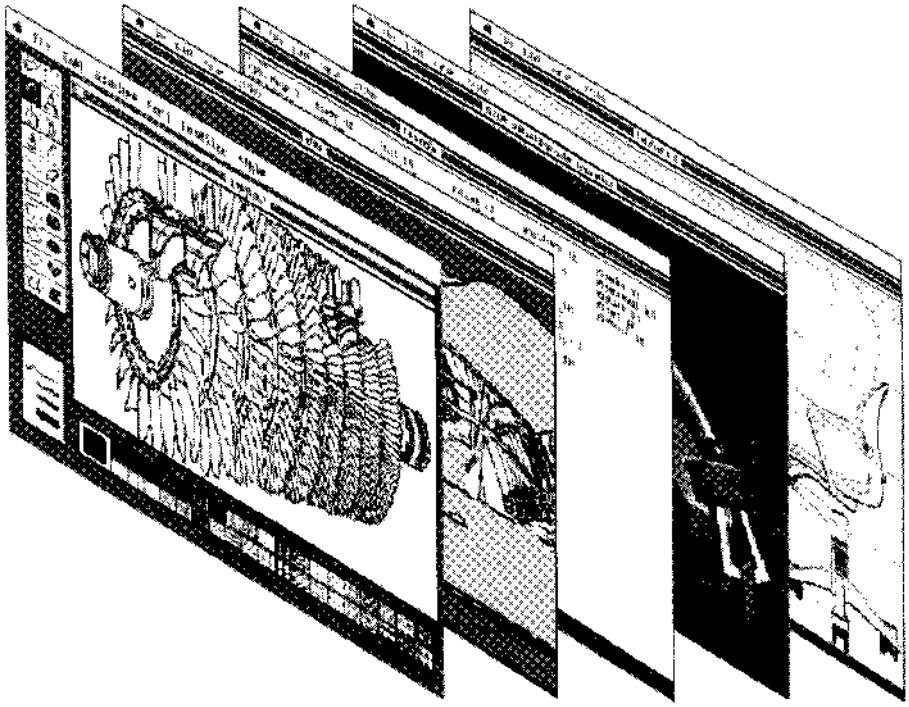
# Apple® A/UX™ Programmer's Reference



## Copyright



Apple Computer, Inc.
20525 Mariani Ave.
Cupertino, California 95014
(408) 996-1010



## Limited Warranty on Media and Replacement

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the media or manuals at no charge to you, provided you return the item to be replaced with proof of purchase to Apple or an authorized Apple dealer during the 90-day period after you purchased the software. In addition, Apple will replace damaged software media and manuals for as long as the software product is included in Apple's Media Exchange Program. While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apple dealer for program coverage and details. In some countries the replacement period may be different; check with your authorized Apple dealer.

**ALL IMPLIED WARRANTIES ON THE**

**MEDIA AND MANUALS, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.**

Even though Apple has tested the software and reviewed the documentation, **APPLE AND ITS SOFTWARE SUPPLIER MAKE NO WARRANTIES OR REPRESENTATIONS, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO SOFTWARE, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD AS IS, AND YOU THE PURCHASER ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE.**

**IN NO EVENT WILL APPLE OR ITS SOFTWARE SUPPLIER BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR ITS DOCUMENTATION,** even if advised of the possibility of such damages. In particular, Apple and its software supplier shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering such programs or data.

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED.** No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

# A/UX Programmer's Reference

## Contents


Preface

Introduction

Section 2 System Calls

Section 3 Subroutines

Section 4 File Formats

Section 5 Miscellaneous Facilities

Appendix A Permuted Index

# Preface

## Conventions Used in This Manual

Throughout the A/UX manuals, words that must be typed exactly as shown or that would actually appear on the screen are in `Courier` type. Words that you must replace with actual values appear in *italics* (for example, *user-name* might have an actual value of `joe`). Key names appear in CAPS (for example, RETURN). Special terms are in **bold** type when they are introduced; many of these terms are also defined in the glossary in the *A/UX System Overview*.

### Syntax notation

All A/UX manuals use the following conventions to represent command syntax. A typical A/UX command has the form

`command` [*flag-option*] [*argument*]...

where:

| | |
|---|---|
| `command` | Command name (the name of an executable file). |
| *flag-option* | One or more flag options. Historically, flag options have the form<br>-[*opt*...]<br>where *opt* is a letter representing an option. The form of flag options varies from program to program. Note that with respect to flag options, the notation<br>`[-a][-b][-c]`<br>means you can select one or more letters from the list enclosed in brackets. If you select more than one letter you use only one hyphen, for example, `-ab`. |
| *argument* | Represents an argument to the command, in this context usually a filename or symbols representing one or more filenames. |
| [] | Surround an optional item. |

| | |
|---|---|
| ... | Follows an argument that may be repeated any number of times. |
| `Courier type` | anywhere in the syntax diagram indicates that characters must be typed literally as shown. |
| *italics* | for an argument name indicates that a value must be supplied for that argument. |

Other conventions used in this manual are:

| | |
|---|---|
| <CR> | indicates that the RETURN key must be pressed. |
| ^*x* | An abbreviation for CONTROL-*x*, where *x* may be any key. |
| *cmd*(*sect*) | A cross-reference to an A/UX reference manual. *cmd* is the name of a command, program, or other facility, and *sect* is the section number where the entry resides. For example, `cat`(1). |

# Introduction

# to the A/UX Reference Manuals

## 1. How to use the reference manuals

The *A/UX Command Reference*, *A/UX Programmer's Reference*, and *A/UX System Administrator's Reference* are reference manuals for all the programs and utilities included with your A/UX system. These manuals provide complete information on these programs and utilities, but they are designed for quick reference and are not tutorials. If you are just learning the A/UX system, or are unfamiliar with a group of programs (such as the shells or the text formatting programs) you should first refer to *Getting Started With A/UX* and the narrative user guides provided with your system. After you have worked with the A/UX system, use these reference manuals to look up a new command or refresh your memory on a command you already know.

## 2. Information contained in the reference manuals

A/UX reference manuals are divided into three volumes:

- The 2-part *A/UX Command Reference* contains information for the general user. It describes commands you type at the A/UX prompt which list your files, compile programs, format text, change your shell, and so on. It also includes programs used in scripts and command language procedures. The commands in this manual generally reside in the directories `/bin`, `/usr/bin` and `/usr/ucb`.
- The *A/UX Programmer's Reference* contains information for the programmer. It describes utilities for programming, such as system calls, subroutines file formats, and miscellaneous programming facilities.
- The *A/UX System Administrator's Reference* contains information for the system administrator. It describes commands you type at the A/UX prompt to control your machine, such as

accounting commands, backing up your system, and charting your system's activity. These commands generally reside in the directories `/etc`, `/usr/etc`, and `/usr/lib`.

These areas can overlap. For example, if you are the only person using your machine, then you are both the general user and the system administrator.

## 3. How the reference manuals are organized

All manual pages are grouped by section. The sections are grouped by general function and are numbered according to standard conventions as follows:

1 User commands

1M System maintenance commands

2 System calls

3 Subroutines

4 File formats

5 Miscellaneous facilities

6 Games

7 Special files (files that refer to devices)

8 System maintenance procedures

Each of the reference manuals is divided into two or more sections and lists each command or utility alphabetically within each section. The sections included in each volume are as follows:

The *A/UX Command Reference* contains sections 1 and 6. Note that both of these sections describe commands and programs available to the general user.

- Section 1
  The commands in Section 1 fall into four categories. These categories are indicated next to the command name at the top of

the page:

1 General-purpose commands, such as `cat` and `ls`.

1C Communications commands, such as `cu` and `tip`.

1G Graphics commands, such as `graph` and `tplot`.

1N Networking commands used by the B-NET program and NFS such as `rcp` and `ypcat`.

- Section 6
  This contains all the games, such as `cribbage` and `worms`.

The *A/UX Programmer's Reference* contains sections 2 through 5.

- Section 2—System Calls
  This describes the services provided by the A/UX system kernel, including the C language interface. It includes two categories (indicated next to the command name at the top of the page):

  2 General system calls

  2N Networking system calls

- Section 3—Subroutines
  This describes the available subroutines. The binary versions are in the system libraries in the `/lib` and `/usr/lib` directories. This section includes six categories (indicated next to the command name at the top of the page):

  3C C and assembler library routines

  3F Fortran library routines

  3M Mathematical library routines

  3N Networking routines

  3S Standard I/O library routines

  3X Miscellaneous routines

- Section 4—File Formats
  This describes the structure of some files, but does not include

files that are used by only one command (such as the assembler's intermediate files). The C language `struct` declarations corresponding to these formats are in the `/usr/include` and `/usr/include/sys` directories. There are two categories in this section (indicated next to the file name at the top of the page):

4 General file formats

4N Networking formats

- Section 5—Miscellaneous facilities
  This section contains various character sets, macro packages, etc. There are three categories in this section (indicated next to the name at the top of the page):

  5 General miscellaneous facilities

  5F Protocol families

  5P Protocol descriptions

The *A/UX System Administrator's Reference* contains sections 1M, 7 and 8.

- Section 1M—System Maintenance Commands
  This section contains system maintenance programs such as `fsck` and `mkfs`.
- Section 7—Special Files
  This section discusses special files that refer to specific hardware peripherals and system device drivers. The names in this section generally refer to device names for the hardware, rather than to the names of the special files themselves.
- Section 8—System Maintenance Procedures
  This section includes crash recovery and boot procedures and the standalone environment.

## 4. How a manual entry is organized

Each section of the reference volumes has an introduction and several entries arranged alphabetically. The entry name and its category (for example 1M or 2N) appear in the upper corners of each page. Each entry is numbered separately (that is, each entry begins on a page numbered "1").

Some entries describe several routines or commands. These multiple subentries are listed under the main entry, and each subentry refers you back to the main entry. For example, `chown` and `chgrp` share a page with the name `chown`(1) at the upper corners. If you turn to the page `chgrp`(1), you will find a reference to `chown`(1). (This is true only for the *A/UX Command Reference* and *A/UX System Administrator's Reference*.

All of the entries have a common format, and may include any of the following parts:

**NAME**
the name(s) and a brief description.

**SYNOPSIS**
describes the syntax for using the command or routine.

**DESCRIPTION**
discusses what the program does.

**EXAMPLE**
gives example(s) of usage.

**RETURN VALUE**
describes the value returned by a function.

**ERRORS**
describes the possible error conditions.

**FILES**
lists the file names that are used by the program.

**SEE ALSO**
provides pointers to related information.

**DIAGNOSTICS**
discusses the diagnostic messages that may be produced. Self-explanatory messages are not listed.

**WARNINGS**
points out potential pitfalls.

**BUGS**
gives known bugs and sometimes deficiencies. Occasionally, it describes the suggested fix.

## 5. Locating information in the reference manuals

The *A/UX Command Reference* and *A/UX System Administrator's Reference* have four summaries to help you locate information. The *A/UX Programmer's Reference* contains two summaries: the table of contents and the permuted index.

### 5.1 Table of contents

Each book contains an overall table of contents and individual chapter table of contents. The general table of contents lists the overall contents of each volume. The more detailed chapter table of contents lists the manual pages contained in each section and a brief description of their function. Note that they appear in alphabetic order within each section.

### 5.2 Command summary by function

This summary groups the commands in the *A/UX Command Reference* and *A/UX System Administrator's Reference* by their general function. This will give you some idea of the commands that are available and how they are used.

### 5.3 Command synopses

This lists the synopsis of the commands in the *A/UX Command Reference* and *A/UX System Administrator's Reference* and is provided as an even briefer reference to help you use commands you are already familiar with.

## 5.4 Permuted index

The permuted index lists commands by the information in the NAME part of each entry. The permuted index contains three columns. The center column is sorted alphabetically by keywords that describe the basic function you may be looking for. When you use the permuted index, you should scan the first words in the center column to find the general area of functionality for various commands.

For example, to look for a text editor, scan the center column for the word "editor." There are several index lines containing an "editor" reference, e.g.:

| | | |
|---|---|---|
| ed, red: text | editor | ed(1) |
| files. ld: link | editor for common object | ld(1) |

The first column contains the the rest of the information that either precedes or follows the keyword in the description of a command's function, and the third column shows the manual page where the command is fully described. This entry is followed by the appropriate section number in parentheses.

You can then turn to the entries listed in the last column, `ed`(1) and `ld`(1), to find information on that editor.

## 5.5 On-line manual pages

You can call up these entries on-line with the `man`(1) command. Just type `man` and the name of the entry you want to look at.

If you are not sure of the manual page name, you can use the `apropos` command with the name of a related command or general function, for example:

```
apropos compile
```

shows you commands related to compiling (see `apropos`(1) for more information).

# Table of Contents

## Section 2: System Calls


intro.....introduction to system calls and error numbers
_exit.....see exit(2)
accept.....accept a connection on a socket
access.....determine accessibility of a file
acct.....enable or disable process accounting
adjtime.....correct the time to allow synchronization of system clock
alarm.....set a process's alarm clock
async_daemon.....see nfssvc(2)
bind.....bind a name to a socket
brk.....change data segment space allocation
chdir.....change working directory
chmod.....change mode of file
chown.....change owner and group of a file
chroot.....change root directory
close.....close a file descriptor
connect.....initiate a connection on a socket
creat.....create a new file or rewrite an existing one
dup.....duplicate a descriptor
exec.....execute a file
execl.....see exec(2)
execle.....see exec(2)
execlp.....see exec(2)
execv.....see exec(2)
execve.....see exec(2)
execvp.....see exec(2)
exit.....terminate process
fchown.....see chown(2)
fcntl.....file control
flock.....apply or remove an advisory lock on an open file
fork.....create a new process
fsmount.....mount an NFS file system
fstat.....see stat(2)

fsync .....synchronize a file's in-core state with that on disk
ftruncate.....see truncate(2)
getcompat.....see setcompat(2)
getdirentries .....get directory entries in a file system independent format
getdomainname .....get/set name of current network domain
getdtablesize.....get descriptor table size
getegid.....see getuid(2)
geteuid.....see getuid(2)
getgid.....see getuid(2)
getgroups.....get group access list
gethostid.....get/set unique identifier of current host
gethostname .....get/set name of current host
getitimer.....get/set value of interval timer
getpeername .....get name of connected peer
getpgrp.....see getpid(2)
getpid.....get process, process group, and parent process IDs
getppid.....see getpid(2)
getsockname .....get socket name
getsockopt.....get and set options on sockets
gettimeofday.....get/set date and time
getuid...get real user, effective user, real group, and effective group IDs
ioctl.....control device
kill.....send a signal to a process or a group of processes
link .....link to a file
listen.....listen for connections on a socket
locking.....provide exclusive file regions for reading or writing
lseek.....move read/write file pointer
lstat.....see stat(2)
mkdir.....make a directory file
mknod .....make a directory, or a special or ordinary file
msgctl .....message control operations
msgget.....get message queue
msgop.....message operations
msgrcv .....see msgop(2)
msgsnd .....see msgop(2)
nfs_getfh .....get a file handle
nfssvc .....NFS daemons
nice .....change priority of a process

open .....open for reading or writing
pause .....suspend process until signal
phys .....allow a process to access physical addresses
pipe .....create an interprocess channel
plock .....lock process, text, or data in memory
profil .....execution time profile
ptrace .....process trace
read .....read from file
readlink .....read value of a symbolic link
readv .....see read(2)
reboot .....reboot the system
recv .....receive a message from a socket
recvfrom .....see recv(2N)
recvmsg .....see recv(2N)
rename .....change the name of a file
rmdir .....remove a directory file
sbrk .....see brk(2)
select .....synchronous I/O multiplexing
semctl .....semaphore control operations
semget .....get set of semaphores
semop .....semaphore operations
send .....send a message from a socket
sendmsg .....see send(2N)
sendto .....see send(2N)
setcompat .....set or get process compatibility mode
setdomainname .....see getdomainname(2N)
setgid .....see setuid(2)
setgroups .....set group access list
sethostid .....see gethostid(2N)
sethostname .....see gethostname(2N)
setitimer .....see getitimer(2)
setpgrp .....set process group ID
setregid .....set real and effective group ID
setreuid .....set real and effective user ID's
setsockopt .....see getsockopt(2N)
settimeofday .....see gettimeofday(2)
setuid .....set user and group IDs
shmat .....see shmop(2)

shmctl .....shared memory control operations
shmdt .....see shmop(2)
shmget .....get shared memory segment
shmop .....shared memory operations
shutdown .....shut down part of a full-duplex connection
sigblock .....block signals
sigpause .....atomically release blocked signals and wait for interrupt
sigsetmask .....set current signal mask
sigstack .....set and/or get signal stack context
sigvec .....optional BSD-compatible software signal facilities
socket .....create an endpoint for communication
stat .....get file status
statfs .....get file system statistics
stime .....set time
symlink .....make symbolic link to a file
sync .....update superblock
time .....get time
times .....get process and child process times
truncate .....truncate a file to a specified length
ulimit .....get and set user limits
umask .....set and get file creation mask
umount .....unmount a file system
uname .....get name of current system
unlink .....remove directory entry
unmount .....remove a file system
ustat .....get file system statistics
utime .....set file access and modification times
uvar .....returns system-specific configuration information
wait .....wait for child process to stop or terminate
wait3 .....wait for child process to stop or terminate
write .....write on a file
writev .....see write(2)

## NAME

intro – introduction to system calls and error numbers

## SYNOPSIS

```
#include <errno.h>
```

## DESCRIPTION

This section describes all of the A/UX system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value. This is almost always −1; the individual descriptions specify the details. An error number is also made available in the external variable errno. errno is not cleared on successful calls, so it should be tested only after an error has been indicated.

There is a table of messages associated with each error, and a routine for printing the message; see perror(3C). Each system call description attempts to list all possible error numbers.

## ERRORS

The following is a complete list of A/UX error numbers and their names as defined in <errno.h>.

1 EPERM Not owner
: Typically this error indicates an attempt to modify a file in some way forbidden except to its owner or the superuser. It is also returned for attempts by ordinary users to do things allowed only by the superuser.

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 pathname does not exist.

3 ESRCH No such process
: No process can be found corresponding to that specified by *pid* in kill or ptrace.

4 EINTR Interrupted system call
: An asynchronous signal (such as interrupt or quit), which the user program has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

5 EIO I/O error
: Some physical I/O error has occurred. 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 which does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not on-line or no disk pack is loaded on a drive.

7 E2BIG Argument list too long
An argument list longer than ARG_MAX is presented to a member of the `exec` family.

8 ENOEXEC `exec` format error
A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number (see `a.out`(4)).

9 EBADF Bad file number
Either a file descriptor does not refer to an open file, or a read (respectively, write) request is made to a file that is open only for writing (respectively, reading).

10 ECHILD No children
A `wait` was executed by a process that had no existing or unwaited-for child processes.

11 EAGAIN No more processes
The system is out of a resource which may be available later. A `fork` failed because the system's process table is full or the user is not allowed to create any more processes. A system call which requires memory may also fail with this error if the system is out of memory or swap space, but the request is less than the system-imposed per process limit (see `ulimit`(2)).

12 ENOMEM Not enough space
During an `exec`, `brk`, or `sbrk`, a program asks for more space than the system is able to supply. This is not a temporary condition; the maximum space size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers, or if there is not enough swap space during a `fork`.

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 use an argument of a system call.

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

16 EBUSY Mount device busy
The device or resource is currently unavailable. An attempt was made to mount a device that was already mounted or to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment). It will also occur if an attempt is made to enable accounting when it is already enabled.

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

18 EXDEV Cross-device link
A 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., read a write-only device.

20 ENOTDIR Not a directory
A nondirectory was specified where a directory is required, for example in a path prefix or as an argument to `chdir`(2).

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

22 EINVAL Invalid argument
Some invalid argument (e.g., dismounting a nonmounted device; mentioning an undefined signal in `signal`, or `kill`; reading or writing a file for which `lseek` has generated a negative pointer). Also set by the math functions described in the (3M) entries of this manual.

23 ENFILE File table overflow
The system file table is full, and temporarily no more `opens` can be accepted.

24 EMFILE Too many open files
No process may have more than the maximum number of file descriptors OPEN_MAX open at a time. When a record lock is being created with `fcntl`, there are too many files with record locks on them.

25 ENOTTY Not a typewriter
An attempt was made to `ioctl`(2) a file that is not a special

character device.

26 ETXTBSY Text file busy
An attempt was made to execute a pure-procedure program which is currently open for writing. Also an attempt to open for writing a pure-procedure program that is being executed.

*Note:* If you are running an NFS system and you are accessing a shared binary remotely, it is possible that you will not get this `errno`.

27 EFBIG File too large
The size of a file exceeded the maximum file size ULIMIT; see `ulimit`(2).

28 ENOSPC No space left on device
During a `write` to an ordinary file, there is no free space left on the device. In `fcntl`, the setting or removing of record locks on a file cannot be accomplished because there are no more record entries left on the system

29 ESPIPE Illegal seek
An `lseek` was issued to a pipe. This error should also be issued for other nonseekable 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 was made to create more than the maximum number of links LINK_MAX to a file.

32 EPIPE Broken pipe
A write was attempted on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

33 EDOM Argument out of domain of func
The argument of a function in the math package (3M) is out of the domain of the function.

34 ERANGE Math result not representable
The value of a function in the math package (3M) is not representable within machine precision.

35 ENOMSG No message of desired type
An attempt was made to receive a message of a type that does not exist on the specified message queue; see

msgop(2).

36 EIDRM Identifier removed
This error is returned to processes that resume execution due to the removal of an identifier from the file system's name space (see msgctl(2), semctl(2), and shmctl(2)).

37 ECHRNG Channel number out of range
This errno is included for compatibility with AT&T.

38 EL2NSYNC Level 2 not synchronized
This errno is included for compatibility with AT&T.

39 EL3HLT Level 3 halted
This errno is included for compatibility with AT&T.

40 EL3RST Level 3 reset
This errno is included for compatibility with AT&T.

41 ELNRNG Link number out of range
This errno is included for compatibility with AT&T.

42 EUNATCH Protocol driver not attached
This errno is included for compatibility with AT&T.

43 ENOCSI No CSI structure available
This errno is included for compatibility with AT&T.

44 EL2HLT Level 2 halted
This errno is included for compatibility with AT&T.

45 EDEADLK Deadlock
A deadlock situation was detected and avoided.

55 EWOULDBLOCK Operation would block
An operation which would cause a process to block was attempted on an object in nonblocking mode (see socket(2N) and setcompat(2)).

56 EINPROGRESS Operation now in progress
An operation which takes a long time to complete (such as a connect(2N)) was started on a nonblocking object (see socket(2N)).

57 EALREADY Operation already in progress
An operation was attempted on a nonblocking object which already had an operation in progress.

58 ENOTSOCK Socket operation on nonsocket
A socket operation was attentped on an object that is not a socket.

59 EDESTADDRREQ Destination address required
A required address was omitted from an operation on a socket.

60 EMSGSIZE Message too long
A message sent on a socket was larger than the internal message buffer.

61 EPROTOTYPE Protocol wrong type for socket
A protocol was specified which does not support the semantics of the socket type requested. For example, you cannot use the internet UDP protocol with type SOCK_STREAM.

62 ENOPROTOOPT Bad protocol option
A bad option was specified in a getsockopt(2) or setsockopt(2) system call.

63 EPROTONOSUPPORT Protocol not supported
The protocol has not been configured into the system or there is no implementation for it.

