) file and the phone number is constructed from any sequence of characters terminated only by "," or the end of the line. The "=" and "*" characters are indicators to the auto call units to pause and wait for a second dial tone (when going through an exchange). The "=" is required by the DF02-AC and the "*" is required by the BIZCOMP 1030.

Only one phone number per line is permitted. However, if more than one line in the file contains the same system name tip(1C) will attempt to dial each one in turn, until it establishes a connection.

FILES
/etc/phones

SEE ALSO
tip(1C), remote(5)
NAME
plot – graphics interface

DESCRIPTION
Files of this format are produced by routines described in plot(3X) and plot(3F), and are interpreted for various devices by commands described in plot(1G). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the x and y values; each value is a signed integer. The last designated point in an l, m, n, a, or p instruction becomes the 'current point' for the next instruction. The a and c instructions change the current point in a manner dependent upon the specific device.

Each of the following descriptions begins with the name of the corresponding routine in plot(3X).

m move: The next four bytes give a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes.

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a newline.

a arc: The first four bytes give the center, the next four give the starting point, and the last four give the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

c circle: The first four bytes give the center of the circle, the next two the radius.

e erase: Start another frame of output.

f linemod: Take the following string, up to a newline, as the style for drawing further lines. The styles are 'dotted,' 'solid,' 'longdashed,' 'shortdashed,' and 'dotdashed.' Effective only in plot 4014 and plot ver.

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of plot(1G). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face isn't square.

4013 space(0, 0, 780, 780);
4014 space(0, 0, 3120, 3120);
ver space(0, 0, 2048, 2048);
300, 300s space(0, 0, 4096, 4096);
450 space(0, 0, 4096, 4096);

SEE ALSO
plot(1G), plot(3X), plot(3F), graph(1G)
NAME
printcap – printer capability data base

SYNOPSIS
/etc/printcap

DESCRIPTION
Printcap is a simplified version of the termcap(5) data base 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 data base is used to describe one printer. This data base may not be substituted for, as is possible for termcap, because it may allow accounting to be bypassed.

The default printer is normally /p, though the environment variable PRINTER may be used to override this. Each spooling utility supports an option, -Pprinter, to allow explicit naming of a destination printer.

Refer to the 4.3BSD Line Printer Spooler Manual for a complete discussion on how setup the database for a given printer.

CAPABILITIES
Refer to termcap(5) 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>hl</td>
<td>bool</td>
<td>false</td>
<td>print the burst header page last</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>”/dev/console”</td>
<td>error logging file name</td>
</tr>
<tr>
<td>lo</td>
<td>str</td>
<td>“lock”</td>
<td>name of lock file</td>
</tr>
<tr>
<td>lp</td>
<td>str</td>
<td>”/dev/lp”</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>pc</td>
<td>num</td>
<td>200</td>
<td>price per foot or page in hundredths of cents</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>rg</td>
<td>str</td>
<td>NULL</td>
<td>restricted group. Only members of group allowed access</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>“lp”</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>
</tbody>
</table>

4.2 Berkeley Distribution May 14, 1986
sd  str  "/usr/spool/lpd"  spool directory
sf  bool false  suppress form feeds
sh  bool false  suppress printing of burst page header
st  str  "status"  status file name
tf  str  NULL  troff data filter (cat phototypesetter)
tr  str  NULL  trailer string to print when queue empties
vf  str  NULL  raster image filter
xc  num 0  if lp is a tty, clear local mode bits (tty (4))
xs  num 0  like 'xc' but set bits

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

FILTERS
The lpd(8) daemon creates a pipeline of filters to process files for various printer types. The filters selected depend on the flags passed to lpr(1). The pipeline set up is:

- p  pr  if regular text + pr(1)
none  if regular text
- c  cf  cifplot
- d  df  DVI (tex)
- g  gf  plot(3)
- n  nf  ditroff
- f  rf  Fortran
- t  tf  troff
- v  vf  raster image

The if filter is invoked with arguments:

if [-c ] -w width -l length -i indent -n login -h host acct-file

The -c flag is passed only if the -l flag (pass control characters literally) is specified to lpr. Width and length specify the page width and length (from pw and pl respectively) in characters. The -n and -h parameters specify the login name and host name of the owner of the job respectively. Acct-file is passed from the af printcap entry.

If no if is specified, of is used instead, with the distinction that of is opened only once, while if is opened for every individual job. Thus, if is better suited to performing accounting. The of is only given the width and length flags.

All other filters are called as:

filter -x width -y length -n login -h host acct-file

where width and length are represented in pixels, specified by the px and py entries respectively.

All filters take stdin as the file, stdout as the printer, may log either to stderr or using syslog(3), and must not ignore SIGINT.

LOGGING
Error messages generated by the line printer programs themselves (that is, the lpx programs) are logged by syslog(3) using the LPR facility. Messages printed on stderr of one of the filters are sent to the corresponding if file. The filters may, of course, use syslog themselves.

Error messages sent to the console have a carriage return and a line feed appended to them, rather than just a line feed.

SEE ALSO
termcap(5), lpc(8), lpd(8), pac(8), lpr(1), lpq(1), lprm(1)
4.3BSD Line Printer Spooler Manual

4.2 Berkeley Distribution  May 14, 1986  2
NAME
  protocols – protocol name data base

DESCRIPTION
  The protocols file contains information regarding the known protocols used in the DARPA
  Internet. For each protocol a single line should be present with the following information:
    official protocol name
    protocol number
    aliases
  Items are separated by any number of blanks and/or tab characters. A “#” indicates the
  beginning of a comment; characters up to the end of the line are not interpreted by routines
  which search the file.
  Protocol names may contain any printable character other than a field delimiter, newline, or
  comment character.

FILES
  /etc/protocols

SEE ALSO
  getprotoent(3N)

BUGS
  A name server should be used instead of a static file.
NAME
    remote – remote host description file

DESCRIPTION
    The systems known by tip(1C) and their attributes are stored in an ASCII file which is struc­
tured somewhat like the termcap(5) file. Each line in the file provides a description for a sin­
gle system. Fields are separated by a colon ("":). Lines ending in a \ character with an imme­diately following newline are continued on the next line.

    The first entry is the name(s) of the host system. If there is more than one name for a system,
the names are separated by vertical bars. After the name of the system comes the fields of the
description. A field name followed by an '="' sign indicates a string value follows. A field
name followed by a '##' sign indicates a following numeric value.

    Entries named "tip*" and "cu*" are used as default entries by tip, and the cu interface to tip,
as follows. When tip is invoked with only a phone number, it looks for an entry of the form
"tip300", where 300 is the baud rate with which the connection is to be made. When the cu
interface is used, entries of the form "cu300" are used.