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

65 EOPNOTSUPP Operation not supported on socket
The support for the operation on the selected socket type has not been configured or there is no implementation for it. For example, trying to accept a connection on a datagram socket.

66 EPFNOSUPPORT Protocol family not supported
The protocol family has not been configured into the system or there is no implementation for it.

67 EAFNOSUPPORT Address not supported by protocol family
An address incompatible with the requested protocol was used. For example, PUP Internet addresses cannot necessarily be used with ARPA Internet protocols.

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

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

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

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

72 ENETRESET Network dropped connection on reset
The connected host crashed and rebooted.

73 ECONNABORTED Software caused connection abort
A connection abort was caused internal to the host machine.

74 ECONNRESET Connection reset by peer
A connection was forcibly closed by a peer. This normally results from the peer executing a shutdown(2) system call.

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

76 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 other than the connected party.

77 ENOTCONN Socket is not connected
A request to send or receive data was disallowed because the socket had already been shut down with a previous shutdown(2) call.

78 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.

80 ETIMEDOUT Connection timed out
A connect request failed because the connected party did not properly respond after a period of time. (The timeout period is dependent on the communication protocol.)

81 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 which is inactive on the foreign host.

82 ELOOP Too many levels of symbolic links
A pathname lookup involved more than 8 symbolic links.

83 ENAMETOOLONG File name too long
A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters.

84 EHOSTDOWN Host is down
A socket operation encountered a defunct host.

85 EHOSTUNREACH No route to host
A socket operation was attempted to an unreachable host.

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

87 ENOSTR Device not a stream
A stream operation was attempted on a file descriptor that is not a streams device.

88 ENODATA No data (for no delay I/O)
Reading from a stream and the O_NEDELAY flag set (from open(2) or fcntl(2)) but no data is ready to be read.

89 ETIME Stream ioctl timeout
The timer set for a streams ioctl(2) system call has expired. The cause of this error is device specific and could indicate either a hardware or software failure, or perhaps a timeout value that is too short for the specific operation. The status of the ioctl(2) operation is indeterminate.

90 ENOSR Out of stream resources
During a streams open(2), either no streams queues or no streams head data structures were available.

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

96 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.

97 EPROCLIM Too many processes

98 EUSERS Too many users
A write to an ordinary file, the creation of a directory or symbolic link, or the creation of a directory entry failed abecause 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.

100 EDEADLOCK Locking deadlock error
Returned by locking(2) system call if deadlock would occur or when locktable overflows.

## DEFINITIONS

### System Constants

The following are the default implementation-specific constants defined for the A/UX system on the Macintosh II:

| | |
|---|---|
| ARG_MAX | Maximum length of argument to `exec` (5,120). |
| CHAR_BIT | Number of bits in a `char` (8). |
| CHAR_MAX | Maximum integer value of a `char` (255). |
| CHILD_MAX | Maximum number of processes per user ID (25). |
| INT_MAX | Maximum decimal value of an `int` (2,147,483,647). |
| LINK_MAX | Maximum number of links to a single file (1000) |
| LONG_MAX | Maximum decimal value of a `long` (2,147,483,647). |
| MAXDOUBLE | Maximum decimal value of a `double` (1.79769313486231470e+308). |
| NAME_MAX | Maximum number of characters in a filename (255). On System V file systems, names are limited to 14 characters. |
| OPEN_MAX | Maximum number of files a process can have open (32). |
| PATH_MAX | Maximum number of characters in a path-name (1,024). |
| PID_MAX | Maximum value for a process ID (30,001). |
| PIPE_MAX | Maximum number of bytes written to a `pipe` in a `write` (5,120). |
| PROC_MAX | Maximum number of simultaneous processes, system wide (50). |
| SHRT_MAX | Maximum decimal value of a `short` (65,535). |
| SYS_NMLN | Number of characters in a string returned by `uname` (9). |
| UID_MAX | Maximum value for a user ID or group ID (60,001). |

| | |
|---|---|
| USI_MAX | Maximum decimal value of an `unsigned` (4,294,967,295). |
| INT_MIN | Minimum decimal value for an `int` (–2,147,483,648). |
| LONG_MIN | Minimum decimal value for a `long` (–2,147,483,648). |
| SHRT_MIN | Minimum decimal value for a `short` (–32,768). |
| ULIMIT | Maximum number of bytes in a file (16,777,216). |

**Process ID**

Each active process in the system is identified uniquely by a positive integer called a process ID. The range of this ID is from 1 to PID_MAX.

**Parent Process ID**

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

**Process Group**

Each active process is a member of a process group that is identified by a positive integer called the process group ID. This ID is the process ID of the group leader. This grouping permits the signaling of related processes; see `kill`(2).

**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 terminate a group of related processes upon termination of one of the processes in the group; see `exit`(2) and `signal`(3).

**Real User ID and Real Group ID**

Each user allowed on the system is identified by a positive integer called a real user IDs.

Each user is also a member of a group. The group is identified by a positive integer called the real group ID.

An active process has a real user ID and real group ID that are set to the real user ID and real group ID, respectively, of the user responsible for the creation of the process.

**Effective User ID and Effective Group ID**

An active process has an effective user ID and an effective group ID that are used to determine file access permissions (see below). The effective user ID and effective group ID are equal to the process's real user ID and real group ID respectively, unless the process or one of its ancestors evolved from a file that had the set-user-ID bit or set-group ID bit set; see `exec`(2).

**Superuser**

A process is recognized as a "superuser" process and is granted special privileges if its effective user ID is 0.

**Special Processes**

The processes with a process ID of 0 and a process ID of 1 are special processes and are referred to as *proc0* and *proc1*.

*proc0* is the scheduler. *proc1* is the initialization process (`init`). *proc1* is the ancestor of every other process in the system and is used to control the process structure.

**File Descriptor**

A file descriptor is a small integer used to do I/O on a file. The value of a file descriptor is from 0 to OPEN_MAX-1. A process may have no more than OPEN_MAX file descriptors open simultaneously. A file descriptor is returned by system calls such as `open`(2), or `pipe`(2). The file descriptor is used as an argument by calls such as `read`(2), `write`(2), `ioctl`(2), and `close`(2).

**File Pointer**

A file with the associated `stdio` buffering is called a **stream**. A stream is a pointer to a type `FILE` defined by the `<stdio.h>` header file. The `fopen`(3S) routine creates descriptive data for a stream and returns a pointer that identifies the stream in all further transactions with other `stdio` routines.

Most `stdio` routines manipulate either a stream created by the `fopen`(3S) function or one of the three streams that are associated with three files that are expected to be open in the base system (see `termio`(7). These three streams are declared in the `<stdio.h>` header file:

| | |
|---|---|
| `stdin` | the standard input file. |
| `stdout` | the standard output file. |
| `stderr` | the standard error file. |

Output streams, with the exception of the standard error stream `stderr`, are by default buffered if the output refers to a file and

line-buffered if the output refers to a terminal. The standard error output stream `stderr` is by default unbuffered. When an output stream is unbuffered, information is queued for writing on the destination file or terminal as soon as written; when it is buffered, many characters are saved up and written as a block. When it is line-buffered, each line of output is queued for writing on the destination terminal as soon as the line is completed (that is, as soon as a newline character is written or terminal input is requested). The `setbuf`(3S) routines may be used to change the stream's buffering strategy.

**Filename**

Names consisting of 1 to 14 characters may be used to name an ordinary file, special file or directory.

These characters may be selected from the set of all character values excluding \0 (null) and the ASCII code for / (slash).

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. See `sh`(1). Although permitted, it is advisable to avoid the use of unprintable characters in file names.

**Pathname and Path Prefix**

A pathname 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.

Unless specifically stated otherwise, the null pathname is treated as if it named a nonexistent file.

More precisely, a pathname is a null-terminated character string constructed as follows:

*<path-name>*::=*<file>*|*<path-prefix><file>*|/

*<path-prefix>*::=*<rtprefix>*|/*<rtprefix>*

*<rtprefix>*::=*<dirname>*/|*<rtprefix><dirname>*/

where *<file>* is a string of 1 to 14 characters other than the ASCII slash and null, and *<dirname>* is a string of 1 to 14 characters (other than the ASCII slash and null) that names a directory.

If a pathname 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.

**Directory**

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 root directory and a current working directory for the purpose of resolving pathname searches. The root directory of a process need not be the root directory of the root file system.

**File Access Permissions**

Read, write, and execute/search permissions on a file are granted to a process if one or more of the following is true:

The effective user ID of the process is superuser.

The effective user ID of the process matches the user ID of the owner of the file and the appropriate access bit of the "owner" portion (0700) of the file mode is set.

The effective user ID of the process does not match the user ID of the owner of the file, and the effective group ID of the process matches the group of the file and the appropriate access bit of the "group" portion (070) of the file mode is set.

The effective user ID of the process does not match the user ID of the owner of the file, and the effective group ID of the process does not match the group ID of the file, and the appropriate access bit of the "other" portion (07) of the file mode is set.

Otherwise, the corresponding permissions are denied.

INTERPROCESS COMMUNICATION

**Message Queue Identifier**

A message queue identifier (*msqid*) is a unique positive integer created by a `msgget(2)` system call. Each *msqid* has a message queue and a data structure associated with it. The data structure is referred to as `msqid_ds` and contains the following members:

```
struct ipc_perm msg_perm; /* operation permission
                             struct */
ushort msg_qnum;          /* number of msgs on q */
ushort msg_qbytes;        /* max number of bytes on q */
```

```
ushort msg_lspid;          /* pid of last msgsnd
                              operation */
ushort msg_lrpid;          /* pid of last msgrcv
                              operation */
time_t msg_stime;          /* last msgsnd time */
time_t msg_rtime;          /* last msgrcv time */
time_t msg_ctime;          /* last change time */
                           /* Times measured in secs
                              since 00:00:00 GMT, 1/1/70 */
```

msg_perm is an ipc_perm structure that specifies the message operation permission (see below). This structure includes the following members:

```
ushort  cuid;   /* creator user ID */
ushort  cgid;   /* creator group ID */
ushort  uid;    /* user ID */
ushort  gid;    /* group ID */
ushort  mode;   /* r/w permission */
```

msg_qnum is the number of messages currently on the queue. msg_qbytes is the maximum number of bytes allowed on the queue. msg_lspid is the process ID of the last process that performed a msgsnd operation. msg_lrpid is the process id of the last process that performed a msgrcv operation. msg_stime is the time of the last msgsnd operation, msg_rtime is the time of the last msgrcv operation, and msg_ctime is the time of the last msgctl(2) operation that changed a member of the above structure.

**Semaphore Identifier**

A semaphore identifier (*semid*) is a unique positive integer created by a semget(2) system call. Each *semid* has a set of semaphores and a data structure associated with it. The data structure is referred to as semid_ds and contains the following members:

```
struct ipc_perm sem_perm; /* operation permission
                             struct */
ushort sem_nsems;         /* number of sems in set */
time_t sem_otime;         /* last operation time */
time_t sem_ctime;         /* last change time */
                          /* Times measured in secs since
                             00:00:00 GMT, 1/1/970 */
```

sem_perm is an ipc_perm structure that specifies the semaphore operation permission (see below). This structure includes

the following members:

```
ushort  cuid;   /* creator user ID */
ushort  cgid;   /* creator group ID */
ushort  uid;    /* user ID */
ushort  gid;    /* group ID */
ushort  mode;   /* r/a permission */
```

The value of sem_nsems is equal to the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as a sem_num. sem_num values run sequentially from 0 to the value of sem_nsems minus 1. sem_otime is the time of the last semop(2) operation, and sem_ctime is the time of the last semctl(2) operation that changed a member of the above structure.

A semaphore is a data structure that contains the following members:

```
ushort semval;    /* semaphore value */
short  sempid;    /* pid of last operation  */
ushort semncnt;   /* # awaiting semval > cval */
ushort semzcnt;   /* # awaiting semval = 0 */
```

semval is a non-negative integer. sempid is equal to the process ID of the last process that performed a semaphore operation on this semaphore. semncnt is a count of the number of processes that are currently suspended awaiting this semaphore's semval to become greater than its current value. semzcnt is a count of the number of processes that are currently suspended awaiting this semaphore's semval to become zero.

**Shared Memory Identifier**

A shared memory identifier (*shmid*) is a unique positive integer created by a shmget(2) system call. Each *shmid* has a segment of memory (referred to as a shared memory segment) and a data structure associated with it. The data structure referred to as shmid_ds contains the following members:

```
struct  ipc_perm shm_perm; /* operation permission struct*/
int     shm_segsz;         /* size of segment*/
ushort  shm_cpid;          /* creator pid*/
ushort  shm_lpid;          /* pid of last operation*/
short   shm_nattch;        /* number of current attaches*/
time_t  shm_atime;         /* last attach time*/
```

```
time_t  shm_dtime;          /* last detach time*/
time_t  shm_ctime;          /* last change time*/
                            /* Times measured in secs
                               since 00:00:00 GMT, 1/1/70*/
```

shm_perm is an ipc_perm structure that specifies the shared memory operation permission (see below). This structure includes the following members:

```
ushort  cuid;   /* creator user ID */
ushort  cgid;   /* creator group ID */
ushort  uid;    /* user ID */
ushort  gid;    /* group ID */
ushort  mode;   /* r/w permission */
```

shm_segsz specifies the size of the shared memory segment. shm_cpid is the process ID of the process that created the shared memory identifier. shm_lpid is the process ID of the last process that performed a shmop(2) operation. shm_nattch is the number of processes that currently have this segment attached. shm_atime is the time of the last shmat operation, shm_dtime is the time of the last shmdt operation, and shm_ctime is the time of the last shmctl(2) operation that changed one of the members of the above structure.

**IPC PERMISSIONS**

In the msgop(2) and msgctl(2) system call descriptions, the permission required for an operation is interpreted as follows:

00400 Read by user
00200 Write by user
00060 Read, Write by group
00006 Read, Write by others

**Message Operation Permissions**

Read and Write permissions on a *msqid* are granted to a process if one or more of the following is true:

The effective user ID of the process is superuser.

The effective user ID of the process matches msg_perm.[c]uid in the data structure associated with *msqid* and the appropriate bit of the "user" portion (0600) of msg_perm.mode is set.

The effective user ID of the process does not match msg_perm.[c]uid and the process's effective group ID matches msg_perm.[c]gid and the appropriate bit of the

"group" portion (060) of `msg_perm.mode` is set.

The effective user ID of the process does not match `msg_perm.[c]uid` and the effective group ID of the process does not match `msg_perm.[c]gid` and the appropriate bit of the "other" portion (06) of `msg_perm.mode` is set.

Otherwise, the corresponding permissions are denied.

**Semaphore Operation Permissions**

Read and Alter permissions on a *semid* are granted to a process if one or more of the following is true:

The effective user ID of the process is superuser.

The effective user ID of the process matches `sem_perm.[c]uid` in the data structure associated with *semid* and the appropriate bit of the "user" portion (0600) of `sem_perm.mode` is set.

The effective user ID of the process does not match `sem_perm.[c]uid` and the effective group ID of the process matches `sem_perm.[c]gid` and the appropriate bit of the "group" portion (060) of `sem_perm.mode` is set.

The effective user ID of the process does not match `sem_perm.[c]uid` and the effective group ID of the process does not match `sem_perm.[c]gid` and the appropriate bit of the "other" portion (06) of `sem_perm.mode` is set.

Otherwise, the corresponding permissions are denied.

**Shared Memory Operation Permissions**

Read and Write permissions on a *shmid* are granted to a process if one or more of the following is true:

The effective user ID of the process is superuser.

The effective user ID of the process matches `shm_perm.[c]uid` in the data structure associated with `shmid` and the appropriate bit of the "user" portion (0600) of `shm_perm.mode` is set.

The effective user ID of the process does not match `shm_perm.[c]uid` and the effective group ID of the process matches `shm_perm.[c]gid` and the appropriate bit of

the "group" portion (060) of `shm_perm.mode` is set.

The effective user ID of the process does not match `shm_perm.[c]uid` and the effective group ID of the process does not match `shm_perm.[c]gid` and the appropriate bit of the "other" portion (06) of `shm_perm.mode` is set.

Otherwise, the corresponding permissions are denied.

**SEE ALSO**

`close`(2), `ioctl`(2), `open`(2), `pipe`(2), `read`(2), `write`(2), `intro`(3), `perror`(3).
"Overview of the Programming Environment" in *A/UX Programming Languages and Tools, Volume 1*.

## NAME

accept – accept a connection on a socket

## SYNOPSIS

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

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

## DESCRIPTION

The argument *s* is a socket which has been created with socket(2N), bound to an address with bind(2N), and is listening for connections after a listen(2N). 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 for the socket. If no pending connections are present on the queue, and the socket is not marked as nonblocking, accept blocks the caller until a connection is present. If the socket is marked nonblocking and no pending connections are present on the queue, accept returns an error as described below. The accepted socket may not be used to accept more connections. The original socket *s* remains open.

The argument *addr* is a result parameter which 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(2N) 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 which is a descriptor for the accepted socket.

## ERRORS

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 nonblocking and no connections are present to be accepted. |

**SEE ALSO**

bind(2N), connect(2N), listen(2N), select(2N), socket(2N).

**NAME**

access – determine accessibility of a file

**SYNOPSIS**

```
int access(path, amode)
char *path;
int amode;
```

**DESCRIPTION**

access is used to determine the accessibility of a file. *path* points to a path name naming a file. access checks the named file for accessibility according to the bit pattern contained in *amode*, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID. The bit pattern contained in *amode* is constructed as follows:

| | |
|---|---|
| 04 | read |
| 02 | write |
| 01 | execute (search) |
| 00 | check existence of file |

**RETURN VALUE**

If the requested access is permitted, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

**ERRORS**

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

| | |
|---|---|
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOTDIR] | A component of the path prefix is not a directory. |
| [ENOENT] | Read, write, or execute (search) permission is requested for a null path name. |
| [ENOENT] | The named file does not exist. |
| [EACCES] | Search permission is denied on a component of the path prefix. |

| | |
|---|---|
| [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. |
| | *Note:* If you are running an NFS system and you are accessing a shared binary remotely, it is possible that you will not get this errno. |
| [EACCESS] | Permission bits of the file mode do not permit the requested access. |
| [EFAULT] | *path* points outside the allocated address space for the process. |

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 permissions checked with respect to the "group" mode bits, and all others have permissions checked with respect to the "other" mode bits.

The superuser is always granted execute permission even though (1) execute permission is meaningful only for directories and regular files, and (2) exec requires that at least one execute mode bit be set for regular file to be executable.

Notice that it is only access bits that are checked. A directory may be announced as writable by access, but an attempt to open it for writing will fail because it is not allowed to write into the directory structure itself, although files may be created there. A file may look executable, but exec will fail unless it is in proper format.

**SEE ALSO**

chmod(2), stat(2).

## NAME

acct – enable or disable process accounting

## SYNOPSIS

```
int  acct(path)
char  *path;
```

## DESCRIPTION

acct is used to enable or disable the system process accounting routine. If the routine is enabled, an accounting record will be written on an accounting file for each process that terminates. Termination can be caused by one of two things: an exit call or a signal; see exit(2) and signal(3). The effective user ID of the calling process must be superuser to use this call.

*path* points to a path name naming the accounting file. The accounting file format is given in acct(4).

The accounting routine is enabled if *path* is nonzero and no errors occur during the system call. It is disabled if *path* is zero and no errors occur during the system call.

## 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

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

| | |
|---|---|
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [EPERM] | The effective user ID of the calling process is not superuser. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [EBUSY] | An attempt is being made to enable accounting when it is already enabled. |
| [ENOTDIR] | A component of the path prefix is not a directory. |
| [ENOENT] | One or more components of the accounting file path name do not exist. |

| | |
|---|---|
| [EACCES] | A component of the path prefix denies search permission. |
| [EACCES] | The file named by *path* is not an ordinary file. |
| [EACCES] | mode permission is denied for the named accounting file. |
| [EROFS] | The named file resides on a read-only file system. |
| [EFAULT] | *path* points to an illegal address. |

**SEE ALSO**

acct(1M), exit(2), signal(3), acct(4).

## 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 nonzero, 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 superuser.

## 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

adjtime will fail if:

| | |
|---|---|
| [EFAULT] | An argument points outside the process's allocated address space. |
| [EPERM] | The process's effective user ID is not that of the superuser. |

## SEE ALSO

date(1).

## NAME

alarm – set a process's alarm clock

## SYNOPSIS

```
unsigned alarm(sec)
unsigned sec;
```

## DESCRIPTION

alarm instructs the calling process's alarm clock to send the signal SIGALRM to the calling process after the number of real time seconds specified by *sec* have elapsed; see signal(3).

alarm requests are not stacked; successive calls reset the calling process's alarm clock. If the argument is 0, any alarm request is canceled. Because the clock has a 1-second resolution, the signal may occur up to one second early; 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 4,294,967,295 (2**32-1) seconds, or 136 years.

## RETURN VALUE

alarm returns the amount of time previously remaining in the calling process's alarm clock.

## SEE ALSO

pause(2), setitimer(2), signal(3).

## NAME

bind – bind a name to a socket

## SYNOPSIS

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

int 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(2N) it exists in a name space (address family) but has no name assigned. bind requests that the *name* be assigned to the socket.

## NOTES

The rules used in name binding vary between communication domains. Consult the manual entries in Section 5 (specifically inet(5F)) 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

bind 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. |
| [EACCESS] | 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. |

**SEE ALSO**

connect(2N), getsockname(2N), listen(2N), socket(2N).

## NAME

brk, sbrk – change data segment space allocation

## SYNOPSIS

```
int  brk(endds)
char  *endds;

char  *sbrk(incr)
int  incr;
```

## DESCRIPTION

brk and sbrk are used to change dynamically the amount of space allocated for the calling process's data segment; see exec(2). The change is made by resetting the process's break value and allocating the appropriate amount of space. The break value is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases. The newly allocated space is set to zero.

brk sets the break value to *endds* and changes the allocated space accordingly.

sbrk adds *incr* bytes to the break value and changes the allocated space accordingly. *incr* can be negative, in which case the amount of allocated space is decreased.

## RETURN VALUE

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

## ERRORS

brk and sbrk will fail without making any change in the allocated space if the following is true:

| | |
|---|---|
| [ENOMEM] | Not enough space. Program asks for more space than the system is able to supply. |
| [EAGAIN] | The system has temporarily exhausted its available memory or swap space. |

Such a change would result in more space being allocated than is allowed by a system-imposed maximum (see ulimit(2)). Such a change would result in the break value being greater than or equal to the start address of any attached shared memory segment (see shmop(2)).

## SEE ALSO

exec(2), shmop(2), ulimit(2).

**NAME**

chdir – change working directory

**SYNOPSIS**

```
int  chdir(path)
char  *path;
```

**DESCRIPTION**

chdir causes the named directory to become the current working directory, the starting point for path searches for path names not beginning with /. *path* points to the path name of a 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:

| | |
|---|---|
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOTDIR] | A component of the path name is not a directory. |
| [ENOENT] | The named directory does not exist. |
| [EACCES] | Search permission is denied for any component of the path name. |
| [EFAULT] | *path* points outside the allocated address space of the process. |

**SEE ALSO**

csh(1), ksh(1), sh(1), chroot(2).

**NAME**

chmod – change mode of file

**SYNOPSIS**

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

**DESCRIPTION**

chmod sets the access permission portion of the named file's mode according to the bit pattern contained in *mode*. *path* points to a path name naming a file.

Access permission bits are interpreted as follows:

| | |
|---|---|
| 04000 | Set effective user ID on execution. |
| 02000 | Set effective group ID on execution. |
| 01000 | Save text image after execution. |
| 00400 | Read by owner. |
| 00200 | Write by owner. |
| 00100 | Execute (search if a directory) by owner. |
| 00070 | Read, write, execute (search) by group. |
| 00007 | Read, write, execute (search) by others. |

The effective user ID of the calling process must match the owner of the file or be the superuser to change the mode of a file.

If the effective user ID of the process is not the superuser, mode bit 01000 (save text image on execution) is cleared.

If the effective user ID of the process is not superuser and the effective group ID of the process does not match the group ID of the file, mode bit 02000 (set group ID on execution) is cleared.

If an executable file is prepared for sharing (see the cc −n option), then mode bit 01000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus, when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time.

Changing the owner of a file turns off the set user ID bit, unless the superuser does it. This makes the system somewhat more secure 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 one or more of the following are true:

| | |
|---|---|
| `[ENOTDIR]` | A component of the path prefix is not a directory. |
| `[ENAMETOOLONG]` | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| `[ELOOP]` | Too many symbolic links were encountered in translating a pathname. |
| `[ENOENT]` | The named file does not exist. |
| `[EACCES]` | Search permission is denied on a component of the path prefix. |
| `[EPERM]` | A pathname contains a character with the high-order bit set. |
| `[EPERM]` | The effective user ID does not match the owner of the file and the effective user ID is not superuser. |
| `[EROFS]` | The named file resides on a read-only file system. |
| `[EFAULT]` | *path* points outside the allocated address space of the process. |

**SEE ALSO**

`chmod`(1), `chown`(2), `mknod`(2), `open`(2), `stat`(2), `mknod`(2), `umask`(2).

## NAME

chown, fchown – change owner and group of a file

## SYNOPSIS

```
int chown(path, owner, group)
char *path;
int owner, group;

int fchown(fd, owner, group)
int fd, owner, group;
```

## DESCRIPTION

The file which is named by *path* or referenced by *fd* has its *owner* and *group* changed as specified. Only the superuser or the file's owner may execute this call.

chown clears the set userID and set group ID bits on the file to prevent accidental creation of set user ID and set group ID programs owned by the superuser.

If chown is invoked successfully by other than the superuser, the set user ID and set group ID bits of the file mode, 04000 and 02000 respectively, will be cleared. (This prevents ordinary users from making themselves effectively other users or members of a group to which they don't belong.)

Only one of the owner and group ID's may be set by specifying the other as –1.

## 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:

| | |
|---|---|
| [EINVAL] | The argument path does not refer to a file. |
| [ENOTDIR] | A component of the path prefix is not a directory. |
| [ENOENT] | The argument pathname is too long. |
| [EPERM] | The argument contains a byte with the high-order bit set. |
| [ENOENT] | The named file does not exist. |

| | |
|---|---|
| [EACCES] | Search permission is denied on a component of the path prefix. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [EPERM] | The effective user ID does not match the owner of the file and the effective user ID is not the superuser. |
| [EROFS] | The named file resides on a read-only file system. |
| [EFAULT] | *path* points outside the process's allocated address space. |
| [ELOOP] | Too many symbolic links were encountered in translating the pathname. |

`fchown` will fail if:

| | |
|---|---|
| [EBADF] | *fd* does not refer to a valid descriptor. |
| [EINVAL] | *fd* refers to a socket, not a file. |

**SEE ALSO**

`chown`(1), `chgrp`(2), `chmod`(2).

**NAME**

chroot – change root directory

**SYNOPSIS**

```
int  chroot (path)
char  *path;
```

**DESCRIPTION**

chroot causes the named directory to become the root directory, the starting point for path searches for path names beginning with /. The user's working directory is unaffected by the chroot system call. *path* points to a path name naming a directory.

The effective user ID of the process must be the superuser to change the root directory.

The .. entry in the root directory is interpreted to mean the root directory itself. Thus, .. cannot be used to access files outside the subtree rooted at the root 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**

chroot will fail and the root directory will remain unchanged if one or more of the following are true:

| | |
|---|---|
| [ENOTDIR] | Any component of the path name is not a directory. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOENT] | The named directory does not exist. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [EPERM] | The effective user ID is not the superuser. |
| [EFAULT] | *path* points outside the allocated address space of the process. |

**SEE ALSO**

chroot(1M), chdir(2).

**NAME**

close – close a file descriptor

**SYNOPSIS**

```
int close(fildes)
int fildes;
```

**DESCRIPTION**

close closes the file descriptor indicated by *fildes*. All outstanding record locks owned by the process (on the file indicated by *fildes*) are removed.

*fildes* is a file descriptor obtained from a creat, open, dup, fcntl, pipe, or socket system call. A close of all files is automatic on exit, but since there is a small, finite limit on the number of open files per process, OPEN_MAX, close is necessary for programs which deal with many files.

**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**

close will fail if:

[EBADF] *fildes* is not a valid open file descriptor.

**SEE ALSO**

creat(2), dup(2), exec(2), fcntl(2), open(2), pipe(2), socket(2N).

**NAME**

connect – initiate a connection on a socket

**SYNOPSIS**

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

int connect(s, name, namelen)
int s;
struct sockaddr *name;
int namelen;
```

**DESCRIPTION**

connect is used to initiate a connection on a socket. The parameter *s* is a socket. If it is of type SOCK_DGRAM, then this call permanently specifies the peer to which datagrams are to be sent; if it 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.

**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**

connect 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. |
| [EWOULDBLOCK] | The socket is nonblocking and the and the connection cannot be completed immediately. It is possible to select(2N) the socket while it is connecting by selecting it for writing. |

**SEE ALSO**

accept(2N), getsockname(2N), select(2N), socket(2N).

## NAME

creat – create a new file or rewrite an existing one

## SYNOPSIS

```
int creat(path, mode)
char *path;
int mode;
```

## DESCRIPTION

creat creates a new ordinary file or prepares to rewrite an existing file named by the path name pointed to by *path*.

If the file exists, the length is truncated to 0 and the mode and owner are unchanged. Otherwise, the file's owner ID is set to the effective user ID, of the process the group ID of the process is set to the effective group ID, of the process and the low-order 12 bits of the file mode are set to the value of *mode* modified as follows:

All bits set in the process's file mode creation mask are cleared. See umask(2).

The "save text image after execution bit" of the mode is cleared. See chmod(2).

Upon successful completion, the file descriptor is returned and the file is open for writing, even if the mode does not permit writing. The file pointer is set to the beginning of the file. The file descriptor is set to remain open across exec system calls. See fcntl(2). No process may have more than the maximum number of files, OPEN_MAX, open simultaneously.

The *mode* given is arbitrary; it need not allow writing. This feature is used by programs which deal with temporary files of fixed names. The creation is done with a mode that forbids writing. Then, if a second instance of the program attempts a creat, an error is returned and the program knows that the name is unusable for the moment.

## RETURN VALUE

Upon successful completion, a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of –1 is returned and errno is set to indicate the error.

## ERRORS

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

| | |
|---|---|
| [ENOTDIR] | A component of the path prefix is not a directory. |

| | |
|---|---|
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOENT] | A component of the path prefix does not exist. |
| [EACCES] | Search permission is denied on a component of the path prefix. |
| [ENOENT] | The path name is null. |
| [EACCES] | The file does not exist and the directory in which the file is to be created does not permit writing. |
| [EROFS] | The named file resides or would reside on a read-only file system. |
| [ETXTBSY] | The file is a pure procedure (shared text) file that is being executed. |
| | *Note:* If you are running an NFS system and you are accessing a shared binary remotely, it is possible that you will not get this `errno`. |
| [EACCES] | The file exists and write permission is denied. |
| [EISDIR] | The named file is an existing directory. |
| [EMFILE] | the maximum number of file descriptors are currently open. |
| [EFAULT] | *path* points outside the allocated address space of the process. |
| [ENFILE] | The system file table is full. |

**BUGS**

The system-scheduling algorithm does not make this a true uninterruptible operation, and a race condition may develop if `creat` is done at precisely the same time by two different processes.

**SEE ALSO**

chmod(2), close(2), dup(2), fcntl(2), lseek(2), open(2), read(2), umask(2), write(2).

**NAME**

dup – duplicate a descriptor

**SYNOPSIS**

```
int  dup(oldd)
int  oldd;
```

**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(2N). The new descriptor returned by the call is the lowest numbered descriptor which is not currently in use by the process.

The object referenced by the descriptor does not distinguish between references using the old and new descriptor in any way. Thus if the old and new descriptor are duplicate references to an open file, read(2), write(2), and lseek(2) calls all move a single pointer into the file. If a separate pointer into the file is desired, a different object reference to the file must be obtained by issuing an additional open(2) call.

**RETURN VALUE**

The value −1 is returned if an error occurs in either call and errno is set to indicate the error.

**ERRORS**

dup fails if:

| | |
|---|---|
| [EBADF] | The old descriptor is not a valid active descriptor |
| [EMFILE] | Too many descriptors are active. |

**SEE ALSO**

accept(2N), open(2), close(2), getdtablesize(2N), pipe(2), socket(2N), dup2(3N).

## NAME

execl, execv, execle, execve, execlp, execvp – execute a file

## SYNOPSIS

```
int execl(path, arg0, arg1,..., argn, 0);
char *path, *arg0, *arg1,..., *argn;

int execv(path, argv)
char *path, *argv[];

int execle(path, arg0, arg1,..., argn, 0, envp)
char *path, *arg0, *arg1,..., *argn, *envp[];

int execve(path, argv, envp)
char *path, *argv[], *envp[];

int execlp(file, arg0, arg1,..., argn, 0)
char *file, *arg0, *arg1,..., *argn;

int execvp(file, argv)
char *file, *argv[];
```

## DESCRIPTION

exec in all its forms transforms the calling process into a new process. The new process is constructed from an ordinary, executable file called the "new process file." There can be no return from a successful exec because the calling process is overlaid by the new process.

*path* points to a path name that identifies the new process file.

*file* points to the new process file. The path prefix for this file is obtained by a search of the directories passed as the environment variable PATH (see environ(5)).

The shell is invoked if a command file is found by execlp or execvp.

*arg0*, *arg1*, ..., *argn* are pointers to null terminated character strings. These strings constitute the argument list available to the new process. By convention, at least *arg0* must be present and point to a string that is the same as *path* (or its last component).

*argv* is an array of character pointers to null terminated strings. These strings constitute the argument list available to the new process. By convention, *argv* must have at least one member, and it must point to a string that is the same as *path* (or its last component). *argv* is terminated by a null pointer and is directly usable in another execv because *argv*[*argc*] is 0.

*envp* is an array of character pointers to null terminated strings. These strings constitute the environment for the new process. *envp* is terminated by a null pointer. For `execl` and `execv`, the C runtime start-off routine places a pointer to the environment of the calling process in the global cell:

```
extern char **environ;
```

and it is used to pass the environment of the calling process to the new process.

File descriptors open in the calling process remain open in the new process, except for those whose close-on-exec flag is set; see `fcntl`(2). For those file descriptors that remain open, the file pointer is unchanged.

The new process automatically has the System V, Release 2 signal mechanism. Signals set to terminate the calling process will be set to terminate the new process. Signals set to be ignored by the calling process will be set to be ignored by the new process. Signals set to be caught by the calling process will be set to terminate new process; see `signal`(3).

If the set user ID mode bit of the new process file is set (see `chmod`(2)), `exec` sets the effective user ID of the new process to the owner ID of the new process file. Similarly, if the set group ID mode bit of the new process file is set, the effective group ID of the new process is set to the group ID of the new process file. The real user ID and real group ID of the new process remain the same as those of the calling process.

The shared memory segments attached to the calling process will not be attached to the new process (see `shmop`(2)).

Profiling is disabled for the new process; see `profil`(2).

Regions of physical memory mapped into the virtual address space of the calling process are detached from the address space of the new process; see `phys`(2).

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

access groups (see `getgroups`(2))
nice value (see `nice`(2))
process ID
parent process ID

process group ID
`semadj` values (see `semop`(2))
tty group ID (see `exit`(2) and `signal`(3))
trace flag (see `ptrace`(2) request 0)
time left until an alarm clock signal (see `alarm`(2))
current working directory
root directory
file mode creation mask (see `umask`(2))
file size limit (see `ulimit`(2))
`utime`, `stime`, `cutime`, and `cstime` (see `times`(2))

`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.

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.

*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`(5) 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. The `exec` routines use lower-level routines as follows to pass an environment explicitly:

```
execve(file, argv, environ);
execle(file, arg0, arg1, ... , argn, 0, environ);
```

`execlp` 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.

RETURN VALUE

If exec returns to the calling process an error has occurred; the return value will be –1 and errno will be set to indicate the error.

ERRORS

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

| | |
|---|---|
| [ENOENT] | One or more components of the new process file's path name do not exist. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOTDIR] | A component of the new process file's path prefix is not a directory. |
| [EACCES] | Search permission is denied for a directory listed in the new process file's path prefix. |
| [EACCES] | The new process file is not an ordinary file. |
| [EACCES] | The new process file mode denies execution permission. |
| [EAGAIN] | The system has temporarily exhausted its available memory or swap space. |
| [ENOEXEC] | The exec is not an execlp or execvp, and the new process file has the appropriate access permission but 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 by some process. |

*Note:* If you are running an NFS system and you are accessing a shared binary remotely, it is possible that you will not get this

errno.

| | |
|---|---|
| [ENOMEM] | The new process requires more memory than is allowed by the system-imposed maximum (MAXMEM). |
| [E2BIG] | The number of bytes in the new process's argument list is greater than the system-imposed limit of ARG_MAX. |
| [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. |

**SEE ALSO**

csh(1), ksh(1), sh(1), alarm(2), exit(2), fork(2), nice(2), phys(2), ptrace(2), semop(2), setcompat(2), times(2), signal(3).

## NAME

exit, _exit – terminate process

## SYNOPSIS

```
void  exit(status)
int  status;

void  _exit(status)
int  status;
```

## DESCRIPTION

exit terminates the calling process with the following consequences:

All of the file descriptors open in the calling process are closed.

If the parent process of the calling process is executing a wait, it is notified of the calling process's termination and the low order eight bits (i.e., bits 0377) of *status* are made available to it; see wait(2).

If the parent process of the calling process is not executing a wait, the calling process is transformed into a zombie process. A "zombie process" is a process that only occupies a slot in the process table. It has no other space allocated either in user or kernel space. The process table slot that it occupies is partially overlaid with time accounting information (see <sys/proc.h>) to be used by times.

The parent process ID of all of the calling process's existing child processes and zombie processes is set to 1. This means the initialization process (see intro(2)) inherits each of these processes.

Each attached shared memory segment is detached and the value of shm_nattach in the data structure associated with its shared memory identifier is decremented by 1.

For each semaphore for which the calling process has set a semadj value (see semop(2)), that semadj value is added to the semval of the specified semaphore.

If the process has a process, text, or data lock, an unlock is performed (see plock(2)).

An accounting record is written on the accounting file if the system's accounting routine is enabled; see acct(2).

If the process ID, tty group ID, and process group ID of the calling process are equal, the SIGHUP signal is sent to each process that has a process group ID equal to that of the calling process.

The C function exit may cause cleanup actions before the process exits. The function _exit circumvents all cleanup.

**SEE ALSO**

acct(2), fork(2), intro(2), plock(2), semop(2), wait(2), signal(3).

**WARNING**

See WARNING section in signal(3).

**NAME**

fcntl – file control

**SYNOPSIS**

```
#include <fcntl.h>

int fcntl (fildes, cmd, arg)
int fildes, cmd, arg;
```

**DESCRIPTION**

fcntl provides for control over open files. *fildes* is an open file descriptor obtained from a creat, open, dup, fcntl, socket, or pipe system call.

The *cmd*s available are:

F_DUPFD — Return a new file descriptor as follows:

Lowest numbered available file descriptor greater than or equal to *arg*.

Same open file (or pipe) as the original file.

Same file pointer as the original file (i.e., both file descriptors share one file pointer).

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 exec(2) system calls.

F_GETFD — Get the close-on-exec flag associated with the file descriptor *fildes*. 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 *fildes* to the low-order bit of *arg* (0 or 1 as above).

F_GETFL — Get *file* status flags.

F_SETFL — Set *file* status flags to *arg*. Only certain flags can be set; see fcntl(5).

F_GETLK — Get the first lock which blocks the lock description given by the variable of type struct flock pointed to by *arg*. The information retrieved overwrites the information passed to

fcntl in the flock structure. If no lock is found that would prevent this lock from being created, then the structure is passed back unchanged except for the lock type which will be set to F_UNLCK.

F_SETLK — Set or clear a file segment lock according to the variable of type struct flock pointed to by *arg* (see fcntl(5)). The *cmd* F_SETLK is used to establish read (F_RDLCK) and write (F_WRLCK) locks, as well as remove either type of lock (F_UNLCK). If a read or write lock cannot be set, fcntl will return immediately with an error value of −1.

F_SETLKW — This *cmd* is the same as F_SETLK except that if a read or write lock is blocked by other locks, the process will sleep until the segment is free to be locked.

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.

A read lock prevents any process from write locking the protected area. More than one read lock may exist for a given segment of a file at a given time. The file descriptor on which a read lock is being placed must have been opened with read access.

A write lock prevents any process from read locking or write locking the protected area. Only one write lock may exist for a given segment of a file at a given time. The file descriptor on which a write lock is being placed must have been opened with write access.

The structure flock describes the type (l_type), starting offset (l_whence), relative offset (l_start), size (l_len), and process ID (l_pid) of the segment of the file to be affected. The process ID field is only used with the F_GETLK *cmd* to return the value for a block in lock. Locks may start and extend beyond the current end of a file, but may not be negative relative to the beginning of the file. A lock may be set to always extend to

the end of file by setting `l_len` to zero (0). If such a lock also has `l_start` set to zero (0), the whole file will be locked. Changing or unlocking a segment from the middle of a larger locked segment leaves two smaller segments for either end. Locking a segment that is already locked by the calling process causes the old lock type to be removed and the new lock type to take affect. All locks associated with a file for a given process are removed when a file descriptor for that file is closed by that process or the process holding that file descriptor terminates. Locks are not inherited by a child process in a `fork`(2) system call.

**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_SETFD` | Value other than −1. |
| `F_GETFL` | Value of file flags. |
| `F_SETFL` | Value other than −1. |
| `F_GETLK` | Value other that −1. |
| `F_SETLK` | Value other than −1. |
| `F_SETLKW` | Value other than −1. |
| `F_GETOWN` | Value other than −1. |
| `F_SETOWN` | Value other than −1. |

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

**ERRORS**

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

| | |
|---|---|
| `[EBADF]` | *fildes* is not a valid open file descriptor. |
| `[EMFILE]` | *cmd* is `F_DUPFD` and the maximum number of file descriptors are currently open. |
| `[EINFILE]` | *cmd* is `F_DUPFD` and *arg* is negative or greater than the maximum number file descriptors currently open. |

[EINVAL] *cmd* is F_GETLK, F_SETLK, or SETLKW and *arg* or the data it points to is not valid.

[EACCESS] *cmd* is F_SETLK the type of lock (l_type) is a read (F_RDLCK) or write (F_WRLCK) lock and the segment of a file to be locked is already write locked by another process or the type is a write lock and the segment of a file to be locked is already read or write locked by another process.

[EMFILE] *cmd* is F_SETLK or F_SETLKW, the type of lock is a read or write lock and there are no more file locking headers available (too many files have segments locked).

[ENOSPC] *cmd* is F_SETLK or F_SETLKW, the type of lock is a read or write lock and there are no more file locking headers available (too many files have segments locked) or there are no more record locks available (too many file segments locked).

[EDEADLK] *cmd* is F_SETLK, when the lock is blocked by some lock from another process and sleeping (waiting) for that lock to become free, this causes a deadlock situation.

[ENOTSOCK] *cmd* is F_GETOWN or F_SETOWN and *fildes* is not a file descriptor for a socket.

[EREMOTE] *cmd* is F_GETLK F_SETLK or F_SETLKW and *fildes* references a file on a remotely mounted file system.

**SEE ALSO**

close(2), creat(2), dup(2), exec(2), ioctl(2), open(2), pipe(2), socket(2N), lockf(3C), fcntl(5).

## 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   /* nonblocking lock */
#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, the *operation* should be LOCK_UN.

Advisory locks allow cooperating processes to perform consistent operations on files, but do not guarantee exclusive access (i.e., processes may still access files without using advisory locks, possibly resulting in inconsistencies).

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

A shared lock may be *upgraded* to an exclusive lock, and vice versa, simply by specifying the appropriate lock type; the previous lock will be 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 block 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 on success, –1 on error, with an error code stored in `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. |
| `[EOPNOTSUPP]` | The argument *fd* refers to an object other than a file. |

SEE ALSO

`close`(2), `dup`(2), `execve`(2), `fcntl`(2), `fork`(2), `open`(2), `lockf`(3).

BUGS

Locks obtained through the `flock` mechanism are known only within the system on which they were placed. Thus, multiple clients may successfully acquire exclusive locks on the same remote file. If this behavior is not explicitly desired, the `fcntl`(2) or `lockf`(3) system calls should be used instead.

## NAME

fork – create a new process

## SYNOPSIS

```
int fork()
```

## DESCRIPTION

fork causes creation of a new process. The new process (child process) is an exact copy of the calling process (parent process). The child process inherits the following attributes from the parent process:

environment
close-on-exec flag (see exec(2))
signal handling settings (i.e., SIG_DFL, SIG_IGN, function address)
set user ID mode bit
set group ID mode bit
process compatibility flags (see setcompat(2))
profiling on/off status
access groups (see getgroups(2))
nice value (see nice(2))
all attached shared memory segments (see shmop(2))
process group ID
tty group ID (see exit(2) and signal(3))
trace flag (see ptrace(2) request 0)
time left until an alarm clock signal (see alarm(2))
current working directory
root directory
file mode creation mask (see umask(2))
file size limit (see ulimit(2))
phys regions see phys(2).

The child process differs from the parent process in the following ways:

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 file descriptors. Each of the child's file descriptors shares a common file pointer with the corresponding file descriptor of the parent.

All semadj values are cleared (see semop(2)).

Process locks, text locks and data locks are not inherited by the child (see plock(2)).

The child process's utime, stime, cutime, and cstime are set to 0 (see times(2)). The time left until an alarm clock signal is reset to 0.

**RETURN VALUE**

Upon successful completion, fork returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of −1 is returned to the parent process, no child process is created, and 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. |
| [EAGAIN] | The system-imposed limit on the total number of processes under execution by a single user would be exceeded. |
| [EAGAIN] | The system has temporarily exhausted its available memory or swap space. |

**SEE ALSO**

exec(2), nice(2), phys(2), plock(2), ptrace(2), semop(2), setcompat(2), shmop(2), times(2), wait(2), wait3(2N), signal(3).

## NAME

fsmount – mount an NFS file system

## SYNOPSIS

```
#include <sys/mount.h>
int fsmount(type, dir, flags, data)
int type;
char *dir;
int flags;
caddr_t data;
```

## DESCRIPTION

fsmount 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 user ID 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 file system. 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 file system 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**

`fsmount` 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**

`fsmount` will fail when one of the following occurs:

| | |
|---|---|
| `[EPERM]` | The caller is not the superuser. |
| `[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. |
| `[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 *special* or *name* is not a directory. |
| `[EPERM]` | The pathname of *special* or *name* contains a character with the high-order bit set. |
| `[ENAMETOOLONG]` | The pathname of *special* or *name* was too long. |
| `[ENOENT]` | *special* or *name* does not exist. |
| `[EACCES]` | Search permission is denied for a component of the path prefix of *special* or *name*. |

| | |
|---|---|
| [EFAULT] | *special* or *name* points outside the process's allocated address space. |
| [ELOOP] | Too many symbolic links were encountered in translating the pathname of *special* or *name*. |
| [EIO] | An I/O error occurred while reading from or writing to the file system. |

**SEE ALSO**

unmount(2), umount(2), mount(3).

**BUGS**

Too many errors appear to the caller as one value.

**NAME**

fsync – synchronize a file's in-core state with that on disk

**SYNOPSIS**

```
int  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 which 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**

fsync fails if:

[EBADF] *fd* is not a valid descriptor.