CAPABILITIES
    Capabilities are either strings (str), numbers (num), or boolean flags (bool). A string capa­
bility is specified by capability=value; e.g. "dv=/dev/harris". A numeric capability is specified
by capability*value; e.g. "xa*99". A boolean capability is specified by simply listing the capa­
bility.

    at (str) Auto call unit type.
    br (str) The baud rate used in establishing a connection to the remote host. This is a
decimal number. The default baud rate is 300 baud.
    cm (str) An initial connection message to be sent to the remote host. For example, if a
host is reached through port selector, this might be set to the appropriate sequence
required to switch to the host.
    cu (str) Call unit if making a phone call. Default is the same as the ‘dv’ field.
    di (str) Disconnect message sent to the host when a disconnect is requested by the user.
    du (bool) This host is on a dial-up line.
    dv (str) UNIX device(s) to open to establish a connection. If this file refers to a terminal
line, tip(1C) attempts to perform an exclusive open on the device to insure only one
user at a time has access to the port.
    el (str) Characters marking an end-of-line. The default is NULL. "" escapes are only
recognized by tip after one of the characters in ‘el’, or after a carriage-return.
    fs (str) Frame size for transfers. The default frame size is equal to BUFSIZ.
    hd (bool) The host uses half-duplex communication, local echo should be performed.
    ie (str) Input end-of-file marks. The default is NULL.
    oe (str) Output end-of-file string. The default is NULL. When tip is transferring a file,
this string is sent at end-of-file.
    pa (str) The type of parity to use when sending data to the host. This may be one of
"even", "odd", "none", "zero" (always set bit 8 to zero), "one" (always set bit 8 to
1). The default is even parity.
    pn (str) Telephone number(s) for this host. If the telephone number field contains an @
sign, tip searches the file /etc/phones file for a list of telephone numbers; c.f.
phones(5).
tc (str) Indicates that the list of capabilities is continued in the named description. This is used primarily to share common capability information.

Here is a short example showing the use of the capability continuation feature:

UNIX-1200:
:dv=/dev/cau0:el="D"U"C"S"O"O@:du:at=ventel:ie=#$%:oe="D;br#1200:
arpavax|ax:
:pn=76543211:tc=UNIX-1200

FILES
/etc/remote

SEE ALSO
tip(1C), phones(5)
NAME
resolver configuration file

SYNOPSIS
/etc/resolv.conf

DESCRIPTION
The resolver configuration file contains information that is read by the resolver routines the
first time they are invoked by a process. The file is designed to be human readable and con­tains a list of name-value pairs that provide various types of resolver information.

On a normally configured system this file should not be necessary. The only name server to
be queried will be on the local machine and the domain name is retrieved from the system.

The different configuration options are:

nameserver
followed by the Internet address (in dot notation) of a name server that the resolver
should query. At least one name server should be listed. Up to MAXNS (currently 3)
name servers may be listed, in that case the resolver library queries tries them in the
order listed. If no nameserver entries are present, the default is to use the name
server on the local machine. (The algorithm used is to try a name server, and if the
query times out, try the next, until out of name servers, then repeat trying all the
name servers until a maximum number of retries are made).

domain followed by a domain name, that is the default domain to append to names that do
not have a dot in them. If no domain entries are present, the domain returned by
gethostname(2) is used (everything after the first "."). Finally, if the host name does
not contain a domain part, the root domain is assumed.

The name value pair must appear on a single line, and the keyword (e.g. nameserver) must
start the line. The value follows the keyword, separated by white space.

FILES
/etc/resolv.conf

SEE ALSO
gethostbyname(3N), resolver(3), named(8)
Name Server Operations Guide for BIND
NAME
rmtab — remotely mounted file system table

DESCRIPTION
Rmtab resides in directory /etc and contains a record of all clients that have done remote mounts of file systems from this machine. Whenever a remote mount is done, an entry is made in the rmtab file of the machine serving up that file system. Umount removes entries, if of a remotely mounted file system. Umount —a broadcasts to all servers, and informs them that they should remove all entries from rmtab created by the sender of the broadcast message. By placing a umount —a command in /etc/rc.boot, rmtab tables can be purged of entries made by a crashed host, which upon rebooting did not remount the same file systems it had before. The table is a series of lines of the form:

hostname:directory

This table is used only to preserve information between crashes, and is read only by mountd(8) when it starts up. Mountd keeps an in-core table, which it uses to handle requests from programs like showmount(1) and shutdown(8).

FILES
/etc/rmtab

SEE ALSO
showmount(1), mountd(8), mount(8), umount(8), shutdown(8)

BUGS
Although the rmtab table is close to the truth, it is not always 100% accurate.
NAME
services - service name data base

DESCRIPTION
The services file contains information regarding the known services available in the DARPA Internet. For each service a single line should be present with the following information:

- official service name
- port number
- protocol name
- aliases

Items are separated by any number of blanks and/or tab characters. The port number and protocol name are considered a single item; a "/" is used to separate the port and protocol (e.g. "512/tcp"). A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Service names may contain any printable character other than a field delimiter, newline, or comment character.

FILES
/etc/services

SEE ALSO
getservent(3N)

BUGS
A name server should be used instead of a static file.
This page intentionally left almost blank.
NAME

stb - symbol table types

SYNOPSIS

#include <stab.h>

DESCRIPTION

*Stab.h* defines some values of the n_type field of the symbol table of a.out files. These are the types for permanent symbols (i.e. not local labels, etc.) used by the old debugger *sdb* and the Berkeley Pascal compiler *pc*(1). Symbol table entries can be produced by the *stabs* assembler directive. This allows one to specify a double-quote delimited name, a symbol type, one char and one short of information about the symbol, and an unsigned long (usually an address). To avoid having to produce an explicit label for the address field, the *stabd* directive can be used to implicitly address the current location. If no name is needed, symbol table entries can be generated using the *stabn* directive. The loader promises to preserve the order of symbol table entries produced by *stabs* directives. As described in *a.out*(5), an element of the symbol table consists of the following structure:

```c
/*
 * Format of a symbol table entry.
 */
struct nlist {
    union {
        char .n_name; /* for use when in-core */
        long n_strx;  /* index into file string table */
    } n_un;
    unsigned char n_type; /* type flag */
    char n_other;  /* unused */
    short n_desc;  /* see struct desc, below */
    unsigned n_value; /* address or offset or line */
};
```

The low bits of the n_type field are used to place a symbol into at most one segment, according to the following masks, defined in <a.out.h>. A symbol can be in none of these segments by having none of these segment bits set.

```c
/*
 * Simple values for n_type.
 */
#define N_UNDF 0x0  /* undefined */
#define N_ABS 0x2  /* absolute */
#define N_TEXT 0x4  /* text */
#define N_DATA 0x6  /* data */
#define N_BSS 0x8  /* bss */
#define N_EXT 01  /* external bit, or'ed in */
```

The n_value field of a symbol is relocated by the linker, *ld*(1) as an address within the appropriate segment. N_value fields of symbols not in any segment are unchanged by the linker. In addition, the linker will discard certain symbols, according to rules of its own, unless the n_type field has one of the following bits set:

```c
/*
 * Other permanent symbol table entries have some of the N_STAB bits set.
 * These are given in <stab.h>
 */
#define N_STAB 0xe0 /* if any of these bits set, don't discard */
```
This allows up to 112 (7 * 16) symbol types, split between the various segments. Some of these have already been claimed. The old symbolic debugger, sdb, uses the following n_type values:

```c
#define N_GSYM 0x20 /* global symbol: name,,0,type,0 */
#define N_FNAME 0x22 /* procedure name (F77 kludge): name,,0 */
#define N_FUN 0x24 /* procedure: name,,0,linenumber,address */
#define N_STSYM 0x26 /* static symbol: name,,0,type,address */
#define N_LCSYM 0x28 /* .lcomm symbol: name,,0,type,address */
#define N_RSYM 0x20 /* register sym: name,,0,type,register */
#define N_SLINE 0x44 /* src line: O,,0,linenumber,address */
#define N_SSYM 0x60 /* structure elt: name,,0,type,struct_offset */
#define N_SO 0x64 /* source file name: name,,0,0,address */
#define N_LSYM 0x80 /* local sym: name,,0,type,offset */
#define N_SOL 0x84 /* #included file name: name,,0,0,address */
#define N_PSYM 0xa0 /* parameter: name,,0,type,offset */
#define N_ENTRY 0xa4 /* alternate entry: name,linenumber,address */
#define N_LBRAC 0xc0 /* left bracket: O,,0,nesting level,address */
#define N_RBRAC 0xe0 /* right bracket: O,,0,nesting level,address */
#define N_BCOMM 0xe2 /* begin common: name,, */
#define N_ECOMM0xe4 /* end common: name,, */
#define N_ECOML 0xe8 /* end common (local name): ,,address */
#define N_LENG 0xfe /* second stab entry with length information */
```

where the comments give sdb conventional use for .stabs and the n_name, n_other, n_desc, and n_value fields of the given n_type. Sdb uses the n_desc field to hold a type specifier in the form used by the Portable C Compiler, cc(1); see the header file pcc.h for details on the format of these type values.

The Berkeley Pascal compiler, pc(1), uses the following n_type value:

```c
#define N_PC 0x30 /* global pascal symbol: name,,0,subtype,line */
```

and uses the following subtypes to do type checking across separately compiled files:

1. source file name
2. included file name
3. global label
4. global constant
5. global type
6. global variable
7. global function
8. global procedure
9. external function
10. external procedure
11. library variable
12. library routine

SEE ALSO

as(1), ld(1), dbx(1), a.out(5)

BUGS

More basic types are needed.
NAME
tar – tape archive file format

DESCRIPTION
Tar, (the tape archive command) dumps several files into one, in a medium suitable for trans-
portation.
A “tar tape” or file is a series of blocks. Each block is of size TBLOCK. A file on the tape is
represented by a header block which describes the file, followed by zero or more blocks which
give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as
an end-of-file indicator.
The blocks are grouped for physical I/O operations. Each group of n blocks (where n is set by
the b keyletter on the tar(1) command line — default is 20 blocks) is written with a single sys-
tem call; on nine-track tapes, the result of this write is a single tape record. The last group is
always written at the full size, so blocks after the two zero blocks contain random data. On
reading, the specified or default group size is used for the first read, but if that read returns
less than a full tape block, the reduced block size is used for further reads.
The header block looks like:

```c
#define TBLOCK 512
#define NAMSIZ 100

union hblock {
    char dummy[TBLOCK];
    struct header {
        char name[NAMSIZ];
        char mode[8];
        char uid[8];
        char gid[8];
        char size[12];
        char mtime[12];
        char chksum[8];
        char linkflag;
        char linkname[NAMSIZ];
    } dbuf;
};
```

Name is a null-terminated string. The other fields are zero-filled octal numbers in ASCII.
Each field (of width w) contains w-2 digits, a space, and a null, except size and mtime, which
do not contain the trailing null and chksum which has a null followed by a space. Name is
the name of the file, as specified on the tar command line. Files dumped because they were
in a directory which was named in the command line have the directory name as prefix and
/filename as suffix. Mode is the file mode, with the top bit masked off.Uid and gid are the
user and group numbers which own the file. Size is the size of the file in bytes. Links and
symbolic links are dumped with this field specified as zero. Mtime is the modification time of
the file at the time it was dumped. Chksum is an octal ASCII value which represents the sum
of all the bytes in the header block. When calculating the checksum, the chksum field is
treated as if it were all blanks. Linkflag is NULL if the file is “normal” or a special file,
ASCII ‘1’ if it is an hard link, and ASCII ‘2’ if it is a symbolic link. The name linked-to, if
any, is in linkname, with a trailing null. Unused fields of the header are binary zeros (and are
included in the checksum).
The first time a given i-node number is dumped, it is dumped as a regular file. The second
and subsequent times, it is dumped as a link instead. Upon retrieval, if a link entry is
retrieved, but not the file it was linked to, an error message is printed and the tape must be
manually re-scanned to retrieve the linked-to file.
The encoding of the header is designed to be portable across machines.

SEE ALSO
tar(1)