[EINVAL] *fd* refers to a socket, not to a file.

**SEE ALSO**

sync(1), sync(2).

**BUGS**

The current implementation of this call is expensive for large files.

## NAME

getdirentries – gets directory entries in a file system independent format

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/dir.h>

int getdirentries(d, buf, nbytes, basep)
int d;
char *buf;
int nbytes;
long *basep
```

## DESCRIPTION

getdirentries attempts to put directory entries from the directory referenced by the descriptor *d* into the buffer pointed to by *buf*, in a file system 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 file systems.

The data in the buffer is a series of direct structures. The direct structure is defined as

```
struct direct {
        unsigned long   d_fileno;
        unsigned short  d_reclen;
        unsigned short  d_namlen;
        char            d_name[MAXNAMELEN + 1];
};
```

The d_fileno entry is a number which is unique for each distinct file in the file system. 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 and d_namelen entries specify the actual file name and its length.

Upon return, the actual number of bytes transferred is returned. The current position pointer associated with *d* 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). The *basep* entry is a pointer to a location into which the current position of the buffer just transferred is placed. 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 *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**

`link`(2), `lseek`(2), `open`(2), `stat`(2), `directory`(3).

## NAME

getdomainname, setdomainname – get/set name of current network domain

## SYNOPSIS

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

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

## DESCRIPTION

getdomainname returns the name of the network 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 superuser 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* or *namelen* parameter gave an invalid address. |
| [EPERM] | The caller was not the superuser. |

## BUGS

Domain names are limited to 255 characters.

**NAME**

getdtablesize – get descriptor table size

**SYNOPSIS**

```
int getdtablesize()
```

**DESCRIPTION**

Each process has a fixed size descriptor table which is guaranteed to have at least the maximum number of open slots OPEN_MAX. The entries in the descriptor table are numbered with small integers starting at 0. getdtablesize returns the size of this table.

**SEE ALSO**

close(2), dup(2), open(2).

## NAME

getgroups – get group access list

## SYNOPSIS

```
#include <sys/param.h>

int getgroups(gidsetlen, gidset)
int 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 getgroups 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.

**NAME**

gethostid, sethostid – get/set unique identifier of current host

**SYNOPSIS**

```
int  gethostid()

int  sethostid(hostid)
int  hostid
```

**DESCRIPTION**

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

**RETURN VALUE**

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

sethostid returns zero upon successful completion and −1 upon error.

**SEE ALSO**

hostid(1N), gethostname(2N).

**BUGS**

32 bits for the identifier is too small.

## NAME

gethostname, sethostname – get/set name of current host

## SYNOPSIS

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

int 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 superuser 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 was not the superuser. |

## SEE ALSO

gethostid(2N).

## BUGS

Host names are limited to 255 characters.

## NAME

getitimer, setitimer – get/set value of interval timer

## SYNOPSIS

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

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 nonzero, it indicates the time to the next timer expiration. If it_interval is nonzero, 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 nonzero).

Time values smaller than the resolution of the system clock are rounded up to this resolution (16 milliseconds on this system, 10 milliseconds on the VAX).

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 nonzero, 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**

getpeername – get name of connected peer

**SYNOPSIS**

```
int 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).

**RETURN VALUES**

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

**ERRORS**

getpeername fails if:

| | |
|---|---|
| [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**

bind(2N), getsockname(2N), socket(2N).

## NAME

getpid, getpgrp, getppid – get process, process group, and parent process IDs

## SYNOPSIS

```
int  getpid()
int  getpgrp()
int  getppid()
```

## DESCRIPTION

The getpid system call returns the process ID of the calling process. Each active process in the system is uniquely identified by a positive integer. The range of this integer is from 1 to the system-imposed limit, or PID_MAX.

The getpgrp system call returns the process group ID of the calling process. Each active process is a member of a process group that is identified by a positive integer. This grouping permits the signaling of related processes.

The getppid system call returns the parent process ID of the calling process. The parent process ID is the process ID of its creator.

## RETURN VALUE

| | |
|---|---|
| getpid | returns the process ID of the calling process. |
| getpgrp | returns the process group ID of the calling process. |
| getppid | returns the parent process ID of the calling process. |

These system calls are useful for generating uniquely-named temporary files.

## SEE ALSO

exec(2), fork(2), gethostid(2N), intro(2), setpgrp(2), signal(3).

**NAME**

getsockname – get socket name

**SYNOPSIS**

```
int 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).

**RETURN VALUES**

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

**ERRORS**

getsockname fails if:

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

**SEE ALSO**

bind(2N), getpeername(2N), getsockopt(2N), socket(2N).

## NAME

getsockopt, setsockopt – get and set options on sockets

## SYNOPSIS

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

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

int 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 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 of the requested options(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; see socket(2N). Options at other protocol levels vary in format and name; consult the appropriate entries in Section 5 of this manual (appropriate entries are marked (5P)).

**RETURN VALUE**

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

**ERRORS**

The callls fail if:

| | |
|---|---|
| [EBADF] | The argument *s* is not a valid descriptor. |
| [ENOTSOCK] | The argument *s* is a file, not a socket. |
| [ENOPROTOOPT] | The option is unknown. |
| [EFAULT] | The options are not in a valid part of the process address space. |

**SEE ALSO**

getsockname(2N), socket(2N), getprotoent(3N).

## NAME

gettimeofday, settimeofday – get/set date and time

## SYNOPSIS

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

int gettimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;

int 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 referenced 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 only when Dayling Savings Time is in effect.

Only the superuser may set the time of day or time zone. Changes to the time zone structure are effective for the current process only.

## RETURN VALUE

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 calls fail if:

| | |
|---|---|
| [EFAULT] | An argument address referenced invalid memory. |
| [EPERM] | A user other than the superuser attempted to set the time. |

**SEE ALSO**

date(1), adjtime(2), time(2), stime(2), ctime(3).

## NAME

getuid, geteuid, getgid, getegid – get real user, effective user, real group, and effective group IDs

## SYNOPSIS

```
unsigned short getuid()
unsigned short geteuid()
unsigned short getgid()
unsigned short getegid()
```

## DESCRIPTION

Each user allowed on the system is identified by a positive integer called a real user ID. The `getuid` system call returns the real user ID of the calling process.

Each active process has an effective user ID which is equal to the process's real user ID (unless the process of one of its ancestors evolved from a fail that had the set-user-ID bit set; see `exec`(2)). The `geteuid` system call returns the effective user ID of the calling process.

Each user is a member of a group which is identified by a positive integer called a real group ID. The `getgid` system call returns the real group ID of the calling process.

Each active process has an effective group ID which is equal to the process's real group ID (unless the process of one of its ancestors evolved from a fail that had the set-group-ID bit set; see `exec`(2)). The `getegid` system call returns the effective group ID of the calling process.

## RETURN VALUE

| | |
|---|---|
| `getuid` | returns the real user ID of the calling process. |
| `geteuid` | returns the effective user ID of the calling process. |
| `getgid` | returns the real group ID of the calling process. |
| `getegid` | returns the effective group ID of the calling process. |

## SEE ALSO

`intro`(2), `setreuid`(2), `setuid`(2).

**NAME**

ioctl – control device

**SYNOPSIS**

```
int ioctl(fildes, request, arg)
int fildes, request;
```

**DESCRIPTON**

ioctl performs a variety of functions on character special files (devices). Section 7 of the *A/UX System Administrator's Reference* describes the ioctl requests that apply to the given device.

**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] | *fildes* is not a valid open file descriptor. |
| [ENOTTY] | *fildes* is not associated with a character special device. |
| [EINVAL] | *request* or *arg* is not valid. See Section 7 of the *A/UX System Administrator's Reference*. |
| [EINTR] | A signal was caught during the ioctl system call. |

**SEE ALSO**

intro(2), fcntl(2), intro(7). termio(7).

## NAME

kill – send a signal to a process or a group of processes

## SYNOPSIS

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

## DESCRIPTION

kill sends a signal to a process or a group of processes. The process or group of processes to which the signal is to be sent is specified by *pid*. The signal that is to be sent is specified by *sig* and is either one from the list given in signal(3), or 0. If *sig* is 0 (the null signal), error checking is performed but no signal is actually sent. This can be used to check the validity of *pid*.

The real or effective user ID of the sending process must match the real or effective user ID of the receiving process, unless the effective user ID of the sending process is the superuser.

The processes with a process ID of 0 and a process ID of 1 are special processes (see intro(2)) and will be referred to below as *proc0* and *proc1* respectively.

If *pid* is greater than zero, *sig* will be sent to the process whose process ID is equal to *pid*. *pid* may equal 1.

If *pid* is 0, *sig* will be sent to all processes excluding *proc0* and *proc1* whose process group ID is equal to the process group ID of the sender.

If *pid* is –1 and the effective user ID of the sender is not the superuser, *sig* will be sent to all processes excluding *proc0* and *proc1* whose real user ID is equal to the effective user ID of the sender.

If *pid* is –1 and the effective user ID of the sender is the superuser, *sig* will be sent to all processes excluding *proc0* and *proc1*.

If *pid* is negative but not –1, *sig* will be sent to all processes whose process group ID is equal to the absolute value of *pid*.

## 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 one or more of the following are true:

| | |
|---|---|
| [EINVAL] | *sig* is not a valid signal number. |
| [EINVAL] | *sig* is SIGKILL and *pid* is 1 (*proc1*). |
| [ESRCH] | No process can be found corresponding to that specified by *pid*. |
| [EPERM] | The sending process is not sending to itself, its effective user ID is not the superuser, and its real or effective user ID does not match the real or effective user ID of the receiving process. |

**SEE ALSO**

kill(1), getpid(2), setpgrp(2), sigvec(2), signal(3).

## NAME

link – link to a file

## SYNOPSIS

```
int link(path1, path2)
char *path1, *path2;
```

## DESCRIPTION

link creates a new link (directory entry) for an existing file. *path1* points to a path name naming an existing file. *path2* points to a path name naming the new directory entry to be created.

## 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. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOENT] | A component of either path prefix does not exist. |
| [EACCES] | A component of either path prefix denies search permission. |
| [ENOENT] | The file named by *path1* does not exist. |
| [EEXIST] | The link named by *path2* exists. |
| [EPERM] | The file named by *path1* is a directory and the effective user ID is not the superuser. |
| [EXDEV] | The link named by *path2* and the file named by *path1* are on different logical devices (file systems). |
| [ENOENT] | *path2* points to a null path name. |

| | |
|---|---|
| [EACCES] | The requested link requires writing in a directory with a mode that denies write permission. |
| [EROFS] | The requested link requires writing in a directory on a read-only file system. |
| [EFAULT] | *path* points outside the allocated address space of the process. |
| [EMLINK] | The maximum number of links to a file would be exceeded. |

**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(2N), a backlog for incoming connections is specified with listen(2N) and then the connections are accepted with accept(2N). The listen call applies only to sockets of type SOCK_STREAM or SOCK_PKTSTREAM.

The *backlog* parameter defines the maximum length the queue of pending connections may grow to.

## RETURN VALUE

A 0 return value indicates success; –1 indicates an error.

## ERRORS

listen will fail if:

| | |
|---|---|
| [EBADF] | The argument *s* is not a valid descriptor. |
| [ENOTSOCK] | The argument *s* is not a socket. |
| [EOPNOTSUPP] | The operation is not supported on a socket. |

If a connection request arrives with the queue full the client will receive an error with an indication of ECONNREFUSED. The socket is not of a type that supports the operation listen.

## SEE ALSO

accept(2N), connect(2N), socket(2N).

## BUGS

The *backlog* is currently limited (silently) to 5.

**NAME**

`locking` – provide exclusive file regions for reading or writing

**SYNOPSIS**

```
int locking(fildes, mode, size)
int fildes;
int mode;
int size;
```

**DESCRIPTION**

`locking` will allow a specified number of bytes to be accessed only by the locking process (mandatory locking). Other processes which attempt to lock, read, or write the locked area will sleep until the area becomes unlocked. (Advisory locking is available via `lockf`(3C)).

*fildes* is the word returned from a successful `open`, `creat`, `dup`, or `pipe` system call.

*mode* is zero to unlock the area. *mode* is one or two for making the area locked. If the *mode* is one and the area has some other lock on it, then the process will sleep until the entire area is available. If the *mode* is two and the area is locked, an error will be returned.

*size* is the number of contiguous bytes to be locked or unlocked. The area to be locked starts at the current offset in the file. If *size* is zero, the area to the end of file is locked.

The potential for a deadlock occurs when a process controlling a locked area is put to sleep by accessing another process's locked area. Thus calls to `locking`, `read`, or `write` scan for a deadlock prior to sleeping on a locked area. An error return is made if sleeping on the locked area would cause a deadlock.

Lock requests may, in whole or part, contain or be contained by a previously locked area for the same process. When this or adjacent areas occur, the areas are combined into a single area. If the request requires a new lock element with the lock table full, an error is returned, and the area is not locked.

Unlock requests may, in whole or part, release one or more locked regions controlled by the process. When regions are not fully released, the remaining areas are still locked by the process. Release of the center section of a locked area requires an additional lock element to hold the cut off section. If the lock table is full, an error is returned, and the requested area is not released.

While locks may be applied to special files or pipes, read/write operations will not be blocked. Locks may not be applied to a directory.

Note that `close`(2) automatically removes any locks that were associated with the closed file descriptor.

**RETURN VALUE**

The value –1 is returned if the file does not exist, or if a deadlock using file locks would occur.

**ERRORS**

`locking` will fail if the following are true:

| | |
|---|---|
| `[EACCES]` | The area is already locked by another process. |
| `[EDEADLOCK]` | Returned by `read`, `write`, or `locking` if a deadlock would occur. |
| `[EDEADLOCK]` | Locktable overflow. |
| `[EREMOTE]` | *fildes* is a file descriptor that refers to file on a remotely mounted file system. |

**SEE ALSO**

`close`(2), `creat`(2), `dup`(2), `open`(2), `read`(2), `write`(2), `lockf`(3C).

## NAME

lseek – move read/write file pointer

## SYNOPSIS

```
long lseek(fildes, offset, whence)
int fildes;
long offset;
int whence;
```

## DESCRIPTION

*fildes* is a file descriptor returned from a `creat`, `open`, `dup`, or `fcntl` system call. `lseek` sets the file pointer associated with *fildes* as follows:

If *whence* is 0, the pointer is set to *offset* bytes.

If *whence* is 1, the pointer is set to its current location plus *offset*.

If *whence* is 2, the pointer is set to the size of the file plus *offset*.

Upon successful completion, the resulting pointer location, as measured in bytes from the beginning of the file, is returned.

## RETURN VALUE

Upon successful completion, a non-negative integer indicating the 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 one or more of the following are true:

`[EBADF]` *fildes* is not an open file descriptor.

`[ESPIPE]` *fildes* is associated with a pipe or FIFO.

`[EINVAL]` and the `SIGSYS` signal
*whence* is not 0, 1, or 2.

`[EINVAL]` The resulting file pointer would be negative.

Some devices are incapable of seeking. The value of the file pointer associated with such a device is undefined.

## SEE ALSO

`creat`(2), `dup`(2), `fcntl`(2), `open`(2).

**NAME**

mkdir – make a directory file

**SYNOPSIS**

```
int 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:

| | |
|---|---|
| [EPERM] | The process's effective user ID is not the superuser. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [EPERM] | The *path* argument contains a byte with the high-order bit set. |
| [ENOTDIR] | A component of the path prefix is not a directory. |
| [ENOENT] | A component of the path prefix does not exist. |

| | |
|---|---|
| [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. |
| [ELOOP] | Too many symbolic links were encountered in translating the pathname. |
| [EIO] | An I/O error occured while writing to the file system. |

**SEE ALSO**

mkdir(1), chmod(2), rmdir(2), stat(2), umask(2).

## NAME

mknod – make a directory, or a special or ordinary file

## SYNOPSIS

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

## DESCRIPTION

mknod creates a new file named by the path name pointed to by *path*. The mode of the new file is initialized from *mode*, where the value of *mode* is interpreted as follows:

0170000 file type mask; one of the following:

0010000 FIFO special
0020000 character special
0040000 directory
0060000 block special
0100000 or 0000000 ordinary file
0120000 symbolic link
0140000 socket
0004000 set user ID on execution
0002000 set group ID on execution
0001000 save text image after execution

0000777 access permissions; constructed from the following

0000400 read by owner
0000200 write by owner
0000100 execute (search on directory) by owner
0000070 read, write, execute (search) by group
0000007 read, write, execute (search) by others

The owner ID of the file is set to the effective user ID of the process. The group ID of the file is set to the effective group ID of the process.

Values of *mode* other than those above are undefined and should not be used. 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). 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 superuser for file types other than FIFO special.

**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 new file will not be created if one or more of the following are true:

| | |
|---|---|
| [EPERM] | The effective user ID of the process is not superuser. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOTDIR] | A component of the path prefix is not a directory. |
| [ENOENT] | A component of the path prefix does not exist. |
| [EROFS] | The directory in which the file is to be created is located on a read-only file system. |
| [EEXIST] | The named file exists. |
| [EFAULT] | *path* points outside the allocated address space of the process. |

**SEE ALSO**

mkdir(1), mknod(1), chmod(2), exec(2), stat(2), umask(2), fs(4), stat(5).

**NAME**

msgctl – message control operations

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl(id, cmd, buf)
int id, cmd;
struct msqid_ds *buf;
```

**DESCRIPTION**

msgctl provides a variety of message control operations as specified by *cmd*. The following *cmd*s are available:

IPC_STAT — Place the current value of each member of the data structure associated with *id* into the structure referenced by *buf*. The contents of this structure are defined in intro(2).

IPC_SET — Set the value of the following members of the data structure associated with *id* to the corresponding value found in the structure referenced by *buf*:

```
msg_perm.uid
msg_perm.gid
msg_perm.mode (only low 9 bits)
msg_qbytes
```

This *cmd* can only be executed by a process that has an effective user ID equal to either that of superuser or to the value of msg_perm.uid in the data structure associated with *id*. Only the superuser can raise the value of msg_qbytes.

IPC_RMID — Remove the message queue identifier specified by *id* from the system and destroy the message queue and data structure associated with it. This *cmd* can only be executed by a process that has an effective user ID equal to either that of super user or to the value of msg_perm.uid in the data structure associated with *id*. The identifier and its associated data structure are not actually removed until there are no more referencing processes. See ipcrm(1), and ipcs(1).

**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**

`msgctl` will fail if one or more of the following are true:

`[EINVAL]` *id* is not a valid message queue identifier.

`[EINVAL]` *cmd* is not a valid command.

`[EACCES]` *cmd* is equal to `IPC_STAT` and operation permission is denied to the calling process (see `intro`(2)).

`[EPERM]` *cmd* is equal to `IPC_RMID` or `IPC_SET`. The effective user ID of the calling process is not equal to that of superuser and it is not equal to the value of `msg_perm.uid` in the data structure associated with *id*.

`[EPERM]` *cmd* is equal to `IPC_SET`, an attempt is being made to increase to the value of `msg_qbytes`, and the effective user ID of the calling process is not equal to that of superuser.

`[EFAULT]` *buf* points to an illegal address.

**SEE ALSO**

`intro`(2), `msgget`(2), `msgop`(2).

**NAME**

msgget – get message queue

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(key, msgflg)
key_t key;
int msgflg;
```

**DESCRIPTION**

`msgget` returns the message queue identifier associated with *key*.

A message queue identifier and associated message queue and data structure (see `intro`(2)) are created for *key* if one of the following is true:

*key* is equal to `IPC_PRIVATE`.

*key* does not already have a message queue identifier associated with it, and (*msgflg* `&` `IPC_CREAT`) is "true".

The key `IPC_PRIVATE` will create an identifier and associated data structure that is unique to the calling process and its children.

Upon creation, the data structure associated with the new message queue identifier is initialized as follows:

`msg_perm.cuid`, `msg_perm.uid`, `msg_perm.cgid`, and `msg_perm.gid` are set equal to the effective user ID and effective group ID, respectively, of the calling process.

The low-order 9 bits of `msg_perm.mode` are set equal to the low-order 9 bits of *msgflg*.

`msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime`, and `msg_rtime` are set equal to 0.

`msg_ctime` is set equal to the current time.

`msg_qbytes` is set equal to the system limit.

**RETURN VALUE**

Upon successful completion, a non-negative integer, namely a message queue identifier, is returned. Otherwise, a value of –1 is returned and `errno` is set to indicate the error.

**ERRORS**

`msgget` will fail if one or more of the following are true:

[EACCES] A message queue identifier exists for *key*, but operation permission (see intro(2)) as specified by the low-order 9 bits of *msgflg* would not be granted.

[ENOENT] A message queue identifier does not exist for *key* and (*msgflg* & IPC_CREAT) is "false".

[ENOSPC] A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system wide would be exceeded.

[EEXIST] A message queue identifier exists for *key* but ((*msgflg* & IPC_CREAT) && (*msgflg* & IPC_EXCL)) is "true".

**SEE ALSO**

intro(2), msgctl(2), msgop(2).

## NAME

msgop, msgsnd, msgsrv – message operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd(msqid, msgp, msgsz, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz, msgflg;

int msgrcv(msqid, msgp, msgsz, msgtyp, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz;
long msgtyp;
int msgflg;
```

## DESCRIPTION

msgsnd is used to send a message to the queue associated with the message queue identifier specified by *msqid*. *msgp* points to a structure containing the message. This structure is composed of the following members:

```
long mtype;      /* message type */
char mtext[];    /* message text */
```

*mtype* is a positive integer that can be used by the receiving process for message selection (see msgrcv below). *mtext* is any text of length *msgsz* bytes. *msgsz* can range from 0 to a system-imposed maximum.

*msgflg* specifies the action to be taken if one or more of the following are true:

The number of bytes already on the queue is equal to msg_qbytes (see intro(2)).

The total number of messages on all queues systemwide is equal to the system-imposed limit.

These actions are as follows:

If (*msgflg* & IPC_NOWAIT) is "true", the message will not be sent and the calling process will return immediately.

If (*msgflg* & IPC_NOWAIT) is "false", the calling process will suspend execution until one of the following

occurs:

The condition responsible for the suspension no longer exists, in which case the message is sent.

*msqid* is removed from the system (see `msgctl`(2)). When this occurs, `errno` is set equal to `EIDRM`, and a value of −1 is returned.