BUGS
Names or linknames longer than NAMSIZ produce error reports and cannot be dumped.
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, 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>ae</td>
<td>str</td>
<td>(P)</td>
<td>End alternate character set</td>
</tr>
<tr>
<td>AL</td>
<td>str</td>
<td>(NP*)</td>
<td>Add n new blank lines</td>
</tr>
<tr>
<td>al</td>
<td>str</td>
<td>(P*)</td>
<td>Add new blank line</td>
</tr>
<tr>
<td>am</td>
<td>bool</td>
<td></td>
<td>Terminal has automatic margins</td>
</tr>
<tr>
<td>as</td>
<td>str</td>
<td>(P)</td>
<td>Start alternate character set</td>
</tr>
<tr>
<td>bc</td>
<td>str</td>
<td>(o)</td>
<td>Backspace if not &quot;H</td>
</tr>
<tr>
<td>bl</td>
<td>str</td>
<td>(P)</td>
<td>Audible signal (bell)</td>
</tr>
<tr>
<td>bs</td>
<td>bool</td>
<td>(o)</td>
<td>Terminal can backspace with &quot;H</td>
</tr>
</tbody>
</table>
Back tab
Backspace wraps from column 0 to last column
Terminal settable command character in prototype
Clear to end of display
Clear to end of line
Set cursor column (horizontal position)
Clear screen and home cursor
Memory-relative cursor addressing
Screen-relative cursor motion
Number of columns in a line (See BUGS section below)
Carriage return
Change scrolling region (VT100)
Clear all tab stops
Set cursor row (vertical position)
Display may be retained above the screen
Milliseconds of bs delay needed (default 0)
Display may be retained below the screen
Delete n characters
Milliseconds of cr delay needed (default 0)
Delete character
Milliseconds of ff delay needed (default 0)
Delete n lines
Delete line
Enter delete mode
Milliseconds of nl delay needed (default 0)
Move cursor down n lines
Down one line
Disable status line
Milliseconds of horizontal tab delay needed (default 0)
Milliseconds of vertical tab delay needed (default 0)
Erase n characters
End delete mode
End insert mode
Can erase overstrikes with a blank
Even parity
Escape can be used on the status line
Hardcopy terminal page eject
Return from status line
Generic line type (e.g. dialup, switch)
Hardcopy terminal
Half-duplex
Half-line down (forward 1/2 linefeed)
Home cursor
Has extra "status line"
Half-line up (reverse 1/2 linefeed)
Cannot print "s (Hazeltine)
Terminal initialization strings (terminfo only)
Insert n blank characters
Insert character
Name of file containing initialization string
Enter insert mode
Insert mode distinguishes nulls
Pathname of program for initialization (terminfo only)
**ip**  str  *(P*)  Insert pad after character inserted  
**is**  str  Terminal initialization string *(termcap only)*  
**it**  num  Tabs initially every *n* positions  
**K1**  str  Sent by keypad upper left  
**K2**  str  Sent by keypad upper right  
**K3**  str  Sent by keypad center  
**K4**  str  Sent by keypad lower left  
**K5**  str  Sent by keypad lower right  
**k0-k9**  str  Sent by function keys 0-9  
**kA**  str  Sent by insert-line key  
**ka**  str  Sent by clear-all-tabs key  
**kb**  str  Sent by backspace key  
**KC**  str  Sent by clear-screen or erase key  
**KD**  str  Sent by delete-character key  
**kd**  str  Sent by down-arrow key  
**kE**  str  Sent by clear-to-end-of-line key  
**ke**  str  Out of “keypad transmit” mode  
**kF**  str  Sent by scroll-forward/down key  
**KH**  str  Sent by home-down key  
**kh**  str  Sent by home key  
**KI**  str  Sent by insert-character or enter-insert-mode key  
**kL**  str  Sent by delete-line key  
**kl**  str  Sent by left-arrow key  
**KM**  str  Sent by insert key while in insert mode  
**km**  bool  Has a “meta” key (shift, sets parity bit)  
**kN**  str  Sent by next-page key  
**kn**  num  *(o)*  Number of function key (k0–k9) keys (default 0)  
**ko**  str  *(o)*  Termcap entries for other non-function keys  
**KP**  str  Sent by previous-page key  
**KR**  str  Sent by scroll-backward/up key  
**kr**  str  Sent by right-arrow key  
**kS**  str  Sent by clear-to-end-of-screen key  
**ks**  str  Sent terminal in “keypad transmit” mode  
**kT**  str  Sent by set-tab key  
**kt**  str  Sent by clear-tab key  
**ku**  str  Sent by up-arrow key  
**l0-l9**  str  Labels on function keys if not “fn”  
**LC**  bool  *(o)*  Lower-case only  
**LE**  str  *(NP)*  Move cursor left *n* positions  
**le**  str  *(P)*  Move cursor left one position  
**li**  num  Number of lines on screen or page *(See BUGS section below)*  
**ll**  str  Last line, first column  
**lm**  num  Lines of memory if > ll *(0 means varies)*  
**ma**  str  *(o)*  Arrow key map *(used by vi version 2 only)*  
**mb**  str  Turn on blinking attribute  
**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  *(o)*  Memory lock on above cursor  
**mm**  str  Turn on “meta mode” *(8th bit)*  
**mo**  str  Turn off “meta mode”
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mp</td>
<td>str</td>
<td>Turn on protected attribute</td>
</tr>
<tr>
<td>mr</td>
<td>str</td>
<td>Turn on reverse-video attribute</td>
</tr>
<tr>
<td>ms</td>
<td>bool</td>
<td>Safe to move in standout modes</td>
</tr>
<tr>
<td>mu</td>
<td>str</td>
<td>Memory unlock (turn off memory lock)</td>
</tr>
<tr>
<td>nc</td>
<td>bool</td>
<td>No correctly-working cr (Datamedia 2500, Hazeltine 2000)</td>
</tr>
<tr>
<td>nd</td>
<td>str</td>
<td>Non-destructive space (cursor right)</td>
</tr>
<tr>
<td>NL</td>
<td>bool</td>
<td>\n is newline, not line feed</td>
</tr>
<tr>
<td>nl</td>
<td>str</td>
<td>Newline character if not \n</td>
</tr>
<tr>
<td>ns</td>
<td>bool</td>
<td>Terminal is a CRT but doesn't scroll</td>
</tr>
<tr>
<td>nw</td>
<td>str</td>
<td>Newline (behaves like cr followed by do)</td>
</tr>
<tr>
<td>OP</td>
<td>bool</td>
<td>Odd parity</td>
</tr>
<tr>
<td>os</td>
<td>bool</td>
<td>Terminal overstrikes</td>
</tr>
<tr>
<td>pb</td>
<td>num</td>
<td>Lowest baud where delays are required</td>
</tr>
<tr>
<td>pc</td>
<td>str</td>
<td>Pad character (default NUL)</td>
</tr>
<tr>
<td>pf</td>
<td>str</td>
<td>Turn off the printer</td>
</tr>
<tr>
<td>pk</td>
<td>str</td>
<td>Program function key n to type string s (terminfo only)</td>
</tr>
<tr>
<td>pl</td>
<td>str</td>
<td>Program function key n to execute string s (terminfo only)</td>
</tr>
<tr>
<td>pO</td>
<td>str</td>
<td>(N) Turn on the printer for n bytes</td>
</tr>
<tr>
<td>po</td>
<td>str</td>
<td>Turn on the printer</td>
</tr>
<tr>
<td>ps</td>
<td>str</td>
<td>Print contents of the screen</td>
</tr>
<tr>
<td>pt</td>
<td>bool</td>
<td>Has hardware tabs (may need to be set with is)</td>
</tr>
<tr>
<td>px</td>
<td>str</td>
<td>Program function key n to transmit string s (terminfo only)</td>
</tr>
<tr>
<td>r1-r3</td>
<td>str</td>
<td>Reset terminal completely to sane modes (terminfo only)</td>
</tr>
<tr>
<td>rc</td>
<td>str</td>
<td>(P) Restore cursor to position of last sc</td>
</tr>
<tr>
<td>rf</td>
<td>str</td>
<td>Name of file containing reset codes</td>
</tr>
<tr>
<td>RI</td>
<td>str</td>
<td>(NP) Move cursor right n positions</td>
</tr>
<tr>
<td>rp</td>
<td>str</td>
<td>(NP*) Repeat character c n times</td>
</tr>
<tr>
<td>rs</td>
<td>str</td>
<td>Reset terminal completely to sane modes (termcap only)</td>
</tr>
<tr>
<td>sa</td>
<td>str</td>
<td>(NP) Define the video attributes</td>
</tr>
<tr>
<td>sc</td>
<td>str</td>
<td>(P) Save cursor position</td>
</tr>
<tr>
<td>se</td>
<td>str</td>
<td>End standout mode</td>
</tr>
<tr>
<td>SF</td>
<td>str</td>
<td>(NP*) Scroll forward n lines</td>
</tr>
<tr>
<td>sf</td>
<td>str</td>
<td>(P) Scroll text up</td>
</tr>
<tr>
<td>sg</td>
<td>num</td>
<td>Number of garbage chars left by so or se (default 0)</td>
</tr>
<tr>
<td>so</td>
<td>str</td>
<td>Begin standout mode</td>
</tr>
<tr>
<td>SR</td>
<td>str</td>
<td>(NP*) Scroll backward n lines</td>
</tr>
<tr>
<td>sr</td>
<td>str</td>
<td>(P) Scroll text down</td>
</tr>
<tr>
<td>st</td>
<td>str</td>
<td>Set a tab in all rows, current column</td>
</tr>
<tr>
<td>ta</td>
<td>str</td>
<td>(P) Tab to next 8-position hardware tab stop</td>
</tr>
<tr>
<td>tc</td>
<td>str</td>
<td>Entry of similar terminal – must be last</td>
</tr>
<tr>
<td>te</td>
<td>str</td>
<td>String to end programs that use termcap</td>
</tr>
<tr>
<td>ti</td>
<td>str</td>
<td>String to begin programs that use termcap</td>
</tr>
<tr>
<td>ts</td>
<td>str</td>
<td>(N) Go to status line, column n</td>
</tr>
<tr>
<td>UC</td>
<td>bool</td>
<td>(o) Upper-case only</td>
</tr>
<tr>
<td>uc</td>
<td>str</td>
<td>Underscore one character and move past it</td>
</tr>
<tr>
<td>ue</td>
<td>str</td>
<td>End underscore mode</td>
</tr>
<tr>
<td>ug</td>
<td>num</td>
<td>Number of garbage chars left by us or ue (default 0)</td>
</tr>
<tr>
<td>ul</td>
<td>bool</td>
<td>Underline character overstrikes</td>
</tr>
<tr>
<td>UP</td>
<td>str</td>
<td>(NP*) Move cursor up n lines</td>
</tr>
<tr>
<td>up</td>
<td>str</td>
<td>Upline (cursor up)</td>
</tr>
<tr>
<td>us</td>
<td>str</td>
<td>Start underscore mode</td>
</tr>
<tr>
<td>vb</td>
<td>str</td>
<td>Visible bell (must not move cursor)</td>
</tr>
<tr>
<td>ve</td>
<td>str</td>
<td>Make cursor appear normal (undo vs/vi)</td>
</tr>
</tbody>
</table>
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
ws num Number of columns in status line
xb bool Beehive (f1 = ESC, f2 = “C”)
xn bool Newline ignored after 80 cols (Concept)
ox bool Terminal uses xoff/xon (DC3/DC1) handshaking
xr bool (o) Return acts like ce cr nl (Delta Data)
xs bool Standout not erased by overwriting (Hewlett-Packard)
xh bool Tabs ruin, magic so char (Teleray 1061)
xx bool (o) Tektronix 4025 insert-line

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.

cap | concept.100 | concept | c104 | concept100-4p | HDS Concept-100 Handle

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 ‘=’, then a string ending at the next following ‘:’. A delay in milliseconds may appear after the ‘=’ 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 ‘*’, i.e., ‘3*’. An ‘*’ 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 ‘*’ is specified, it is sometimes useful to give a delay of the form

3rd Berkeley Distribution 1 November 1985
'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 \f map to linefeed, return, tab, backspace, and formfeed, respectively. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\.

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the first cr 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 TERMCP to the absolute pathname of a file containing the description you are working on and programs will look there rather than in /etc/termcap. TERMCP 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 n 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 be 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 ad, 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 aw (newline). It is permissible for this to clear the remainder of the current line, so if the terminal has no correctly-working CR and LF it may still be possible to craft a working aw 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

T3 | tty33 | 33 | tty | Teletype model 33:\n:bl="G:co#72:cr="M:do="J:h:cos:

and the Lear Siegler ADM-3 is described as


Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with printf(3S)-like escapes %x in it, while other characters are passed through unchanged. For example, to address the cursor the 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 CM capability.)

The % encodings have the following meanings:

<table>
<thead>
<tr>
<th>%</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>output &quot;%&quot;</td>
</tr>
<tr>
<td>%d</td>
<td>output value as in printf %d</td>
</tr>
<tr>
<td>%2</td>
<td>output value as in printf %2d</td>
</tr>
<tr>
<td>%3</td>
<td>output value as in printf %3d</td>
</tr>
<tr>
<td>%.</td>
<td>output value as in printf %c</td>
</tr>
<tr>
<td>%+x</td>
<td>add x to value, then do %</td>
</tr>
<tr>
<td>%&gt;xy</td>
<td>if value &gt; x then add y, no output</td>
</tr>
<tr>
<td>%r</td>
<td>reverse order of two parameters, no output</td>
</tr>
<tr>
<td>%i</td>
<td>increment by one, no output</td>
</tr>
<tr>
<td>%n</td>
<td>exclusive-or all parameters with 0140 (Datamedia 2500)</td>
</tr>
<tr>
<td>%B</td>
<td>BCD (16*(value/10)) + (value%10), no output</td>
</tr>
<tr>
<td>%D</td>
<td>Reverse coding (value - 2*(value%16)), no output (Delta Data)</td>
</tr>
</tbody>
</table>

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent "\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 cm capability is "cm=6\E%r%2c%2Y".

The Microterm ACT-IV needs the current row and column sent simply encoded in binary preceded by a "T, cm="T%.". Terminals that use "%" need to be able to backspace the cursor (le) and to move the cursor up one line on the screen (up). This is necessary because it is not always safe to transmit \n, \D, and \r, as the system may change or discard them.
(Programs using termcap must set terminal modes so that tabs are not expanded, so \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 “cm=\E=\%=\%+”.

Row or column absolute cursor addressing can be given as single parameter capabilities ch (horizontal position absolute) and 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 cm. If there are parameterized local motions (e.g., move n positions to the right) these can be given as DO, LE, RI, and UP with a single parameter indicating how many positions to move. These are primarily useful if the terminal does not have cm, 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 ho. Similarly, a fast way of getting to the lower left-hand corner can be given as ll; this may involve going up with up from the home position, but a program should never do this itself (unless 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 “\\EH” sequence on Hewlett-Packard terminals cannot be used for 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 ce. If the terminal can clear from the current position to the end of the display, this should be given as cd. cd 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 cd 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 al; 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 dl; this must only be used from the first position on the line to be deleted. Versions of al and dl which take a single parameter and insert or delete that many lines can be given as AL and DL. If the terminal has a settable scrolling region (like the VT100), the command to set this can be described with the cs 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 sc and re (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using sr or 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 wi. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order. (This terminfo capability is described for completeness. It is unlikely that any termcap-using program will support it.)