The calling process receives a signal that is to be caught. In this case the message is not sent and the calling process resumes execution in the manner prescribed in `sigvec`(2)).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see `intro`(2)).

`msg_qnum` is incremented by 1.

`msg_lspid` is set equal to the process ID of the calling process.

`msg_stime` is set equal to the current time.

`msgrcv` reads a message from the queue associated with the message queue identifier specified by *msqid* and places it in the structure pointed to by *msgp*. This structure is composed of the following members:

```
long mtype;      /* message type */
char mtext[];    /* message text */
```

*mtype* is the received message's type as specified by the sending process. *mtext* is the text of the message. *msgsz* specifies the size in bytes of *mtext*. The received message is truncated to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* `&` `MSG_NOERROR`) is "true". The truncated part of the message is lost and no indication of the truncation is given to the calling process.

*msgtyp* specifies the type of message requested as follows:

If *msgtyp* is equal to 0, the first message on the queue is received.

If *msgtyp* is greater than 0, the first message of type *msgtyp* is received.

If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* is received.

*msgflg* specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

If (*msgflg* & IPC_NOWAIT) is "true", the calling process will return immediately with a return value of –1 and errno is set to ENOMSG.

If (*msgflg* & IPC_NOWAIT) is "false", the calling process will suspend execution until one of the following occurs:

A message of the desired type is placed on the queue.

*msqid* is removed from the system. When this occurs, errno is set equal to EIDRM, and a value of –1 is returned.

The calling process receives a signal that is to be caught. In this case a message is not received and the calling process resumes execution in the manner prescribed in sigvec(2)).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see intro(2)).

msg_qnum is decremented by 1.

msg_lrpid is set equal to the process ID of the calling process.

msg_rtime is set equal to the current time.

RETURN VALUES

If msgsnd or msgrcv return due to the receipt of a signal, a value of –1 is returned to the calling process and errno is set to EINTR. If they return due to removal of *msqid* from the system, a value of –1 is returned and errno is set to EIDRM.

Upon successful completion, the return value is as follows:

*msgsnd* returns a value of 0.

msgrcv returns a value equal to the number of bytes actually placed into *mtext*.

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

ERRORS

msgsnd will fail and no message will be sent if one or more of the following are true:

[EINVAL] *msqid* is not a valid message queue identifier.

[EACCES] Operation permission is denied to the calling process (see intro(2)).

[EINVAL] *mtype* is less than 1.

[EAGAIN] The message cannot be sent for one of the reasons cited above and (*msgflg* & IPC_NOWAIT) is "true".

[EINVAL] *msgsz* is less than zero or greater than the system-imposed limit.

[EFAULT] *msgp* points to an illegal address.

msgrcv will fail and no message will be received if one or more of the following are true:

[EINVAL] *msqid* is not a valid message queue identifier.

[EACCES] Operation permission is denied to the calling process.

[EINVAL] *msgsz* is less than 0.

[E2BIG] *mtext* is greater than *msgsz* and (*msgflg* & MSG_NOERROR) is "false".

[ENOMSG] The queue does not contain a message of the desired type and (*msgtyp* & IPC_NOWAIT) is "true".

[EFAULT] *msgp* points to an illegal address.

**SEE ALSO**

intro(2), msgctl(2), msgget(2), sigvec(2), signal(3).

## NAME

nice – change priority of a process

## SYNOPSIS

```
int  nice(incr)
int  incr;
```

## DESCRIPTION

nice adds the value of *incr* to the value of the calling process. A process's nice value is a positive number for which a higher value results in lower CPU priority.

A maximum nice value of 39 and a minimum nice value of 0 are imposed by the system. Requests for values above or below these limits result in the nice value being set to the corresponding limit.

## RETURN VALUE

Upon successful completion, nice returns the new nice value minus 20. Otherwise, a value of –1 is returned and errno is set to indicate the error. If a value of –1 is a valid return value on successful completion (i.e., if your new nice value is 19), errno is not changed.

## ERRORS

nice will fail if:

[EPERM] nice will fail and not change the nice value if *incr* is negative or greater than 40 and the effective user ID of the calling process is not superuser.

## SEE ALSO

nice(1), exec(2).

## NAME

nfssvc, async_daemon – NFS daemons

## SYNOPSIS

```
int  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(1M), nfsd(1M).

**NAME**

nfs_getfh – get a file handle

**SYNOPSIS**

```
#include <rpc/types.h>
#include <sys/errno.h>
#include <sys/time.h>
#include <nfs/nfs.h>

int nfs_getfh (fildes, fhp)
int fildes;
fhandle_t *fhp;
```

**DESCRIPTION**

nfs_getfh returns the file handle associated with the file descriptor *fd*. This call is restricted to the superuser.

**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 int the global location errno.

**ERRORS**

The following errors may be returned by these calls:

| | |
|---|---|
| [EPERM] | The caller was not the superuser. |
| [EBADF] | *fd* is not a valid open file descriptor. |
| [EFAULT] | The *fhp* parameter gave an invalid address. |

NAME

open – open for reading or writing

SYNOPSIS

```
#include <fcntl.h>
int open(path, oflag [,mode])
char *path;
int oflag, mode;
```

DESCRIPTION

open opens a file descriptor for the named file and sets the file status flags according to the value of *oflag*. *path* points to a path name naming a file. *oflag* values are constructed by or-ing flags from the following list (only one of the first three flags below may be used):

O_RDONLY Open for reading only.

O_WRONLY Open for writing only.

O_RDWR Open for reading and writing.

O_NDELAY This flag may affect subsequent reads and writes. See read(2) and write(2).

When opening a FIFO with O_RDONLY or O_WRONLY set:

If O_NDELAY is set:

An open for reading-only will return without delay. An open for writing-only will return an error if no process currently has the file open for reading.

If O_NDELAY is clear:

An open for reading-only will block until a process opens the file for writing. An open for writing-only will block until a process opens the file for reading.

When opening a file associated with a communication line:

If O_NDELAY is set:

The open will return without waiting for carrier.

If O_NDELAY is clear:

The open will block until carrier is present.

O_APPEND — If set, the file pointer will be set to the end of the file prior to each write.

O_CREAT — If the file exists, this flag has no effect. Otherwise, the owner ID of the file is set to the effective user ID of the process, the group ID of the file is set to the effective group ID of the process, and the low-order 12 bits of the file mode are set to the value of *mode* modified as follows (see creat(2)):

All bits set in the file mode creation mask of the process are cleared. See umask(2).

The "save text image after execution bit" of the mode is cleared. See chmod(2).

O_TRUNC — If the file exists, its length is truncated to 0 and the mode and owner are unchanged.

O_EXCL — If O_EXCL and O_CREAT are set, open will fail if the file exists.

The file pointer used to mark the current position within the file is set to the beginning of the file.

The new file descriptor is set to remain open across exec system calls. See fcntl(2).

**RETURN VALUE**

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

**ERRORS**

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

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

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

[ENAMETOOLONG] — A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX.

| | |
|---|---|
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOENT] | O_CREAT is not set and the named file does not exist. |
| [EACCES] | A component of the path prefix denies search permission. |
| [EACCES] | *oflag* permission is denied for the named file. |
| [EISDIR] | The named file is a directory and *oflag* is write or read/write. |
| [EROFS] | The named file resides on a read-only file system and *oflag* is write or read/write. |
| [EMFILE] | The per-process open file limit would be exceeded. |
| [ENXIO] | The named file is a character special or block special file, and the device associated with this special file does not exist. |
| [ETXTBSY] | The file is a pure procedure (shared text) file that is being executed and *oflag* is write or read/write.<br><br>*Note:* If you are running an NFS system and you are accessing a shared binary remotely, it is possible that you will not get this errno. |
| [EFAULT] | *path* points outside the allocated address space of the process. |
| [EEXIST] | O_CREAT and O_EXCL are set, and the named file exists. |
| [ENXIO] | O_NDELAY is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading. |
| [EINTR] | A signal was caught during the open system call. |
| [ENFILE] | The system file table is full. |

**SEE ALSO**

chmod(2), close(2), creat(2), dup(2), fcntl(2), lseek(2), read(2), umask(2), write(2), fopen(3), ferror(3).

## NAME

pause – suspend process until signal

## SYNOPSIS

```
pause()
```

## DESCRIPTION

pause suspends the calling process until it receives a signal. The signal must be one that is not currently set to be ignored by the calling process.

If the signal causes termination of the calling process, pause will not return.

The behavior of pause will vary when a signal is caught by the calling process according to flags set by setcompat(2) or set42sig(3). If the COMPAT_SYSCALLS flag is set when control is returned from the signal catching function, then the process will once again pause; otherwise, flag not set will resume as above.

## ERRORS

If the signal is caught by the calling process and control is returned from the signal-catching function (see signal(3)), the calling process resumes execution from the point of suspension; with a return value of –1 from pause and errno set to EINTR.

## SEE ALSO

alarm(2), kill(2), wait(2), signal(3).

## NAME

phys – allow a process to access physical addresses

## SYNOPSIS

```
int phys (physnum, virtaddr, size, physaddr)
int physnum;
char *virtaddr;
unsigned int size;
char *physaddr;
```

## DESCRIPTION

The phys system call allows the superuser to map a region of physical memory into a process's virtual address space.

The calling process chooses *physnum* to specify the phys region this call references. The maximum number of regions per process is defined by the v_phys field in the var structure returned by uvar(2). *physnum* must be between zero and v_phys −1, and is only used to identify a particular phys region to the kernel during a phys system call.

*virtaddr* is the base virtual address for the region in the process's virtual address space, and *size* is the length in bytes of the desired region. The virtual address range of the region must not overlap any of the existing address space of the process, including text, data, stack, shared memory regions (see shmget(2)), and any other active phys regions. All addresses in this range must be valid user virtual addresses (see the example below). Care should also be taken to avoid placing a phys region at a virtual address that the data or stack segments might grow to encompass.

If *size* is zero, any previous phys mapping is cleared for the region specified by *physnum*.

A phys region's *virtaddr* and *size* are dependent on the implementation decisions for the memory management unit. In particular, the base *virtaddr* must be on a kernel segment boundary and the *size* will be rounded up to an integral multiple of the page size. These values may be computed from the v_segshift and v_pageshift fields returned by uvar(2); i.e., the segment size is

```
1 << v_segshift
```

and the page size is

```
1 << v_pageshift
```

The *physaddr* argument is the base physical address for the region. *physaddr* is rounded down to the previous page boundary. Also, *physaddr* to *physaddr + size* should be inside the range of physical addresses supported by the hardware. `phys` regions are inherited across `fork`(2) system calls and disowned across `execs`.

`phys` may only be executed by a process with an effective user ID of root.

As an example, suppose a process wishes to map a piece of memory-mapped hardware into its address space. This hardware has 0x8800 bytes of memory and control registers located at physical address 0xFA000000. By calling `uvar`(2), the process finds that `v_pageshift` is 12 and `v_segshift` is 20; thus, the page size is 0x1000 and the segment size is 0x100000. Also, `v_phys` is found to be 32, so any number from zero to 31 may be used for *physnum*.

The `var` structure also contains `v_ustart` and `v_uend`, the starting and ending virtual addresses for user processes. For this example, assume `v_ustart` is zero and `v_uend` is 0x20000000. The first few segments are used for the running program's text and data and the last are used for the user stack. The process might decide it is unlikely its data and text segment will exceed 0x4000000, which is an integral multiple of 0x100000 (the segment size).

The call:

```
phys(0, 0x4000000, 0x8800, 0xFA000000);
```

will allow the process access to physical locations from 0xFA00000 to 0xFA009000 by referencing virtual addresses 0x4000000 to 0x4009000. The range has been adjusted to 0x9000 bytes because that is the next page boundary.

In this example, referencing 0x4008804 (an address in the `phys` region, but outside of the known hardware memory) will result in unpredictable failures. A useless value may be read off the hardware lines, a write may appear to succeed without affecting anything, the program may get a `SIGSEGV` (see `signal`(3)), the hardware may react randomly, or the entire system may crash. There may be other possibilities depending on system configuration.

If the process wished to add another `phys` region without deleting the first region, the next available *virtaddr* would be 0x4100000 (the next segment boundary) and *physnum* could be any number from one to 31.

RETURN VALUES

The value zero is returned if the call was successful; otherwise −1 is returned. `phys` will fail if the effective user ID of the calling process is not root, if *virtaddr* or *physaddr* is not in the proper range, or if the range of virtual addresses overlaps a portion of the user's virtual address space that is already in use.

NOTES

`phys` is hardware and implementation dependent and must be used with extreme caution. The intention is to give the superuser complete access to the physical hardware. To insure maximum portability, *virtaddr* and *size* should be calculated as described in the example.

Different hardware may respond differently to mistakes in addressing. Sometimes all the bits of a physical address are not decoded, making (for example) 0xFD100000 the same as 0xFD000000. If *physaddr* or *size* is wrong it is possible to crash the system.

Most versions of UNIX do not support this system call.

SEE ALSO

`uvar`(2), `shmget`(2), `signal`(3).

## NAME

pipe – create an interprocess channel

## SYNOPSIS

```
int  pipe (fildes)
int  fildes[2];
```

## DESCRIPTION

pipe creates an I/O mechanism called a pipe and returns two file descriptors, *fildes*[0] and *fildes*[1]. *fildes*[0] is opened for reading and *fildes*[1] is opened for writing.

Up to PIPE_MAX bytes of data are buffered by the pipe before the writing process is blocked. A read only file descriptor *fildes*[0] accesses the data written to *fildes*[1] on a first-in-first-out (FIFO) basis.

## 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

pipe will fail if one or more of the following is true:

[EMFILE] pipe will fail if the per-process open file limit would be exceeded.

[ENFILE] The system file table is full.

## SEE ALSO

read(2), write(2).

## NAME

plock – lock process, text, or data in memory

## SYNOPSIS

```
#include <sys/lock.h>

int plock(op)
int op;
```

## DESCRIPTION

plock allows the calling process to lock its text segment (text lock), its data segment (data lock), or both its text and data segments (process lock) into memory. Locked segments are immune to all routine swapping. plock also allows these segments to be unlocked. The effective user ID of the calling process must be superuser to use this call. *op* specifies the following:

| | |
|---|---|
| PROCLOCK | lock text and data segments into memory (process lock) |
| TXTLOCK | lock text segment into memory (text lock) |
| DATLOCK | lock data segment into memory (data lock) |
| UNLOCK | remove locks |

## RETURN VALUE

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

## ERRORS

plock will fail and not perform the requested operation if one or more of the following are true:

| | |
|---|---|
| [EPERM] | The effective user ID of the calling process is not superuser. |
| [EAGAIN] | The system has temporarily exhausted its available memory or swap space. |
| [EINVAL] | *op* is equal to PROCLOCK and a process lock, a text lock, or a data lock already exists on the calling process. |
| [EINVAL] | *op* is equal to TXTLOCK and a text lock, or a process lock already exists on the calling process. |
| [EINVAL] | *op* is equal to DATLOCK and a data lock, or a process lock already exists on the calling process. |

[EINVAL] *op* is equal to UNLOCK and no type of lock exists on the calling process.

**SEE ALSO**

exec(2), exit(2), fork(2).

## NAME

profil – execution time profile

## SYNOPSIS

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

## DESCRIPTION

profil is used to report performance analysis of an application. *buff* points to an area of core whose length (in bytes) is given by *bufsiz*. After the call, the user's program counter (pc) is examined for each clock tick; *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; 2 maps all instructions onto the beginning of buff (producing a noninterrupting core clock).

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 exec is executed, but remains on in child and parent both after a fork. Profiling will be turned off if an update in buff would cause a memory fault.

## RETURN VALUE

Not defined.

## SEE ALSO

prof(1), monitor(3C).

## NAME

ptrace – process trace

## SYNOPSIS

```
int 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. Its primary use is for the implementation of breakpoint debugging. The child process behaves normally until it encounters a signal (see sigvec(2) for the list), at which time it enters a stopped state and its parent is notified via wait(2). When the child is in the stopped state, its parent can examine and modify its "core image" using ptrace. Also, the parent can cause the child either to terminate or continue, with the possibility of ignoring the signal that caused it to stop.

The *request* argument determines the precise action to be taken by ptrace and is one of the following:

0 This request must be issued by the child process if it is to be traced by its parent. It turns on the child's trace flag that stipulates that the child should be left in a stopped state upon receipt of a signal rather than the state specified by *func*; see sigvec(2). The *pid*, *addr*, and *data* arguments are ignored, and a return value is not defined for this request. Peculiar results will ensue if the parent does not expect to trace the child.

The remainder of the requests can only be used by the parent process. For each, *pid* is the process ID of the child. The child must be in a stopped state before these requests are made.

1, 2
With these requests, the word at location *addr* in the address space of the child is returned to the parent process. Either request 1 or request 2 may be used with equal results. The *data* argument is ignored. These two requests will fail if *addr* is not the start address of a word, in which case a value of –1 is returned to the parent process and the parent's errno is set to EIO.

3 With this request, the word at location *addr* in the child's USER area in the system's address space (see <sys/user.h>) is returned to the parent process. Addresses are system dependent. The *data* argument is

ignored. This request will fail if *addr* is not the start address of a word or is outside the USER area, in which case a value of −1 is returned to the parent process and the parent's errno is set to EIO.

4, 5
With these requests, the value given by the *data* argument is written into the address space of the child at location *addr*. Either request 4 or request 5 may be used with equal results. Upon successful completion, the value written into the address space of the child is returned to the parent. These two requests will fail if *addr* is a location in a pure procedure space and another process is executing in that space, or *addr* is not the start address of a word. Upon failure, a value of −1 is returned to the parent process and the parent's errno is set to EIO.

6 With this request, a few entries in the child's USER area can be written. *data* gives the value that is to be written and *addr* is the location of the entry. The few entries that can be written are:

the general registers
the condition codes
certain bits of the Processor Status Word

7 This request causes the child to resume execution. If the *data* argument is 0, all pending signals including the one that caused the child to stop are canceled before it resumes execution. If the *data* argument is a valid signal number, the child resumes execution as if it had incurred that signal, and any other pending signals are canceled. The *addr* argument must be equal to 1 for this request. Upon successful completion, the value of *data* is returned to the parent. This request will fail if *data* is not 0 or a valid signal number, in which case a value of −1 is returned to the parent process and the parent's errno is set to EIO.

8 This request causes the child to terminate with the same consequences as exit(2).

9 This request sets the trace bit in the Processor Status Word of the child and then executes the same steps as listed above for request 7. The trace bit causes an interrupt upon completion of one machine instruction. This effectively allows single stepping of the child.

*Note:* The trace bit remains set after an interrupt.

10 Read user register; *pid* = child process ID; *addr* = register number; *data* is ignored; returns value of child's register.

11 Write user register; *pid* = child process ID; *addr* = register number; *data* = integer value to be written into named register.

*Note:* For both requests 10 and 11, the register numbers are as shown below for the 68000 family (these numbers are system dependent).

| Register | Register # | Register | Register # |
|---|---|---|---|
| d0 | 0 | a1 | 9 |
| d1 | 1 | a2 | 10 |
| d2 | 2 | a3 | 11 |
| d3 | 3 | a4 | 12 |
| d4 | 4 | a5 | 13 |
| d5 | 5 | a6 | 14 |
| d6 | 6 | SP | 15 |
| d7 | 7 | PC | 16 |
| a0 | 8 | PS | 17 |

To forestall possible fraud, `ptrace` inhibits the set-user-ID facility on subsequent `exec`(2) calls. If a traced process calls `exec`, it will stop before executing the first instruction of the new image showing signal `SIGTRAP`.

**ERRORS**

`ptrace` will in general fail if one or more of the following are true:

`[EIO]` *request* is an illegal number.

`[ESRCH]` *pid* identifies a child that does not exist or has not executed a `ptrace` with request `0`.

**NOTE**

Request 11 largely supercedes request 6, and request 10 largely supercedes request 3 (request 3 can read any part of the child's user area while request 10 can only read register values of the child).

**SEE ALSO**

`exec`(2), `sigvec`(2), `wait`(2), `signal`(3).

## NAME

read, readv – read from file

## SYNOPSIS

```
int read(fildes, buf, nbytes)
int fildes;
char *buf;
int nbytes;

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

int readv(fildes, iov, iovcnt)
int fildes;
struct iovec *iov;
int iovcnt;
```

## DESCRIPTION

read attempts to read *nbytes* bytes from the file associated with *fildes* 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 *iovec*

*fildes* is a file descriptor obtained from a creat, open, dup, fcntl, pipe, or socket system call.

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 devices capable of seeking, the read starts at a position in the file given by the file pointer associated with *fildes*. Upon return from read, the file pointer is incremented by the number of bytes actually read.

Devices that are incapable of seeking always read from the current position. The value of a file pointer associated with such a file is undefined.

Upon successful completion, read and readv return the number of bytes actually read and placed in the buffer; this

number may be less than *nbytes* if the file is associated with a communication line (see `ioctl`(2), `socket`(2N), and `termio`(7)), or if the number of bytes left in the file is less than *nbytes* bytes. A value of 0 is returned when an end-of-file has been reached.

When attempting to read from an empty pipe (or FIFO):

If `O_NDELAY` is set, the read will return a 0.

If `O_NDELAY` is clear, the read will block until data is written to the file or the file is no longer open for writing.

When attempting to read a file associated with a tty that has no data currently available:

If `O_NDELAY` is set, the read will return a 0.

If `O_NDELAY` is clear, the read will block until data becomes available.

**RETURN VALUE**

Upon successful completion, a nonnegative integer is returned indicating the number of bytes actually read. Otherwise, a −1 is returned and `errno` is set to indicate the error.

**ERRORS**

When attempting to read from a stream that has no data currently available, if `O_NDELAY` is set, the `read` will return −1 and `errno` will be set to `ENODATA`. If `O_NDELAY` is clear, the `read` will block until data becomes available.

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

| | |
|---|---|
| `[EIO]` | A physical I/O error has occurred. |
| `[ENXIO]` | The device associated with the file descriptor is a block-special or character-special file and the value of the file pointer is out of range. |
| `[EWOULDBLOCK]` | The file was marked for nonblocking I/O, and no data were ready to be read. |
| `[EBADF]` | *fildes* is not a valid file descriptor open for reading. |
| `[EFAULT]` | *buf* points outside the allocated address space. |
| `[EINTR]` | A signal was caught during the `read` system call. |

[ENODATA] A read from a stream was attempted when no data was available and O_NDELAY was set.

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

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

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

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

**SEE ALSO**

creat(2), fcntl(2), ioctl(2), open(2), pipe(2), socket(2N), setcompat(2), termio(7).

## NAME

readlink – read value of a symbolic link

## SYNOPSIS

```
int readlink(path, buf, bufsiz)
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:

| | |
|---|---|
| [EPERM] | The *path* argument contained a byte with the high-order bit set. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOENT] | The pathname was too long. |
| [ENOTDIR] | A component of the path prefix is not a directory. |
| [ENOENT] | The named file does not exist. |
| [ENXIO] | The named file is not a symbolic link. |
| [EACCES] | Search permission is denied on a component of the path prefix. |
| [EPERM] | The effective user ID does not match the owner of the file and the effective user ID is not the superuser. |
| [EINVAL] | The named file is not a symbolic link. |

| | |
|---|---|
| [EFAULT] | *buf* extends outside the process's allocated address space. |
| [ELOOP] | Too many symbolic links were encountered in translating the pathname. |

**SEE ALSO**

stat(2), lstat(2), symlink(2).

**NAME**

reboot – reboot the system

**SYNOPSIS**

```
reboot()
```

**DESCRIPTION**

reboot causes the kernel to execute the initial bootstrap code that was used to boot the operating system.

The reboot(2) call takes the place of a manual restart and requres effective user ID of root (superuser) to run.

**SEE ALSO**

reboot(1M).

## NAME

recv, recvfrom, recvmsg – receive a message from a socket

## SYNOPSIS

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

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

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

int recvmsg(s, msg, flags)
int s;
struct msghdr msg[];
int flags;
```

## DESCRIPTION

recv, recvfrom, and recvmsg are used to receive messages from a socket.