If the terminal can retain display memory above the screen, then the da capability should be given; if display memory can be retained below, then 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 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 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 "abc  def" using local cursor motions (not spaces) between the "abc" and the "def". Then position the cursor before the "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 ie any sequence that needs to be sent just before each character to be inserted. Most terminals with a true insert mode will not give ie; 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 ie. 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/el and ie can be given, and both will be used. The IC capability, with one parameter n, will repeat the effects of ie 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 Datamedia'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 ro through f9, the labels can be given as 10, 11, 19. 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 supplant 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=AHhAKfZk"` 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 `ti` and `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 screensized window must be fixed into the display for cursor addressing to work properly. This is also used for the Tektronix 4025, where `ti` sets the command character to be the one used by `termcap`.

Other capabilities include `is`, an initialization string for the terminal, and `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 `termcap` description. They are normally sent to the terminal by the `tset` program each time the user logs in. They will be printed in the following order: `is`; setting tabs using `ct` and `st`; and finally `if`. (`Terminfo` uses `il-il` instead of `is` and runs the program `iP` and prints `i3` after the other initializations.) A pair of sequences that does a harder reset from a totally unknown state can be analogously given as `rs` and `rf`. These strings are output by the `reset` program, which is used when the terminal gets into a wedged state. (`Terminfo` uses `rl-r3` instead of `rs`.) Commands are normally placed in `rs` and `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 `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 `ta` (usually `'T`). A “backtab” command which moves leftward to the previous tab stop can be given as `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 `ta` or `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 `n` positions when the terminal is powered up, then the numeric parameter it is given, showing the number of positions between tab stops. This is normally used by the `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 `termcap` description can assume that they are properly set.

If there are commands to set and clear tab stops, they can be given as `ct` (clear all tab stops) and `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 `is` or `if`.

Delays

Certain capabilities control padding in the terminal driver. These are primarily needed by hardcopy terminals and are used by the `tset` program to set terminal driver modes appropriately. Delays embedded in the capabilities `cr`, `sf`, `le`, `ff`, and `ta` will cause the appropriate delay bits to be set in the terminal driver. If `pb` (padding baud rate) is given, these values can be ignored at baud rates below the value of `pb`. For 4.2BSD `tset`, the delays are given as numeric capabilities `dC`, `dN`, `dB`, `dF`, and `dT` instead.
Miscellaneous

If the terminal requires other than a NUL (zero) character as a pad, this can be given as \texttt{pc}. Only the first character of the \texttt{pc} string is used.

If the terminal has commands to save and restore the position of the cursor, give them as \texttt{sc} 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 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{AL}).

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 \textit{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 \textit{virtual} terminal descriptions for which the escape sequences are known.)

If the terminal uses \texttt{xoff/xon} (\texttt{DC3/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{im}. 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 virtual terminal protocol, the terminal number can be given as vt.

Media copy strings which control an auxiliary printer connected to the terminal can be given as ps: print the contents of the screen; pf: turn off the printer; and 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 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, where tabs 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), more(1), tset(1), ul(1), vi(1), curses(3X), printf(3S), term(7).

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.

Lines and columns are now stored by the kernel as well as in the termcap entry. Most programs now use the kernel information primarily; the information in this file is used only if the kernel does not have any information.

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.
NAME
  tp – DEC/mag tape formats

DESCRIPTION
  Tp dumps files to and extracts files from DECtape and magtape. The formats of these tapes
  are the same except that magtapes have larger directories.

  Block zero contains a copy of a stand-alone bootstrap program. See reboot(8).

  Blocks 1 through 24 for DECtape (1 through 62 for magtape) contain a directory of the tape.
  There are 192 (resp. 496) entries in the directory; 8 entries per block; 64 bytes per entry.
  Each entry has the following format:

    struct {
        char         pathname[32];
        unsigned short mode;
        char         uid;
        char         gid;
        char         unused1;
        char         size[3];
        long         modtime;
        unsigned short tapeaddr;
        char         unused2[16];
        unsigned short checksum;
    };

  The path name entry is the path name of the file when put on the tape. If the pathname
  starts with a zero word, the entry is empty. It is at most 32 bytes long and ends in a null
  byte. Mode, uid, gid, size and time modified are the same as described under i-nodes (see file
  system ft(5)). The tape address is the tape block number of the start of the contents of the
  file. Every file starts on a block boundary. The file occupies (size+511)/512 blocks of con-
NAME
ttys – terminal initialization data

DESCRIPTION
The ttys file contains information that is used by various routines to initialize and control the use of terminal special files. This information is read with the gettyent(3) library routines. There is one line in the ttys file per special file. Fields are separated by tabs and/or spaces. Some fields may contain more than one word and should be enclosed in double quotes. Blank lines and comments can appear anywhere in the file; comments are delimited by ‘#’ and new line. Unspecified fields default to null. The first field is the terminal’s entry in the device directory, /dev. The second field of the file is the command to execute for the line, typically getty(8), which performs such tasks as baud-rate recognition, reading the login name, and calling login(1). It can be, however, any desired command, for example the start up for a window system terminal emulator or some other daemon process, and can contain multiple words if quoted. The third field is the type of terminal normally connected to that tty line, as found in the termcap(5) data base file. The remaining fields set flags in the ty_status entry (see gettyent(3)) or specify a window system process that init(8) will maintain for the termin­ nal line. As flag values, the strings ‘on’ and ‘off’ specify whether init should execute the command given in the second field, while ‘secure’ in addition to ‘on’ allows root to login on this line. These flag fields should not be quoted. The string ‘window=’ is followed by a quoted command string which init will execute before starting getty. If the line ends in a comment, the comment is included in the ty_comment field of the ttyent structure.

Some examples:

<table>
<thead>
<tr>
<th>console</th>
<th>/etc/getty std.1200</th>
<th>vt100</th>
<th>on secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttyd0</td>
<td>/etc/getty d1200</td>
<td>dialup</td>
<td>on</td>
</tr>
<tr>
<td>ttyh0</td>
<td>/etc/getty std.9600</td>
<td>hp2621-nl</td>
<td>on</td>
</tr>
<tr>
<td>ttyh1</td>
<td>/etc/getty std.9600</td>
<td>plugboard</td>
<td>on</td>
</tr>
<tr>
<td>ttyp0</td>
<td>none</td>
<td>network</td>
<td></td>
</tr>
<tr>
<td>ttypl</td>
<td>none</td>
<td>network</td>
<td>off</td>
</tr>
<tr>
<td>ttyv0</td>
<td>/usr/new/xterm -L :0</td>
<td>vs100</td>
<td>on window=’/usr/new/Xvs100 0’</td>
</tr>
</tbody>
</table>

The first example permits root login on the console at 1200 baud, the second allows dialup at 1200 baud without root login, the third and fourth allow login at 9600 baud with terminal types of "hp2621-nl" and "plugboard" respectively, the fifth and sixth line are examples of network pseudo ttys, which should not have getty enabled on them, and the last example shows a terminal emulator and window system startup entry.

FILES
/etc/ttys

SEE ALSO
login(1), gettyent(3), gettytab(5), init(8), getty(8)
NAME
  types – primitive system data types

SYNOPSIS
  #include <sys/types.h>

DESCRIPTION
  The data types defined in the include file are used in UNIX system code; some data of these
types are accessible to user code:

  /*
  * Copyright (c) 1982 Regents of the University of California.
  * All rights reserved. The Berkeley software License Agreement
  * specifies the terms and conditions for redistribution.
  *
  * @(#)types.h 6.8 (Berkeley) 3/28/86
  */

#ifndef _TYPES_
#define _TYPES_
/*
 * Basic system types and major/minor device constructing/busting macros.
 */

/* major part of a device */
#define major(x) ((int)(((unsigned)(x)>>8)&0377))

/* minor part of a device */
#define minor(x) ((int)((x)&0377))

/* make a device number */
#define makedev(x,y) ((dev_t)(((x)<<8) | (y)))

typedef unsigned char   u_char;
typedef unsigned short  u_short;
typedef unsigned int    u_int;
typedef unsigned long   u_long;
typedef unsigned short  ushort; /* sys III compat */

#ifdef vax
typedef struct  _physadr { int r[1]; } *physadr;
typedef struct  label_t {
    int       val[14];
 } label_t;
#endif
typedef struct  _quad { long val[2]; } quad;
typedef long    daddr_t;
typedef char *  caddr_t;
typedef u_long   ino_t;
typedef long    swblk_t;
typedef long    size_t;
typedef long    time_t;
typedef short   dev_t;
typedef long    off_t;
typedef u_short  uid_t;

4th Berkeley Distribution      May 15, 1985
typedef u_short gid_t;

#define NBBY 8 /* number of bits in a byte */

/*
 * Select uses bit masks of file descriptors in longs.
 * These macros manipulate such bit fields (the filesystem macros use chars).
 * FD_SETSIZE may be defined by the user, but the default here
 * should be >= NOFILE (param.h).
 */
#ifndef FD_SETSIZE
#define FD_SETSIZE 256
#endif

typedef long fd_mask;
#define NFDBITS (sizeof(fd_mask) * NBBY) /* bits per mask */
#ifndef howmany
#define howmany(x, y) (((x)+((y)-1))/(y))
#endif

typedef struct fd_set {
    fd_mask fds_bits[howmany(FD_SETSIZE, NFDBITS)];
} fd_set;

#define FD_SET(n, p) ((p)->fds_bits[(n)NFDBITSll = (1 « (n) « NBBY) « ((n) % NFDBITS))
#define FD_CLR(n, p) ((p)->fds_bits[(n)NFDBITS] & = ~(1 « (n) « NBBY) « (n) % NFDBITS))
#define FD_ISSET(n, p) ((p)->fds_bits[(n)NFDBITS] & (1 « (n) % NFDBITS))
#define FD_ZERO(p) bzero(char *)((p), sizeof(*(p)))

#endif

The form daddr_t is used for disk addresses except in an i-node on disk, see fs(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The label_t variables are used to save the processor state while another process is running.