The recv call may be used only on a connected socket (i.e., when connect(2N) has been used), 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 nonzero, 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. 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(2N).

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 –1 is returned with the external variable errno set to EWOULDBLOCK.

The select(2N) call may be used to determine when more data arrives.

The *flags* argument to a `send` call is formed by `or`'ing one or more of the values,

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

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_namelen;    /* size of address */
  struct   iov *msg_iov;   /* scatter/gather array */
  int      msg_iovlen;     /* # elements in msg_iov */
  caddr_t  msg_accrights; /* access rights sent/received */
  int     msg_accrightslen;
};
```

Here `msg_name` and `msg_namelen` specify the destination address if the socket is unconnected; `msg_name` may be given as a null pointer if no names are desired or required. The `msg_iov` and `msg_iovlen` describe the scatter gather locations. Access rights to be sent along with the message are specified in `msg_accrights`, which has length `msg_accrightslen`.

## RETURN VALUE

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

## ERRORS

The calls fail if:

| | |
|---|---|
| [EBADF] | The argument *s* is an invalid descriptor. |
| [ENOTSOCK] | The argument *s* is not a socket. |
| [EWOULDBLOCK] | The socket is marked nonblocking and the receive operation would block. |
| [EINTR] | The receive was interrupted by delivery of a signal before any data was available for the receive. |
| [EFAULT] | The data was specified to be received into a nonexistent or protected part of the process address space. |

**SEE ALSO**

connect(2N), read(2), send(2N), socket(2N).

## NAME

rename – change the name of a file

## SYNOPSIS

```
int 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 nondirectories), and must reside on the same file system.

rename guarantees that an instance of the file will always exist, even if the system should crash in the middle of the operation.

## 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 files named as arguments will be affected if any of the following are true:

| | |
|---|---|
| [ENOTDIR] | A component of either path prefix is not a directory. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |

| | |
|---|---|
| [ENOENT] | A component of either path prefix does not exist. |
| [EACCES] | A component of either path prefix denies search permission. |
| [ENOENT] | The file named by *from* does not exist. |
| [EPERM] | The file named by *from* is a directory and the effective user ID is not superuser. |
| [EXDEV] | The link named by *to* and the file named by *from* are on different logical devices (file systems). |
| [EACCES] | The requested link requires writing in a directory with a mode that denies write permission. |
| [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*. |

**SEE ALSO**

mv(1), open(2).

**NAME**

rmdir – remove a directory file

**SYNOPSIS**

```
int  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:

| | |
|---|---|
| [ENOTEMPTY] | The named directory contains files other than "." and ".." in it. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOTDIR] | A component of the path prefix is not a directory. |
| [ENOENT] | The named file does not exist. |
| [EACCES] | A component of the path prefix denies search permission. |
| [EACCES] | Write permission is denied on the directory containing the link to be removed. |
| [EBUSY] | The directory to be removed is the mount point for a mounted file system. |
| [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**

rmdir(1), mkdir(2), unlink(2).

## NAME

select – synchronous I/O multiplexing

## SYNOPSIS

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

int select(nfds, readfds, writefds, execptfds, timeout)
int nfds, *readfds, *writefds, *execptfds;
struct timeval *timeout;
```

## DESCRIPTION

select examines the I/O descriptors specified by the bit masks *readfds*, *writefds*, and *execptfds* to see if they are ready for reading, writing, or have an exceptional condition pending, respectively. File descriptor *f* is represented by the bit 1<<*f* in the mask. *nfds* descriptors are checked, i.e., the bits from 0 through *nfds*−1 in the masks are examined. select returns, in place, a mask of those descriptors which are ready. The total number of ready descriptors is returned.

If *timeout* is a nonzero 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 nonzero, pointing to a zero valued *timeval* structure.

Any of *readfds*, *writefds*, and *execptfds* may be given as 0 if no descriptors are of interest.

## RETURN VALUE

select returns the number of descriptors which are contained in the bit masks, or −1 if an error occurred. If the time limit expires then select returns 0.

## ERRORS

An error return from select indicates:

| | |
|---|---|
| [EBADF] | One of the bit masks specified an invalid descriptor. |
| [EINTR] | A signal was delivered before any of the selected for events occurred or the time limit expired. |

## SEE ALSO

accept(2N), connect(2N), recv(2N), readv(2), send(2N), writev(2).

**BUGS**

The descriptor masks are always modified on return, even if the call returns as the result of the timeout.

## NAME

semctl – semaphore control operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl(semid, semnum, cmd, arg)
int semid, cmd;
int semnum;
union semun {
        int val;
        struct semid_ds *buf;
        ushort *array;
} arg;
```

## DESCRIPTION

semctl provides a variety of semaphore control operations as specified by *cmd*.

The following *cmd*s are executed with respect to the semaphore specified by *semid* and *semnum* (see intro(2) for required permissions and structure declarations):

GETVAL Return the value of semval (see intro(2)).

SETVAL Set the value of semval to *arg.val*. When this command is successfully executed, the semadj value corresponding to the specified semaphore in all processes is cleared.

GETPID Return the value of sempid.

GETNCNT Return the value of semncnt.

GETZCNT Return the value of semzcnt.

The following *cmd*s return and set, respectively, every semval in the set of semaphores.

GETALL Place semvals into array pointed to by *arg.array*.

SETALL Set semvals according to the array pointed to by *arg.array*. When this command is successfully executed, the semadj values corresponding to each specified semaphore in all processes are cleared.

The following *cmd*s are also available:

IPC_STAT Place the current value of each member of the data structure associated with *semid* into the structure pointed to by *arg.buf*. The contents of this structure are defined in intro(2).

IPC_SET Set the value of the following members of the data structure associated with *semid* to the corresponding value found in the structure pointed to by *arg.buf*:

```
sem_perm.uid
sem_perm.gid
sem_perm.mode /* only low 9 bits */
```

This command can only be executed by a process that has an effective user ID equal to either that of superuser or to the value of sem_perm.uid in the data structure associated with *semid*.

IPC_RMID Remove the semaphore identifier specified by *semid* from the system and destroy the set of semaphores and data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of superuser or to the value of sem_perm.uid in the data structure associated with *semid*. The identifier and its associated data structure are not actually removed until there are no more referencing processes. See ipcrm(1), and ipcs(1).

## RETURN VALUE

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

| | |
|---|---|
| GETVAL | The value of semval. |
| GETPID | The value of sempid. |
| GETNCNT | The value of semncnt. |
| GETZCNT | The value of semzcnt. |
| All others | A value of 0. |

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

## ERRORS

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

[EINVAL] *semid* is not a valid semaphore identifier.

| | |
|---|---|
| [EINVAL] | *semnum* is less than zero or greater than sem_nsems. |
| [EINVAL] | *cmd* is not a valid command. |
| [EACCES] | Operation permission is denied to the calling process (see intro(2)). |
| [ERANGE] | *cmd* is SETVAL or SETALL and the value to which semval is to be set is greater than the system imposed maximum. |
| [EPERM] | *cmd* is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of superuser and it is not equal to the value of sem_perm.uid in the data structure associated with *semid*. |
| [EFAULT] | *arg.buf* points to an illegal address. |

**SEE ALSO**

intro(2), semget(2), semop(2).

## NAME

semget – get set of semaphores

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget(key, nsems, semflg)
key_t key;
int nsems, semflg;
```

## DESCRIPTION

semget returns the semaphore identifier associated with *key*.

A semaphore identifier and associated data structure and set containing *nsems* semaphores (see intro(2)) are created for *key* if one of the following are true:

*key* is equal to IPC_PRIVATE.

*key* does not already have a semaphore identifier associated with it, and (*semflg* & IPC_CREAT) is "true".

The key IPC_PRIVATE will create an identifier and associated data structure that is unique to the calling process and its children.

Upon creation, the data structure associated with the new semaphore identifier is initialized as follows:

sem_perm.cuid, sem_perm.uid, sem_perm.cgid, and sem_perm.gid are set equal to the effective user ID and effective group ID, respectively, of the calling process.

The low-order 9 bits of sem_perm.mode are set equal to the low-order 9 bits of *semflg*.

sem_nsems is set equal to the value of *nsems*.

sem_otime is set equal to 0 and sem_ctime is set equal to the current time.

## RETURN VALUE

Upon successful completion, a non-negative integer, namely a semaphore identifier, is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

## ERRORS

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

[EINVAL] *nsems* is either less than or equal to zero or greater than the system-imposed limit.

[EACCES] A semaphore identifier exists for *key*, but operation permission (see intro(2)) as specified by the low-order 9 bits of *semflg* would not be granted.

[EINVAL] A semaphore identifier exists for *key*, but the number of semaphores in the set associated with it is less than *nsems* and *nsems* is not equal to zero.

[ENOENT] A semaphore identifier does not exist for *key* and (*semflg* & IPC_CREAT) is "false".

[ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphore identifiers system wide would be exceeded.

[ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores system wide would be exceeded.

[EEXIST] A semaphore identifier exists for *key* but ((*semflg* & IPC_CREAT) && (*semflg* & IPC_EXCL)) is "true".

**SEE ALSO**

intro(2), semctl(2), semop(2).

## NAME

semop – semaphore operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop(semid, sops, nsops)
int semid;
struct sembuf **sops;
int nsops;
```

## DESCRIPTION

semop is used to automatically perform an array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by *semid*. *sops* is a pointer to the array of semaphore-operation structures. *nsops* is the number of such structures in the array. The contents of each structure includes the following members:

```
short  sem_num;  /* semaphore number */
short  sem_op;   /* semaphore operation */
short  sem_flg;  /* operation flags */
```

Each semaphore operation specified by sem_op is performed on the corresponding semaphore specified by *semid* and sem_num.

sem_op specifies one of three semaphore operations as follows (see intro(2) for permissions and structure declarations:

If sem_op is a negative integer, one of the following will occur:

If semval (see intro(2)) is greater than or equal to the absolute value of sem_op, the absolute value of sem_op is subtracted from semval. Also, if (sem_flg & SEM_UNDO) is "true", the absolute value of sem_op is added to the calling process's semadj value (see exit(2)) for the specified semaphore.

If semval is less than the absolute value of sem_op and (sem_flg & IPC_NOWAIT) is "true", semop will return immediately.

If semval is less than the absolute value of sem_op and (sem_flg & IPC_NOWAIT) is "false", semop will increment the semncnt associated with the specified semaphore and suspend execution of the calling process until one of the following conditions occur:

semval becomes greater than or equal to the absolute value of sem_op. When this occurs, the value of semncnt associated with the specified semaphore is decremented, the absolute value of sem_op is subtracted from semval and, if (sem_flg & SEM_UNDO) is "true", the absolute value of sem_op is added to the calling process's semadj value for the specified semaphore.

The *semid* for which the calling process is awaiting action is removed from the system (see semctl(2)). When this occurs, errno is set equal to EIDRM, and a value of −1 is returned.

The calling process receives a signal that is to be caught. When this occurs, the value of semncnt associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in signal(3).

If sem_op is a positive integer, the value of sem_op is added to semval and, if (sem_flg & SEM_UNDO) is "true", the value of sem_op is subtracted from the calling process's semadj value for the specified semaphore.

If sem_op is zero, one of the following will occur:

If semval is zero, semop will return immediately.

If semval is not equal to zero and (sem_flg & IPC_NOWAIT) is "true", semop will return immediately.

If semval is not equal to zero and (sem_flg & IPC_NOWAIT) is "false", semop will increment the semzcnt associated with the specified semaphore and suspend execution of the calling process until one of the following occurs:

semval becomes zero, at which time the value of *semzcnt* associated with the specified semaphore is decremented.

The *semid* for which the calling process is awaiting action is removed from the system. When this occurs, errno is set equal to EIDRM, and a value of −1 is returned.

The calling process receives a signal that is to be caught. When this occurs, the value of semzcnt associated

with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in `signal`(3).

## RETURN VALUE

If `semop` returns due to the receipt of a signal, a value of –1 is returned to the calling process and `errno` is set to `EINTR`. If it returns due to the removal of a *semid* from the system, a value of –1 is returned and `errno` is set to `EIDRM`.

Upon successful completion, the value of `semval` at the time of the call for the last operation in the array pointed to by *sops* is returned. Otherwise, a value of –1 is returned and `errno` is set to indicate the error.

## ERRORS

`semop` will fail if one or more of the following are true for any of the semaphore operations specified by *sops*:

| | |
|---|---|
| `[EINVAL]` | *semid* is not a valid semaphore identifier. |
| `[EFBIG]` | `sem_num` is less than zero or greater than or equal to the number of semaphores in the set associated with *semid*. |
| `[E2BIG]` | *nsops* is greater than the system-imposed maximum. |
| `[EACCES]` | Operation permission is denied to the calling process (see `intro`(2)). |
| `[EAGAIN]` | The operation would result in suspension of the calling process but `(sem_flg & IPC_NOWAIT)` is "true". |
| `[ENOSPC]` | The limit on the number of individual processes requesting an `SEM_UNDO` would be exceeded. |
| `[EINVAL]` | The number of individual semaphores for which the calling process requests a `SEM_UNDO` would exceed the limit. |
| `[ERANGE]` | An operation would cause a `semval` to overflow the system-imposed limit. |
| `[ERANGE]` | An operation would cause a `semadj` value to overflow the system-imposed limit. |
| `[EFAULT]` | *sops* points to an illegal address. |

Upon successful completion, the value of *semid* for each semaphore specified in the array pointed to by *sops* is set equal to the process ID of the calling process.

**SEE ALSO**

exec(2), exit(2), fork(2), intro(2), semctl(2), semget(2).

## NAME

send, sendto, sendmsg – send a message from a socket

## SYNOPSIS

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

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

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

int sendmsg(s, msg, flags)
int 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 (i.e., when connect(2N) has been used), 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.

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

The *flags* parameter may be set to MSG_OOB to send "out-of-band" data on sockets which support this notion (e.g. SOCK_STREAM).

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

## RETURN VALUE

The call returns the number of characters sent, or −1 if an error

occurred.

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

**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 nonblocking and the requested operation would block. |

**SEE ALSO**

`connect`(2N), `recv`(2N), `socket`(2N).

## NAME

setcompat, getcompat – set or get process compatibility mode

## SYNOPSIS

```
#include <compat.h>

int setcompat(flags)
int flags;

int getcompat()
```

## DESCRIPTION

setcompat sets a process's compatibility mode according to the *flags* argument. *flags* governs the type of compatibility enforced. *flags* may be COMPAT_SVID for strictest adherence to the System V interface definition or the bitwise *or* of one or more of the following symbolic constants. If set, other flags always take precedence over COMPAT_SVID.

| | |
|---|---|
| COMPAT_BSDNBIO | Changes the error handling in 4.2 BSD nonblocking I/O code. Read and write system calls on slow devices, i.e., terminals, which are marked for non-blocking may return –1 with errno set to EWOULDBLOCK instead of returning 0. (Operations which may block, i.e., connect, accept and recv, on sockets which are marked for non-blocking always return an error and set errno to EWOULDBLOCK.) |
| COMPAT_BSDPROT | Enables the use of the 4.2 BSD groups code which permits users to be members of more than one group simultaneously and creates files whose group is determined by the group of the directory in which the file is created. When selected, changes the behavior of the setuid and setgid calls to be BSD-compatible; i.e., no handling of the saved set-user (group) ID from exec. When cleared, the setreuid and setregid calls behave as setuid and setgid, respectively. |

| | |
|---|---|
| COMPAT_BSDSIGNALS | Allows a process to use 4.2 BSD-compatible signals. The state of this flag may not be changed unless no signals are pending, caught, or held. This option enables reliable signal delivery. Caught signals will be held while a signal handler is invoked, and reset upon exit from the signal hander. |
| COMPAT_BSDTTY | Enables 4.2 BSD job control. When first set, this process and its descendants will be identified as 4.2 processes via a bit in the flag word of the kernel `proc` data structure. Membership in a 4.2 process group persists across `exec` system calls. Jobs that are 4.2 process group members are effected by job control signals. When COMPAT_BSDTTY is set the `setpgrp` system call may be used to manipulate the process group of other processes. This flag may only be used in conjunction with the COMPAT_BSDSIGNALS flag. Normally COMPAT_BSDTTY is set by a login shell. |
| COMPAT_CLRPGROUP | Disables 4.2 BSD job control. Resets the 4.2 process group bit in the flag word of the kernel `proc` data structure. It may be used by a V.2 process which wants to sever any job control associations with an invoking shell (for itself and its descendants). This bit provides a "one shot" clear. When read by `getcompat`, this bit is always zero. |
| COMPAT_EXEC | If this flag is set, compatability flags are inherited across exec system calls. To provide child process with a System V interface environment, both COMPAT_SVID and COMPAT_EXEC flags must be set by |

ORing the flags.

COMPAT_SYSCALLS — If selected, `read`, `write`, `ioctl`, or `wait` calls which are interrupted by a signal handler will not return an `EINTR` error, but will instead resume at the point they were interrupted. This flag may only be used in conjunction with the `COMPAT_BSDSIGNALS` flag.

`getcompat` returns the current process compatibility flags. By default, compatibility flags are preserved across `forks` and are reset by `execs` (see `COMPAT_EXEC` above).

The default process compatibility flags are `COMPAT_BSDPROT` and `COMPAT_BSDNBIO`.

## RETURN VALUE

Upon successful completion, `setcompat` returns the previous compatibility mask and `getcompat` returns the current compatibility mask. Otherwise, a value of −1 is returned and `errno` is set to indicate the error.

## ERRORS

`setcompat` will return the following error code:

[EINVAL] — *flag* results in a change in the state of the `COMPAT_BSDSIGNALS` bit and a signal is currently pending, caught, or held.

[EINVAL] — *flag* is either `COMPAT_BSDTTY` or `COMPAT_SYSCALLS` and `COMPAT_BSDSIGNALS` is not also set.

## SEE ALSO

`exec`(2), `fork`(2), `sigvec`(2), `set42sig`(3), `signal`(3), `setuid`(3), `termio`(7).

**NAME**

setgroups – set group access list

**SYNOPSIS**

```
#include <sys/param.h>

int 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 superuser 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:

| | |
|---|---|
| [EINVAL] | The value of *ngroups* is greater than NGROUPS. |
| [EPERM] | The caller is not the superuser. |
| [EFAULT] | The address specified for *gidset* is outside the process address space. |

**SEE ALSO**

getgroups(2), initgroups(3X).

**NAME**

setpgrp – set process group ID

**SYNOPSIS**

```
int  setpgrp()
```

or

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

**DESCRIPTION**

The first form of `setpgrp` sets the process group ID of the calling process to the process ID of the calling process and returns the new process group ID.

The second form of `setpgrp` is available when the process has requested 4.2 BSD compatibility. `setpgrp` will then set 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 user is not superuser, then the affected process must have the same effective user ID as the invoking user or be a descendant of the invoking process.

**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 `setpgrp` call fails if:

`[ESRCH]` the process is not found.

`[EPERM]` The caller is not superuser.

**SEE ALSO**

`exec`(2), `fork`(2), `getpid`(2), `intro`(2), `kill`(2), `setcompat`(2), `signal`(3).

**NAME**

setregid – set real and effective group ID

**SYNOPSIS**

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

**DESCRIPTION**

The real and effective group ID's of the current process are set to the arguments. Only the superuser may change the real group ID of a process. Unprivileged users may change the effective group ID to the real group ID, but to no other.

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 superuser and a change other than changing the effective group ID to the real group ID was specified.

**NOTE**

This call only works in COMPAT_BSDPROT compatibility mode.

**SEE ALSO**

getgid(2), setcompat(2), setreuid(2), setgid(3).

## NAME

setreuid – set real and effective user ID's

## SYNOPSIS

```
int 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. Only the superuser may modify the real uid of a process. Users other than the superuser may change the effective uid of a process only to the real user ID.

## 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 superuser and a change other than changing the effective user ID to the real user ID was specified. |

## NOTE

This call only works in COMPAT_BSDPROT compatibility mode.

## SEE ALSO

getuid(2), setcompat(2), setregid(2), setuid(2).

## NAME

setuid, setgid – set user and group IDs

## SYNOPSIS

```
int  setuid(uid)
int  uid;

int  setgid(gid)
int  gid;
```

## DESCRIPTION

setuid (setgid) sets the real user (group) ID and effective user (group) ID of the calling process.

If the effective user ID of the calling process is superuser, the real user (group) ID and effective user (group) ID are set to *uid* (*gid*).

If the effective user ID of the calling process is not superuser, but its real user (group) ID is equal to *uid* (*gid*), the effective user (group) ID is set to *uid* (*gid*).

If the effective user ID of the calling process is not superuser, but the saved set-user (group) ID from exec(2) is equal to *uid* (*gid*), the effective user (group) ID is set to *uid* (*gid*).

## ERRORS

setuid (setgid) will fail if one or more of the following are true:

| | |
|---|---|
| [EPERM] | the real user (group) ID of the calling process is not equal to *uid* (*gid*) and its effective user ID is not superuser. |
| [EINVAL] | The *uid* is out of range. |

## 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.

## SEE ALSO

getuid(2), setregid(2), setreuid(2), intro(2).

**NAME**

shmctl – shared memory control operations

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl(shmid, cmd, buf)
int shmid, cmd;
struct shmid_ds *buf;
```

**DESCRIPTION**

shmctl provides a variety of shared memory control operations as specified by *cmd*. (Structure definitions and permissions are described in intro(2).) The following *cmds* are available:

IPC_STAT — Place the current value of each member of the data structure associated with *shmid* into the structure pointed to by *buf*.

IPC_SET — Set the value of the following members of the data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:

```
shm_perm.uid
shm_perm.gid
shm_perm.mode /*only low 9 bits*/
```

This *cmd* can only be executed by a process that has an effective user ID equal to either that of superuser or to the value of shm_perm.uid in the data structure associated with *shmid*.

IPC_RMID — Remove the shared memory identifier specified by *shmid* from the system and destroy the shared memory segment and data structure associated with it. This *cmd* can only be executed by a process that has an effective user ID equal to either that of superuser or to the value of shm_perm.uid in the data structure associated with *shmid*. The identifier and its associated data structure are not actually removed until there are no more referencing processes. See ipcrm(1), and ipcs(1).

**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**

`shmctl` will fail if one or more of the following are true:

| | |
|---|---|
| `[EINVAL]` | *shmid* is not a valid shared memory identifier. |
| `[EINVAL]` | *cmd* is not a valid command. |
| `[EACCES]` | *cmd* is equal to `IPC_STAT` and READ operation permission is denied to the calling process (see `intro`(2)). |
| `[EAGAIN]` | The system has temporarily exhausted its available memory or swap space. |
| `[EPERM]` | *cmd* is equal to `IPC_RMID` or `IPC_SET` and the effective user ID of the calling process is not equal to that of superuser and it is not equal to the value of `shm_perm.uid` in the data structure associated with *shmid*. |
| `[EFAULT]` | *buf* points to an illegal address. |

**SEE ALSO**

`intro`(2), `shmget`(2), `shmop`(2).