SEE ALSO
fs(5), time(3), lseek(2), adb(1)
NAME
utra, wtmp - login records

SYNOPSIS
#include <utmp.h>

DESCRIPTION
The utmp file records information about who is currently using the system. The file is a sequence of entries with the following structure declared in the include file:

```c
struct utmp {
    char ut_line[8];        /* tty name */
    char ut_name[8];        /* user id */
    char ut_host[16];       /* host name, if remote */
    long ut_time;           /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of time(3C).

The wtmp file records all logins and logouts. A null user name indicates a logout on the associated terminal. Furthermore, the terminal name '"' indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names ']' and '[' indicate the system-maintained time just before and just after a date command has changed the system's idea of the time.

Wtmp is maintained by login(1) and init(8). Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by ac(8).

FILES
/etc/utmp
/usr/adm/wtmp

SEE ALSO
login(1), init(8), who(1), ac(8)
NAME
uuencode – format of an encoded uuencode file

DESCRIPTION
Files output by uuencode(1C) consist of a header line, followed by a number of body lines, and a trailer line. Uudecode(1C) will ignore any lines preceding the header or following the trailer. Lines preceding a header must not, of course, look like a header.

The header line is distinguished by having the first 6 characters “begin ”. The word begin is followed by a mode (in octal), and a string which names the remote file. A space separates the three items in the header line.

The body consists of a number of lines, each at most 62 characters long (including the trailing newline). These consist of a character count, followed by encoded characters, followed by a newline. The character count is a single printing character, and represents an integer, the number of bytes the rest of the line represents. Such integers are always in the range from 0 to 63 and can be determined by subtracting the character space (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a space to make the characters printing. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra garbage will be included to make the character count a multiple of 4. The body is terminated by a line with a count of zero. This line consists of one ASCII space.

The trailer line consists of “end” on a line by itself.

SEE ALSO
uuencode(1C), uudecode(1C), uusend(1C), uucp(1C), mail(1)
NAME
  vfont - font formats for the Benson-Varian or Versatec

SYNOPSIS
  /usr/lib/vfont/

DESCRIPTION
  The fonts for the printer/plotters have the following format. Each file contains a header, an
array of 256 character description structures, and then the bit maps for the characters themselves. The header has the following format:

struct header {
  short       magic;
  unsigned short size;
  short       maxx;
  short       maxy;
  short       xtnd;
} header;

The magic number is 0436 (octal). The maxx, maxy, and xtnd fields are not used at the
current time. Maxx and maxy are intended to be the maximum horizontal and vertical size
of any glyph in the font, in raster lines. The size is the size of the bit maps for the characters
in bytes. Before the maps for the characters is an array of 256 structures for each of the possible
characters in the font. Each element of the array has the form:

struct dispatch {
  unsigned short      addr;
  short               nbytes;
  char                up;
  char                down;
  char                left;
  char                right;
  short               width;
};

The nbytes field is nonzero for characters which actually exist. For such characters, the addr
field is an offset into the rest of the file where the data for that character begins. There are
up+down rows of data for each character, each of which has left+right bits, rounded up to a
number of bytes. The width field is not used by vcat, although it is to make width tables for
troff. It represents the logical width of the glyph, in raster lines, and shows where the base
point of the next glyph would be.

FILES
  /usr/lib/vfont/

SEE ALSO
  troff(1), pti(1), vfontinfo(1)
NAME
vgrindefs - vgrind's language definition data base

SYNOPSIS
/usr/lib/vgrindefs

DESCRIPTION
Vgrindefs contains all language definitions for vgrind. The data base is very similar to
termcap(5).

FIELDS
The following table names and describes each field.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb</td>
<td>str</td>
<td>regular expression for start of a procedure</td>
</tr>
<tr>
<td>bb</td>
<td>str</td>
<td>regular expression for start of a lexical block</td>
</tr>
<tr>
<td>be</td>
<td>str</td>
<td>regular expression for the end of a lexical block</td>
</tr>
<tr>
<td>cb</td>
<td>str</td>
<td>regular expression for the start of a comment</td>
</tr>
<tr>
<td>ce</td>
<td>str</td>
<td>regular expression for the end of a comment</td>
</tr>
<tr>
<td>sb</td>
<td>str</td>
<td>regular expression for the start of a string</td>
</tr>
<tr>
<td>se</td>
<td>str</td>
<td>regular expression for the end of a string</td>
</tr>
<tr>
<td>lb</td>
<td>str</td>
<td>regular expression for the start of a character constant</td>
</tr>
<tr>
<td>le</td>
<td>str</td>
<td>regular expression for the end of a character constant</td>
</tr>
<tr>
<td>tl</td>
<td>bool</td>
<td>present means procedures are only defined at the top lexical level</td>
</tr>
<tr>
<td>oc</td>
<td>bool</td>
<td>present means upper and lower case are equivalent</td>
</tr>
<tr>
<td>kw</td>
<td>str</td>
<td>a list of keywords separated by spaces</td>
</tr>
</tbody>
</table>

Example
The following entry, which describes the C language, is typical of a language entry.

C | :|p:b=\d?\d?\d?:bb=\.:ce=\./se=:e=:e:|\
  :|l:b=\e:|\|\n  :|k:w=asm auto break case char continue default do double else enum\extern float for fortran goto if int long register return short\sizeof static struct switch typedef union unsigned while #define|#else #endif #if #ifndef #include \#undef #ifdef if ifndef include undef if ifdef ifndef include undef:

Note that the first field is just the language name (and any variants of it). Thus the C language could be specified to vgrind(1) as "c" or "C".

Entries may continue onto multiple lines by giving a \ as the last character of a line. Capabilities in vgrindefs are of two types: Boolean capabilities which indicate that the language has some particular feature and string capabilities which give a regular expression or keyword list.

REGULAR EXPRESSIONS
Vgrindefs uses regular expression which are very similar to those of ex(1) and lex(1). The characters '"', '$', ' ' and '\' are reserved characters and must be "quoted" with a preceding \ if they are to be included as normal characters. The metasymbols and their meanings are:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>the end of a line</td>
</tr>
<tr>
<td>^</td>
<td>the beginning of a line</td>
</tr>
<tr>
<td>\d</td>
<td>a delimiter (space, tab, newline, start of line)</td>
</tr>
<tr>
<td>\a</td>
<td>matches any string of symbols (like .* in lex)</td>
</tr>
<tr>
<td>\p</td>
<td>matches any alphanumeric name. In a procedure definition (pb) the string that matches this symbol is used as the procedure name.</td>
</tr>
</tbody>
</table>

4.2 Berkeley Distribution May 15, 1985
() grouping
| alternation
? last item is optional
\ preceding any string means that the string will not match an input string if the input string is preceded by an escape character (\). This is typically used for languages (like C) which can include the string delimiter in a string by escaping it.

Unlike other regular expressions in the system, these match words and not characters. Hence something like "(tramp|steamer)flies?" would match "tramp", "steamer", "trampflies", or "steamerflies".

**KEYWORD LIST**

The keyword list is just a list of keywords in the language separated by spaces. If the "oc" boolean is specified, indicating that upper and lower case are equivalent, then all the keywords should be specified in lower case.

**FILES**
/usr/lib/vgrindefs file containing terminal descriptions

**SEE ALSO**
vgrind(1), troff(1)

**AUTHOR**
Dave Presotto

**BUGS**
NAME

ypfiles — the yellowpages database and directory structure

DESCRIPTION

The yellow pages (YP) network lookup service uses a database of dbm files in the directory hierarchy at /etc/yp. A dbm database consists of two files, created by calls to the dbm(3X) library package. One has the filename extension .pag and the other has the filename extension .dir. For instance, the database named hosts.byname, is implemented by the pair of files hosts.byname.pag and hosts.byname.dir. A dbm database served by the YP is called a YP map. A YP domain is a named set of YP maps. Each YP domain is implemented as a subdirectory of /etc/yp containing the map. Any number of YP domains can exist. Each may contain any number of maps.

No maps are required by the YP lookup service itself, although they may be required for the normal operation of other parts of the system. There is no list of maps which YP serves— if the map exists in a given domain, and a client asks about it, the YP will serve it. For a map to be accessible consistently, it must exist on all YP servers that serve the domain. To provide data consistency between the replicated maps, an entry to run ypfr periodically should be made in /usr/lib/crontab on each server. More information on this topic is in ypfr(8).

YP maps should contain two distinguished key-value pairs. The first is the key YP_LAST_MODIFIED, having as a value a ten-character ASCII order number. The order number should be the UNIX time in seconds when the map was built. The second key is YP_MASTER_NAME, with the name of the YP master server as a value. makedbm generates both key-value pairs automatically. A map that does not contain both key-value pairs can be served by the YP, but the ypserv process will not be able to return values for "Get order number" or "Get master name" requests. In addition, values of these two keys are used by ypfr when it transfers a map from a master YP server to a slave. If ypfr cannot figure out where to get the map, or if it is unable to determine whether the local copy is more recent than the copy at the master, you must set extra command line switches when you run it.

YP maps must be generated and modified only at the master server. They are copied to the slaves using ypfr(8) to avoid potential byte-ordering problems among YP servers running on machines with different architectures, and to minimize the amount of disk space required for the dbm files. The YP database can be initially set up for both masters and slaves by using ypinit(8).

After the server databases are set up, it is probable that the contents of some maps will change. In general, some ASCII source version of the database exists on the master, and it is changed with a standard text editor. The update is incorporated into the YP map and is propagated from the master to the slaves by running /etc/yp/Makefile. All Sun-supplied maps have entries in /etc/yp/Makefile; if you add a YP map, edit this file to support the new map. The makefile uses makedbm to generate the YP map on the master, and yppush to propagate the changed map to the slaves. yppush is a client of the map ypservers, which lists all the YP servers. For more information on this topic, see yppush(8).

SEE ALSO

makedbm(8), ypinit(8), ypmake(8), ypfr(8), yppush(8), yppoll(8), ypserv(8), rpcinfo(8).