## NAME

shmget – get shared memory segment

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget(key, size, shmflg)
key_t key;
int size, shmflg;
```

## DESCRIPTION

shmget returns the shared memory identifier associated with *key*.

A shared memory identifier and associated data structure and shared memory segment of at least size *size* bytes are created for *key* if one of the following are true (see intro(2)):

*key* is equal to IPC_PRIVATE.

*key* does not already have a shared memory identifier associated with it, and (shmflg & IPC_CREAT) is "true".

*Note:* A shared memory segment of *size* is always rounded up to the nearest whole page.

The key IPC_PRIVATE will crte an identifier and associated data structure that is unique to the calling gprocess and its children.

Upon creation, the data structure associated with the new shared memory identifier is initialized as follows:

shm_perm.cuid, shm_perm.uid, shm_perm.cgid, and shm_perm.gid are set equal to the effective user ID and effective group ID, respectively, of the calling process.

The low-order 9 bits of shm_perm.mode are set equal to the low-order 9 bits of *shmflg*. shm_segsz is set equal to the value of *size*.

shm_lpid, shm_nattch, shm_atime, and shm_dtime are set equal to 0.

shm_ctime is set equal to the current time.

## RETURN VALUE

Upon successful completion, a non-negative integer, namely a shared memory identifier is returned. Otherwise, a value of –1 is returned and errno is set to indicate the error.

**ERRORS**

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

[EINVAL] *size* is less than the system-imposed minimum or greater than the system-imposed maximum.

[EACCES] A shared memory identifier exists for *key* but operation permission (see intro(2)) as specified by the low-order 9 bits of *shmflg* would not be granted.

[EAGAIN] The system has temporarily exhausted its available memory or swap space.

[EINVAL] A shared memory identifier exists for *key* but the size of the segment associated with it is less than *size* and *size* is not equal to zero.

[ENOENT] A shared memory identifier does not exist for *key* and (*shmflg* & IPC_CREAT) is "false".

[ENOSPC] A shared memory identifier is to be created but the system-imposed limit on the maximum number of allowed shared memory identifiers system wide would be exceeded.

[ENOMEM] A shared memory identifier and associated shared memory segment are to be created but the amount of available physical memory is not sufficient to fill the request.

[EEXIST] A shared memory identifier exists for *key* but ((*shmflg* & IPC_CREAT) && (*shmflg* & IPC_EXCL)) is "true".

**SEE ALSO**

intro(2), shmctl(2), shmop(2).

## NAME

shmop, shmat, shmdt – shared memory operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

char *shmat(shmid, shmaddr, shmflg)
int shmid;
char *shmaddr
int shmflg;

int shmdt(shmaddr)
char *shmaddr
```

## DESCRIPTION

shmat attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the data segment of the calling process. The segment is attached at the address specified by one of the following criteria:

If *shmaddr* is equal to zero, the segment is attached at the first available address as selected by the system.

If *shmaddr* is not equal to zero and (*shmflg* & SHM_RND) is "true", the segment is attached at the address given by (*shmaddr* - (*shmaddr* modulus SHMLBA)).

If *shmaddr* is not equal to zero and (*shmflg* & SHM_RND) is "false," the segment is attached at the address given by *shmaddr*.

The segment is attached for reading if (*shmflg* & SHM_RDONLY) is "true", otherwise it is attached for reading and writing.

## RETURN VALUES

Upon successful completion, the return value is as follows:

shmat returns the data segment start address of the attached shared memory segment.

shmdt returns a value of 0.

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

## ERRORS

shmat will fail and not attach the shared memory segment if one or more of the following are true:

| | |
|---|---|
| [EINVAL] | *shmid* is not a valid shared memory identifier. |
| [EACCES] | Operation permission is denied to the calling process (see intro(2)). |
| [EAGAIN] | The system has temporarily exhausted its available memory or swap space. |
| [ENOMEM] | The available data space is not large enough to accommodate the shared memory segment. |
| [EINVAL] | *shmaddr* is not equal to zero, and the value of (*shmaddr* - (*shmaddr* modulus SHMLBA)) is an illegal address. |
| [EINVAL] | *shmaddr* is not equal to zero, (*shmflg* & SHM_RND) is "false", and the value of *shmaddr* is an illegal address. |
| [EMFILE] | The number of shared memory segments attached to the calling process would exceed the system-imposed limit. |
| [EINVAL] | shmdt detaches from the calling process's data segment the shared memory segment located at the address specified by *shmaddr*. |
| [EINVAL] | shmdt will fail and not detach the shared memory segment if *shmaddr* is not the data segment start address of a shared memory segment. |

**SEE ALSO**

exec(2), exit(2), fork(2), intro(2), shmctl(2), shmget(2).

## NAME

shutdown – shut down part of a full-duplex connection

## SYNOPSIS

```
int 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.

## RETURN VALUE

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(2N), socket(2N).

**NAME**

sigblock – block signals

**SYNOPSIS**

```
#include <signal.h>

int sigblock(mask);
int mask;

sigmask(signum)
int 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), signal(3).

## NAME

sigpause – atomically release blocked signals and wait for interrupt

## SYNOPSIS

```
int  sigpause (sigmask)
int  sigmask;
```

## DESCRIPTION

sigpause assigns *sigmask* to the set of blocked 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.

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.

## RETURN VALUE

sigpause always terminates by being interrupted, returning –1.

## ERRORS

sigpause always terminates by being interrupted with errno set to EINTR.

## SEE ALSO

sigblock(2), sigvec(2), signal(3).

**NAME**

sigsetmask – set current signal mask

**SYNOPSIS**

```
#include <signal.h>

int sigsetmask(mask);
int mask;

int signum;
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), `signal`(3).

## NAME

sigstack – set and/or get signal stack context

## SYNOPSIS

```
#include <signal.h>

struct sigstack {
    caddr_t ss_sp;
    int ss_onstack;
};

int 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 nonzero, 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 nonzero, 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), signal(3).

## NAME

sigvec – optional BSD-compatible software signal facilities

## SYNOPSIS

```
#include <signal.h>

struct sigvec {
    int (*sv_handler)();
    int sv_mask;
    int sv_onstack;
};

int 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 nonzero, it specifies a handler routine and mask to be used when delivering the specified signal. Further, if `sv_onstack` is one, the system will deliver the signal to the process on a "signal stack," specified with `sigstack`(2). If *ovec* is nonzero, the previous handling information for the signal is returned to the user.

The following is a list of the A/UX signals with names as in the include file <`signal.h`>:

| | | |
|---|---|---|
| `SIGHUP` | 1 | hangup |
| `SIGINT` | 2 | interrupt |
| `SIGQUIT` | 3* | quit |
| `SIGILL` | 4* | illegal instruction |
| `SIGTRAP` | 5* | trace trap |
| `SIGIOT` | 6* | IOT instruction |
| `SIGEMT` | 7* | EMT instruction |
| `SIGFPE` | 8* | floating point exception |
| `SIGKILL` | 9 | kill (cannot be caught, blocked, or 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 |
| `SIGUSR1` | 16 | user defined signal 1 |
| `SIGUSR2` | 17 | user defined signal 2 |
| `SIGCLD` | 18• | child status has changed |
| `SIGPWR` | 19 | power-fail restart |
| `SIGTSTP` | 20† | stop signal generated from keyboard |
| `SIGTTIN` | 21† | background read attempted from control terminal |
| `SIGTTOU` | 22† | background write attempted to control terminal |
| `SIGSTOP` | 23† | stop (cannot be caught, blocked, or ignored) |
| `SIGXCPU` | 24 | cpu time limit exceeded |
| `SIGXFSZ` | 25 | file size limit exceeded |
| `SIGVTALRM` | 26 | virtual time alarm (see `setitimer`(2)) |
| `SIGPROF` | 27 | profiling timer alarm (see `setitimer`(2)) |
| `SIGWINCH` | 28• | window size change |

| | | |
|---|---|---|
| SIGCONT | 29• | continue after stop (cannot be blocked) |
| SIGURG | 30• | urgent condition present on socket |
| SIGIO | 31• | I/O is possible on a descriptor (see fcntl(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 the process is part of a 4.2 job control group. They are ignored when using 5.2 signals. 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 affect 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). This behavior may be modified by options suplied to the setcompat(2) system call.

After a fork(2), the child inherits all signals, the signal mask, and the signal stack.

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; the signal handler reverts to the 5.2 signal mechanism.

## NOTES

The mask specified in *vec* is not allowed to block SIGKILL, SIGSTOP, or SIGCONT. This is done silently by the system.

## 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:

| | |
|---|---|
| [EFAULT] | Either *vec* or *ovec* points to memory that is not a valid part of the process address space. |

| | |
|---|---|
| [EINVAL] | *sig* is not a valid signal number. |
| [EINVAL] | An attempt is made to ignore or supply a handler for SIGKILL or SIGSTOP. |
| [EINVAL] | An attempt is made to ignore SIGCONT (by default SIGCONT is ignored). |

**SEE ALSO**

kill(1), ptrace(2), kill(2), sigblock(2), setcompat(2), sigsetmask(2), sigpause(2), sigstack(2), set42sig(3), signal(3), termio(7).

**NAME**

socket – create an endpoint for communication

**SYNOPSIS**

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

int socket(af, type, protocol)
int af, type, protocol;
```

**DESCRIPTION**

socket creates an endpoint for communication and returns a descriptor.

The *af* parameter specifies an address format with which addresses specified in later operations using the socket should be interpreted. These formats are defined in the include file <sys/socket.h>. The currently understood formats are:

| | |
|---|---|
| AF_UNIX | (UNIX path names) |
| AF_INET | (ARPA Internet addresses) |
| AF_PUP | (Xerox PUP-I Internet addresses) |
| AF_IMPLINK | (IMP "host at IMP" addresses) |

*Note:* The only address format currently supported on this implementation is AF_INET.

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 with an out-of-band data transmission mechanism. A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). SOCK_RAW sockets provide access to internal network interfaces. The types SOCK_RAW, which is available only to the superuser, and SOCK_SEQPACKET and SOCK_RDM, which are 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 using a given address format. 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 services(4N) and protocols(4N).

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(2N) call. Once connected, data may be transferred using read(2) and write(2) calls or some variant of the send(2N) and recv(2N) 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(2N) and received as described in recv(2N).

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_DGRAM and SOCK_RAW sockets allow sending of datagrams to correspondents named in send(2N) calls. It is also possible to receive datagrams at such a socket with recv(2N).

An fcntl(2) call can be used to specify a process group to receive a SIGURG signal when the out-of-band data arrives.

The operation of sockets is controlled by socket level *options*. These options are defined in the file <sys/socket.h> and explained below. setsockopt and getsockopt(2N) are used to set and get options, respectively.

| | |
|---|---|
| SO_DEBUG | turn on recording of debugging information |
| SO_REUSEADDR | allow local address reuse |

| | |
|---|---|
| SO_KEEPALIVE | keep connections alive |
| SO_DONTROUTE | do no apply routing on outgoing messages |
| SO_LINGER | linger on close if data present |
| SO_DONTLINGER | do not linger on close |

SO_DEBUG enables debugging in the underlying protocol modules. SO_REUSEADDR indicates that the rules used in validating addresses supplied in a bind(2N) 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 and SO_DONTLINGER control the actions 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_DONTLINGER is specified and a close is issued, the system will process the close in a manner which allows the process to continue as quickly as possible.

**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:

| | |
|---|---|
| [EAFNOSUPPORT] | The specified address family is not supported in this version of the system. |
| [ESOCKTNOSUPPORT] | The specified socket type is not supported in this address family. |
| [EPROTONOSUPPORT] | The specified protocol is not supported. |

| | |
|---|---|
| [EMFILE] | The per-process descriptor table is full. |
| [ENOBUFS] | No buffer space is available. The socket cannot be created. |

**SEE ALSO**

accept(2N), bind(2N), connect(2N), getsockname(2N), getsockopt(2N), ioctl(2), listen(2N), recv(2N), select(2N), send(2N), shutdown(2N).

**BUGS**

The use of keepalives is a questionable feature for this layer.

## NAME

stat, fstat, lstat – get file status

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(path, buf)
char *path;
struct stat *buf;

int fstat(fildes, buf)
int fildes;
struct stat *buf;

int lstat(path, buff)
char *path;
struct stat *buf;
```

## DESCRIPTION

stat obtains information about the named file. *path* points to a path name naming a file. 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 searchable.

lstat is like stat except in the case where the named file is a symbolic link, in which case lstat returns information about the link, while stat returns information about the file the link references.

Similarly, fstat obtains information about an open file known by the file descriptor *fildes*, obtained from a successful open, creat, dup, fcntl, or pipe system call.

*buf* is a pointer to a stat structure into which information is placed concerning the file.

The contents of the structure referenced by *buf* include the following members:

| | |
|---|---|
| ushort st_mode; | File mode; see stat(5) |
| ino_t st_ino; | Inode number |
| dev_t st_dev; | ID of device containing a directory entry for this file |
| dev_t st_rdev; | ID of device. This entry is defined only for character special or block special files |

| | |
|---|---|
| `short st_nlink;` | Number of links |
| `ushort st_uid;` | User ID of the file's owner |
| `ushort st_gid;` | Group ID of the file's group |
| `off_t st_size;` | File size in bytes |
| `time_t st_atime;` | Time when file data was last accessed (times measured in seconds since 00:00:00 GMT, Jan. 1, 1970). Changed by the following system calls: `creat`(2), `mknod`(2), `pipe`(2), `utime`(2), and `read`(2). |
| `time_t st_mtime;` | Time when data was last modified (times measured in seconds since 00:00:00 GMT, Jan. 1, 1970). Changed by the following system calls: `creat`(2), `mknod`(2), `pipe`(2), `utime`(2), and `write`(2). |
| `time_t st_ctime;` | Time wehn file status last changed (times measured in seconds since 00:00:00 GMT, Jan. 1, 1970) Changed by the following system calls: `chmod`(2), `chown`(2), `creat`(2), `link`(2), `mknod`(2), `pipe`(2), `unlink`(2), `utime`(2), and `write`(2). |
| `long st_blksize;` | optimal blocksize for I/O ops |
| `long st_blocks;` | actual number of blocks allocated |

**RETURN VALUE**

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

**ERRORS**

`stat` and `lstat` will fail if one or more of the following are true:

| | |
|---|---|
| `[ENOTDIR]` | A component of the path prefix is not a directory. |
| `[EPERM]` | A pathname contains a character with the high-order bit set. |

| | |
|---|---|
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOENT] | The named file does not exist. |
| [EACCES] | Search permission is denied for a component of the path prefix. |
| [EFAULT] | *buf* or *path* points to an invalid address. |

`fstat` will fail if one or more of the following are true:

| | |
|---|---|
| [EBADF] | *fildes* is not a valid open file descriptor. |
| [EFAULT] | *buf* points to an invalid address. |

**SEE ALSO**

`chmod`(2), `chown`(2), `creat`(2), `link`(2), `mknod`(2), `pipe`(2), `read`(2), `readlink`(2), `statfs`(2), `time`(2), `unlink`(2), `ustat`(2), `utime`(2), `write`(2), `stat`(5).

## NAME

statfs – get file system statistics

## SYNOPSIS

```
#include <sys/vfs.h>
#include <sys/types.h>

int statfs(path, buf)
char *path;
struct statfs *buf;

int fstatfs(fildes, buf)
int fildes;
struct statfs *buf;
```

## DESCRIPTION

`statfs` returns information about a mounted file system. *path* is the pathname of any file within the mounted file system. *buf* is a pointer to a `statfs` structure defined as follows:

```
typedef long fsid_t[2];

struct statfs {
    long    f_type;       /* type of info, zero
                             for now */
    long    f_bsize;      /* fundamental file system
                             block size */
    long    f_blocks;     /* total blocks in file
                             system */
    long    f_bfree;      /* free blocks */
    long    f_bavail;     /* free blocks available to
                             nonsuperuser */
    long    f_files;      /* total file nodes in
                             file system */
    long    f_ffree;      /* free file nodes in fs */
    fsid_t  f_fsid;       /* file system ID */
    long    f_spare[7];   /* spare for later */
};
```

Fields that are undefined for a particular file system are set to –1. `fstatfs` returns the same information about an open file referenced by descriptor *fildes*.

## 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.

**SEE ALSO**
stat(2), ustat(2).

**NAME**

stime – set time

**SYNOPSIS**

```
int  stime(tp)
long  *tp;
```

**DESCRIPTION**

stime sets the the time and date. *tp* points to the value of time as measured in seconds from 00:00:00 GMT January 1, 1970.

**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**

stime will fail if:

| | |
|---|---|
| [EPERM] | the effective user ID of the calling process is not superuser. |

**SEE ALSO**

date(1), gettimeofday(2), settimeofday(2), time(2).

**NAME**

symlink – make symbolic link to a file

**SYNOPSIS**

```
int  symlink(name1,  name2)
char  *name1,  *name2;
```

**DESCRIPTION**

A symbolic link *name2* is created to *name1* (*name2* is the name of the file created, *name1* is the string used in creating the symbolic link). Either name may be an arbitrary path name; the files need not be on the same file system.

**RETURN VALUE**

Upon successful completion, a zero value is returned. If an error occurs, the error code is stored in `errno` and a –1 value is returned.

**ERRORS**

The symbolic link is made unless on or more of the following are true:

| | |
|---|---|
| [EPERM] | Either *name1* or *name2* contains a character with the high-order bit set. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOENT] | One of the pathnames specified was too long. |
| [ENOTDIR] | A component of the *name2* prefix is not a directory. |
| [EEXIST] | *name2* already exists. |
| [EACCES] | A component of the *name2* path prefix denies search permission. |
| [EROFS] | The file *name2* would reside on a read-only file system. |
| [EFAULT] | *name1* or *name2* points outside the process's allocated address space. |

**SEE ALSO**

ln(1), link(2), readlink(2), unlink(2).

**NAME**

sync – update superblock

**SYNOPSIS**

```
void sync()
```

**DESCRIPTION**

The sync system call causes all information in memory that should be on disk to be written out. This includes modified superblocks, modified inodes, and delayed block I/O.

It should be used by programs which examine a file system, for example fsck, df, etc. It is mandatory before a reboot or a system shutdown.

The writing, although scheduled, is not necessarily complete upon return from sync.

**SEE ALSO**

sync(1), fsync(2).

**NAME**

time – get time

**SYNOPSIS**

```
long  time((long*)0)

long  time(tloc)
long  *tloc;
```

**DESCRIPTION**

time returns the value of time in seconds since 00:00:00 GMT, January 1, 1970.

If *tloc* (taken as an integer) is nonzero, the return value is also stored in the location to which *tloc* points.

**RETURN VALUE**

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

**ERRORS**

time will fail if

[EFAULT] *tloc* points to an illegal address.

**SEE ALSO**

date(1), gettimeofday(2), stime(2), ctime(3).

## NAME

times – get process and child process times

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/times.h>

long times(buffer)
struct tms *buffer;
```

## DESCRIPTION

times fills the structure pointed to by *buffer* with time-accounting information. The following are the contents of this structure:

```
struct tms {
        time_t    tms_utime;
        time_t    tms_stime;
        time_t    tms_cutime;
        time_t    tms_cstime;
};
```

This information comes from the calling process and each of its terminated child processes for which it has executed a wait. All times are in 60ths of a second.

| | |
|---|---|
| tms_utime | CPU time used while executing instructions in the user space of the calling process. |
| tms_stime | CPU time used by the system on behalf of the calling process. |
| tms_cutime | sum of the tms_utimes and tms_cutimes of the child processes. |
| tms_cstime | sum of the tms_stimes and tms_cstimes of the child processes. |

## RETURN VALUE

Upon successful completion, times returns the elapsed real time, in 60ths of a second, since an arbitrary point in the past (e.g., system start-up time). This point does not change from one invocation of times to another. If times fails, a –1 is returned and errno is set to indicate the error.

## ERRORS

times will fail if

[EFAULT] *buffer* points to an illegal address.

**SEE ALSO**

exec(2), fork(2), time(2), wait(2).

## NAME

truncate, ftruncate – truncate a file to a specified length

## SYNOPSIS

```
int truncate(path, length)
char *path;
int length;

int ftruncate(fd, length)
int fd, 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 will fail if:

| | |
|---|---|
| [EPERM] | The pathname contains a character with the high-order bit set. |
| [ENOENT] | The pathname was too long. |
| [ENOTDIR] | A component of the path prefix of *path* is not a directory. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOENT] | The named file does not exist. |
| [EACCES] | A component of the *path* prefix denies search permission. |
| [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.

*Note:* If you are running an NFS system and you are accessing a shared binary remotely, it is possible that you will not get this errno.

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

ftruncate will fail if:

[EBADF] The *fd* is not a valid descriptor.

[EINVAL] The *fd* references a socket, not a file.

**SEE ALSO**

open(2).

**BUGS**

Partial blocks discarded as the result of truncation are not zero filled; this can result in holes in files which do not read as zero.

These calls should be generalized to allow ranges of bytes in a file to be discarded.

**NAME**

ulimit – get and set user limits

**SYNOPSIS**

```
long ulimit(cmd, newlimit)
int cmd;
long newlimit;
```

**DESCRIPTION**

This function provides for control over process limits. The *cmd* values available are:

1 Get the file size limit of the process. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read.

2 Set the file size limit of the process to the value of *newlimit*. Any process may decrease this limit, but only a process with an effective user ID of superuser may increase the limit.

3 Get the maximum possible break value. See brk(2).

**RETURN VALUE**

Upon successful completion, a non-negative value is returned. Otherwise, a value of –1 is returned and errno is set to indicate the error.

**ERRORS**

ulimit will fail and the limit will be unchanged if the following is true:

[EPERM] a process with an effective user ID other than superuser attempts to increase its file size limit.

**SEE ALSO**

brk(2), write(2).

## NAME

umask – set and get file creation mask

## SYNOPSIS

```
int  umask(cmask)
int  cmask;
```

## DESCRIPTION

umask sets the calling process's file mode creation mask to *cmask* and returns the previous value of the mask. Only the low-order 9 bits of *cmask* and the file mode creation mask are used.

The file mode creation mask is used whenever a file is created by creat(2), mknod(2) or open(2). The actual mode (see chmod(2)) of the newly-created file is the difference between the given mode and *cmask*. In other words, *cmask* shows the bits to be turned off when a new file is created.

The previous value of *cmask* is returned by the call. The value is initially 022, which is an octal "mask" number representing the complement of the desired mode. "022" here means that no permissions are withheld from the owner, but write permission is forbidden to group and to others. Its complement, the mode of the file, would be 0755. The file mode creation mask is inherited by child processes.

## RETURN VALUE

The previous value of the file mode creation mask is returned.

## SEE ALSO

csh(1), ksh(1), chmod(1), mkdir(1), sh(1), chmod(2), creat(2), mknod(2), open(2).

**NAME**

umount – unmount a file system

**SYNOPSIS**

```
int  umount(spec)
char  *spec;
```

**DESCRIPTION**

umount is used to unmount System V file systems only. unmount is used to unmount all others (see unmount(2)).

umount requests that a previously mounted file system contained on the block special device identified by *spec* be unmounted. *spec* is a pointer to a path name. After unmounting the file system, the directory upon which the file system was mounted reverts to its ordinary interpretation.

umount may be invoked only by the superuser.

**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**

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

| | |
|---|---|
| [EPERM] | The process's effective user ID is not superuser. |
| [ENXIO] | *spec* does not exist. |
| [ENOTBLK] | *spec* is not a block special device. |
| [EINVAL] | *spec* is not mounted. |
| [EBUSY] | A file on *spec* is busy. |
| [EFAULT] | *spec* points to an illegal address. |

**SEE ALSO**

unmount(2), mount(3).

## NAME

uname – get name of current system

## SYNOPSIS

```
#include <sys/utsname.h>

int uname(name)
struct utsname *name;
```

## DESCRIPTION

uname stores information identifying the current system in the structure referenced by *name*.

uname uses the structure defined in <sys/utsname.h>:

```
struct      utsname {
     char   sysname[9];
     char   nodename[9];
     char   release[9];
     char   version[9];
     char   machine[9];
};
extern struct utsname utsname;
```

uname returns a null-terminated character string naming the current system in the character array sysname. Similarly, nodename contains the name by which the system is known on a communications network. release and version further identify the operating system. machine contains a standard name that identifies the hardware that the system is running on.

## RETURN VALUE

Upon successful completion, a non-negative value is returned. Otherwise, –1 is returned and errno is set to indicate the error.

## ERRORS

uname will fail if the following is true:

[EFAULT] *name* points to an invalid address.

## SEE ALSO

uname(1).

## NAME

unlink – remove directory entry

## SYNOPSIS

```
int  unlink (path)
char  *path;
```

## DESCRIPTION

unlink removes the directory entry named by the path name referenced by *path*.

When all links to a file have been removed and no process has the file open, the space occupied by the file is freed and the file ceases to exist. If one or more processes have the file open when the last link is removed, the removal is postponed until all references to the file have been closed.

## 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 named file is unlinked unless one or more of the following are true:

| | |
|---|---|
| [ENOTDIR] | A component of the path prefix is not a directory. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [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. |
| [EISDIR] | The named file is a directory. |
| [EBUSY] | The entry to be unlinked is the mount point for a mounted file system. |

[ETXTBSY] The entry to be unlinked is the last link to a pure procedure (shared text) file that is being executed.

*Note:* If you are running an NFS system and you are accessing a shared binary remotely, it is possible that you will not get this `errno`.

[EROFS] The directory entry to be unlinked is part of a read-only file system.

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

**SEE ALSO**

rm(1), `rmdir`(1), `close`(2), `link`(2), `open`(2), `rmdir`(2).

## NAME

unmount – remove a file system

## SYNOPSIS

```
unmount (name)
char *name;
```

## DESCRIPTION

unmount is used to unmount all non-System V file systems. umount is used to unmount System V file systems only (see umount(2)).

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:

| | |
|---|---|
| [EINVAL] | The caller is not the superuser. |
| [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. |

## SEE ALSO

fsmount(2), mount(3), umount(2).

## BUGS

The error codes are in a state of disarray; too many errors appear to the caller as one value.

## NAME

ustat – get file system statistics

## SYNOPSIS

```
#include <sys/types.h>
#include <ustat.h>

int ustat(dev, buf)
int dev;
struct ustat *buf;
```

## DESCRIPTION

ustat returns information about a mounted file system. *dev* is a device number identifying a device containing a mounted file system. *buf* is a pointer to a ustat structure that includes to following elements:

```
daddr_t f_tfree;        /* Total free blocks */
ino_t   f_tinode;       /* Number of free inodes */
char    f_fname[6];     /* Filsys name */
char    f_fpack[6];     /* Filsys pack name */
```

## 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

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

[EINVAL] *dev* is not the device number of a device containing a mounted file system.

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

## SEE ALSO

stat(2), statfs(2), fs(4).

## NAME

utime – set file access and modification times

## SYNOPSIS

```
#include <sys/types.h>
int utime(path, times)
char *path;
struct utimbuf *times;
```

## DESCRIPTION

utime sets the access and modification times of the named file. *path* points to a path name naming a file.

If *times* is NULL, the access and modification times of the file are set to the current time. A process must be the owner of the file or have write permission to use utime in this manner.

If times is not NULL, *times* is interpreted as a pointer to a utimbuf structure and the access and modification times are set to the values contained in the designated structure. Only the owner of the file or the superuser may use utime this way.

The times in the following structure are measured in seconds since 00:00:00 GMT, Jan. 1, 1970.

```
struct utimbuf {
        time_t actime;      /* access time */
        time_t modtime;     /* modification time */
};
```

## RETURN VALUE

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

## ERRORS

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

| | |
|---|---|
| [ENOENT] | The named file does not exist. |
| [EPERM] | A pathname contains a character with the high-order bit set. |
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [ENOTDIR] | A component of the path prefix is not a directory. |

| | |
|---|---|
| [EACCES] | Search permission is denied by a component of the path prefix. |
| [EPERM] | The effective user ID is not superuser and not the owner of the file and `times` is not NULL. |
| [EACCES] | The effective user ID is not superuser and not the owner of the file and `times` is NULL and write access is denied. |
| [EROFS] | The file system containing the file is mounted read-only. |
| [EFAULT] | *times* is not NULL and points outside the process's allocated address space. |
| [EFAULT] | *path* points outside the process's allocated address space. |

**SEE ALSO**

`stat`(2).

## NAME

uvar – returns system-specific configuration information

## SYNOPSIS

```
#include <sys/var.h>

int uvar(v)
struct var *v;
```

## DESCRIPTION

uvar returns system-specific configuration information contained in the kernel. The information returned contains table sizes, mask words, and other system-specific information for programs such as and ps(1).

Presently a maximum of 512 bytes of information is returned. *v* points to the var structure:

```
struct var {
        int     v_buf;          /* Number of system buffers */
        int     v_call;         /* Maximum number of
                                   simultaneous callouts */
        int     v_inode;        /* Maximum number of incore
                                   inodes */
        char*   ve_inode;       /* Pointer to last incore
                                   inode table  */
        int     v_file;         /* Maximum number of open
                                   files */
        char*   ve_file;        /* Pointer to last open
                                   file table */
        int     v_mount;        /* Maximum number of file
                                   systems mountable */
        char*   ve_mount;       /* Pointer to last mounted
                                   file system table */
        int     v_proc;         /* Maximum number of
                                   processes */
        char*   ve_proc;        /* Pointer to last process
                                   table  */
        int     v_text;         /* Maximum number of shared
                                   text segments */
        char*   ve_text;        /* Pointer to last shared
                                   text segment table  */
        int     v_clist;        /* Maximum number of clists */
        int     v_sabuf;        /* Maximum number of system
                                   activity buffers */
        int     v_maxup;        /* Maximum number of user
                                   processes */
```

```
int   v_cmap;       /* Size of core memory
                       allocation map */
int   v_smap;       /* Size of swap memory
                       allocation map */
int   v_hbuf;       /* Maximum number of buffer
                       headers */
int   v_hmask;      /* Maximum number of buffer
                       headers - 1 */
int   v_flock;      /* Maximum number of file locks */
int   v_phys;       /* Maximum number of simultaneous
                       phys calls */
int   v_clsize;     /* Click size */
int   v_txtrnd;     /* Number of clicks per segment */
int   v_bsize;      /* Block size */
int   v_cxmap;      /* Context map size */
int   v_clktick     /* Clock tick */
int   v_hz;         /* Hz */
int   v_usize;      /* Size of user structure */
int   v_pageshift;  /* Page shift */
int   v_pagemask;   /* Page mask */
int   v_segshift;   /* Segment shift */
int   v_segmask;    /* Segment mask */
int   v_ustart;     /* Starting virtual address for
                       user program */
int   v_uend;       /* Ending virtual address for
                       user program */
char* ve_call;      /* Pointer to last callout table */
int   v_stkgap;     /* Obsolete */
int   v_cputype;    /* CPU type (1=68000) */
int   v_cpuver;     /* CPU version ID
                       (1=68000, 2=68010, 3=68020) */
int   v_mmutype;    /* MMU type
                       (1=none, 2=SUN, 3=68451) */
int   v_doffset;    /* Data offset */
int   v_kvoffset;   /* Kernel virtual offset */
int   v_svtext;     /* Maximum number of text
                       loitering segments */
char* ve_svtext;    /* Pointer to last text
                       loitering segment
                       in table  */
int   v_pbuf;       /* Maximum number of buffers
                       for physio */
int   v_nscatload;  /* Maximum number of entries
```

```
                                in scatter map */
        int     v_udot;         /* Address of user structure */
        int     v_region;       /* Number of memory regions */
        int     v_sptmap;       /* Size of system virtual space */
        int     v_vhndfrac;     /* Fraction of MAXMEM to set a
                                   limit for running vehand  */
        int     v_maxpmem;      /* Maximum physical memory to use */
        int     v_nmbufs;       /* Buffers for networking */
        int     v_npty;         /* Number of pseudo tty's */
        int     v_maxcore;      /* Space used by kernel's heap
                                   (.../GEN/sys/heap_kmem.c) */
        int     v_maxheader;    /* Headers used by kernel's heap
                                   (.../GEN/sys/heap_kmem.c) */
        int     v_nstream;      /* Number of stream heads */
        int     v_nqueue;       /* Number of stream queues */
        int     v_nblk4096;     /* Number of of 4K stream blocks */
        int     v_nblk2048;     /* Number of of 2K stream blocks */
        int     v_nblk1024;     /* Number of 1K stream blocks */
        int     v_nblk512;      /* Number of 512K stream blocks */
        int     v_nblk256;      /* Number of 256K stream blocks */
        int     v_nblk64;       /* Number of 256K stream blocks */
        int     v_nblk16;       /* Number of 16 byte stream blocks */
        int     v_nblk4;        /* Number of 6 byte stream blocks */
        char    *ve_proctab     /* &proc[0] */
        int     v_slice         /* a process's time slice
        int     v_fill[128-67]/* sized to make var 512 bytes */
};
```

**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**

`uvar` will fail if:

`[EFAULT]` *v* points to an illegal address.

**SEE ALSO**

`ps`(1).

## NAME

wait – wait for child process to stop or terminate

## SYNOPSIS

```
int  wait(stat_loc)
int  *stat_loc;

int  wait((int*)0)
```

## DESCRIPTION

wait suspends the calling process until one of the immediate children terminates or until a child that is being traced stops, because it has hit a break point. The wait system call will return prematurely if a signal is received and if a child process stopped or terminated prior to the call on wait, return is immediate.

If *stat_loc* (taken as an integer) is nonzero, 16 bits of information called *status* are stored in the low order 16 bits of the location pointed to by *stat_loc*. *status* can be used to differentiate between stopped and terminated child processes and if the child process terminated, *status* identifies the cause of termination and passes useful information to the parent. This is accomplished in the following manner:

If the child process stopped, the high order 8 bits of *status* will contain the number of the signal that caused the process to stop and the low order 8 bits will be set equal to 0177.

If the child process terminated due to an exit call, the low order 8 bits of *status* will be zero and the high order 8 bits will contain the low order 8 bits of the argument that the child process passed to exit; see exit(2).

If the child process terminated due to a signal, the high order 8 bits of *status* will be zero and the low order 8 bits will contain the number of the signal that caused the termination. In addition, if the low order seventh bit (i.e., bit 200) is set, a "core image" will have been produced; see signal(3).

If a parent process terminates without waiting for its child processes to terminate, the parent process ID of each child process is set to 1. This means the initialization process inherits the child processes; see intro(2).

## RETURN VALUE

If wait returns due to the receipt of a signal, a value of −1 is returned to the calling process and errno is set to EINTR. If wait returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process.

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

**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.

**SEE ALSO**

`exec`(2), `exit`(2), `fork`(2), `intro`(2), `pause`(2), `ptrace`(2), `wait3`(2N), `signal`(3).

**WARNING**

See WARNING in `signal`(3).

## NAME

wait3 – wait for child process to stop or terminate

## SYNOPSIS

```
#include <sys/wait.h>

int wait3(status, options, 0)
union wait *status;
int options;
```

## DESCRIPTION

wait3 provides an interface for programs which 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 which 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).

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 ORing the two values.

The declaration of "union wait" is found in <sys/wait.h>. The third argument, 0, is a placeholder. The "normal case" is the same as wait(2).

## RETURN VALUE

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.

## SEE ALSO

exit(2), wait(2).

## NAME

write, writev – write on a file

## SYNOPSIS

```
int write(fildes, buf, nbytes)
int fildes;
char *buf;
unsigned nbytes;

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

int writev(fildes, iov, ioveclen)
int fildesd;
struct iovec *iov;
int ioveclen;
```

## DESCRIPTION

write attempts to write *nbytes* bytes from the buffer pointed to by *buf* to the file associated with the *fildes*. writev performs the same action, but gathers the output data from the *iovlen* buffers specified by the members of the iovec array: iov[0], iov[1], etc.

*fildes* is a file descriptor obtained from a creat, open, dup, fcntl, pipe, or socket system call.

On devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from write, the file pointer is incremented by the number of bytes actually written.

On devices incapable of seeking, writing always takes place starting at the current position. The value of a file pointer associated with such a device is undefined.

If the O_APPEND flag of the file status flags is set, the file pointer will be set to the end of the file prior to each write.

## RETURN VALUE

Upon successful completion the number of bytes actually written is returned. Otherwise, –1 is returned and errno is set to indicate the error.

## ERRORS

When attempting to write to a stream when no buffer space is currently available, if O_NDELAY is set, the write will return the number of bytes written before there were no buffers available. If O_NDELAY is clear, the write will block until buffers

become available.

write will fail and the file pointer will remain unchanged if one or more of the following are true:

[EIO] A physical I/O error has occurred.

[ENXIO] The device associated with the file descriptor is a block-special or character-special file and the value of the file pointer is out of range.

[EBADF] *fildes* is not a valid file descriptor open for writing.

[EPIPE] and SIGPIPE signal
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 pipe that is not open for reading by any process.

[EFBIG] An attempt was made to write a file that exceeds the process's file size limit or the maximum file size. See ulimit(2).

[EFAULT]
Part of *iov* or data to be written to the file points outside the process's allocated address space.

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

[EINTR] A signal was caught during the write system call.

[ENOSPC]
Not enough space left on the device containing the file.

If the number of bytes specified in a write request exceeds the available space (i.e., the per-process file size) limit (see ulimit(2)) or the size of the physical media, only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return 20. The next write of a nonzero number of bytes will give a failure return (except as noted below).

If the file being written is a pipe (or FIFO) and the O_NDELAY flag of the file flag word is set, then write to a full pipe (or FIFO) will return a count of 0. Otherwise (O_NDELAY clear), writes to a full pipe (or FIFO) will block until space becomes available.

**SEE ALSO**

creat(2), dup(2), fcntl(2), lseek(2), open(2), pipe(2), select(2N), socket(2N), ulimit(2).

# Table of Contents

## Section 3: Subroutines


intro .....introduction to subroutines and libraries
_tolower .....see conv(3C)
_toupper .....see conv(3C)
a64l .....convert between long integer and base-64 ASCII string
abort .....generate an IOT fault
abort .....terminate Fortran program
abs .....return integer absolute value
abs .....Fortran absolute value
acos .....Fortran arccosine intrinsic function
acos .....see trig(3M)
addmntent .....see getmntent(3)
addptabent .....see getptabent(3)
aimag .....Fortran imaginary part of complex argument
aint .....Fortran integer part intrinsic function
alog .....see log(3F)
alog10 .....see log10(3F)
amax0 .....see max(3F)
amax1 .....see max(3F)
amin0 .....see min(3F)
amin1 .....see min(3F)
amod .....see mod(3F)
and .....see bool(3F)
anint .....see round(3F)
asctime .....see ctime(3)
asin .....Fortran arcsine intrinsic function
asin .....see trig(3M)
assert .....verify program assertion
atan .....Fortran arctangent intrinsic function
atan .....see trig(3M)
atan2 .....Fortran arctangent intrinsic function
atan2 .....see trig(3M)
atof .....convert ASCII string to floating-point number

atoi ..... see strtol(3C)
atol ..... see strtol(3C)
bcmp ..... see bstring(3)
bcopy ..... see bstring(3)
bessel ..... Bessel functions
blt ..... block transfer data
blt512 ..... see blt(3C)
bool ..... Fortran bitwise boolean functions
bsearch ..... binary search a sorted table
bstring ..... bit and byte string operations
byteorder ..... convert values between host and network byte order
bzero ..... see bstring(3)
cabs ..... see abs(3F)
calloc ..... see malloc(3C)
calloc ..... see malloc(3X)
ccos ..... see cos(3F)
ceil ..... see floor(3M)
cexp ..... see exp(3F)
cfree ..... see malloc(3C)
char ..... see ftype(3F)
clearerr ..... see ferror(3S)
clock ..... report CPU time used
clog ..... see log(3F)
closedir ..... see directory(3)
cmplx ..... see ftype(3F)
conjg ..... Fortran complex conjugate intrinsic function
conv ..... translate characters
cos ..... Fortran cosine intrinsic function
cos ..... see trig(3M)
cosh ..... Fortran hyperbolic cosine intrinsic function
cosh ..... see sinh(3M)
crypt ..... generate DES encryption
csin ..... see sin(3F)
csqrt ..... see sqrt(3F)
ctermid ..... generate filename for terminal
ctime ..... convert date and time to ASCII
ctype ..... classify characters
curses ..... CRT screen handling and optimization package

curses5.0.....BSD-style screen functions with "optimal" cursor motion
cuserid.....get character login name of the user
dabs.....see abs(3F)
dacos.....see acos(3F)
dasin.....see asin(3F)
datan.....see atan(3F)
datan2.....see atan2(3F)
dble.....see ftype(3F)
dbm.....data base subroutines
dbminit.....see dbm(3X)
dcmplx.....see ftype(3F)
dconjg.....see conjg(3F)
dcos.....see cos(3F)
dcosh.....see cosh(3F)
ddim.....see dim(3F)
delete.....see dbm(3X)
dexp.....see exp(3F)
dial.....establish an out-going terminal line connection
dim.....Fortran positive difference intrinsic functions
dimag.....see aimag(3F)
dint.....see aint(3F)
directory.....directory operations
dlog.....see log(3F)
dlog10.....see log10(3F)
dmax1.....see max(3F)
dmin1.....see min(3F)
dn_comp.....see resolver(3N)
dn_expand.....see resolver(3N)
dmod.....see mod(3F)
dnint.....see round(3F)
dprod.....Fortran double precision product intrinsic function
drand48.....generate uniformly distributed pseudo-random numbers
dsign.....see sign(3F)
dsin.....see sin(3F)
dsinh.....see sinh(3F)
dsqrt.....see sqrt(3F)
dtan.....see tan(3F)
dtanh.....see tanh(3F)

dup2 ..... duplicate a descriptor
ecvt ..... convert floating-point number to string
edata ..... see end(3C)
encrypt ..... see crypt(3C)
end ..... last locations in program
endgrent ..... see getgrent(3C)
endhostent ..... see gethostent(3N)
endmntent ..... see getmntent(3)
endnetent ..... see getnetent(3N)
endnetgrent ..... see getnetgrent(3N)
endptabent ..... see getptabent(3)
endpwent ..... see getpwent(3C)
endservent ..... see getservent(3N)
endutent ..... see getut(3C)
enprotoent ..... see getprotoent(3N)
erand48 ..... see drand48(3C)
erf ..... error function and complementary error function
erfc ..... see erf(3M)
errno ..... see perror(3C)
etext ..... see end(3C)
exp ..... Fortran exponential intrinsic function
exp ..... exponential, logarithm, power, square root functions
fabs ..... see floor(3M)
fclose ..... close or flush a stream
fcvt ..... see ecvt(3C)
fdopen ..... see fopen(3S)
feof ..... see ferror(3S)
ferror ..... stream status inquiries
fetch ..... see dbm(3X)
fflush ..... see fclose(3S)
ffs ..... see bstring(3)
fgetc ..... see getc(3S)
fgetgrent ..... see getgrent(3C)
fgetpwent ..... see getpwent(3C)
fgets ..... see gets(3S)
fileno ..... see ferror(3S)
firstkey ..... see dbm(3X)
float ..... see ftype(3F)

mallinfo.....see malloc(3X)
malloc .....main memory allocator
malloc.....fast main memory allocator
mallopt.....see malloc(3X)
matherr.....error-handling function
max .....Fortran maximum-value functions
max0.....see max(3F)
max1 .....see max(3F)
mclock.....return Fortran time accounting
memccpy .....see memory(3C)
memchr.....see memory(3C)
memcmp .....see memory(3C)
memcpy .....see memory(3C)
memory .....memory operations
memset .....see memory(3C)
min .....Fortran minimum-value functions
min0 .....see min(3F)
min1 .....see min(3F)
mktemp .....make a unique filename
mod .....Fortran remaindering intrinsic functions
modf .....see frexp(3C)
monitor.....prepare execution profile
mount .....mount a file system
mrand48.....see drand48(3C)
nextkey .....see dbm(3X)
nint .....see round(3F)
nlist.....get entries from name list
not .....see bool(3F)
nrand48.....see drand48(3C)
ntohl .....see byteorder(3N)
ntohs.....see byteorder(3N)
numbptabent .....see getptabent(3)
opendir .....see directory(3)
or .....see bool(3F)
pclose.....see popen(3S)
perror.....system error messages
plot .....graphics interface subroutines
popen .....initiate pipe to/from a process

floor .....floor, ceiling, remainder, absolute value functions
fmod .....see floor(3M)
fopen .....open a stream
fprintf.....see printf(3S)
fputc.....see putc(3S)
fputs .....see puts(3S)
fread.....binary input/output
free .....see malloc(3C)
free.....see malloc(3X)
freopen .....see fopen(3S)
frexp.....manipulate parts of floating-point numbers
fscanf .....see scanf(3S)
fseek .....reposition a file pointer in a stream
ftell.....see fseek(3S)
ftok .....standard interprocess communication package
ftw.....walk a file tree
ftype .....explicit Fortran type conversion
fwrite.....see fread(3S)
gamma.....log gamma function
gcvt.....see ecvt(3C)
getarg .....return Fortran command-line argument
getc .....get character or word from a stream
getchar .....see getc(3S)
getcwd .....get pathname of current working directory
getenv .....return value for environment name
getenv .....return Fortran environment variable
getgrent.....obtain group file entry from a group file
getgrgid.....see getgrent(3C)
getgrnam .....see getgrent(3C)
gethostbyaddr .....see gethostent(3N)
gethostbyname .....see gethostent(3N)
gethostent .....get network host entry
getlogin.....get login name
getmntent .....get file system descriptor file entry
getnetbyaddr .....see getnetent(3N)
getnetbyname.....see getnetent(3N)
getnetent.....get network entry
getnetgrent.....get network group entry

getopt .....get option letter from argument vector
getpass .....read a password
getprotobyname.....see getprotoent(3N)
getprotobynumber .....see getprotoent(3N)
getprotoent.....get protocol entry
getptabent .....get partition table file entry
getpw .....get name from UID
getpwent.....get password file entry
getpwnam.....see getpwent(3C)
getpwuid.....see getpwent(3C)
gets .....get a string from a stream
getservbyname .....see getservent(3N)
getservbyport .....see getservent(3N)
getservent.....get service entry
getut.....access utmp file entry
getutent .....see getut(3C)
getutid.....see getut(3C)
getutline .....see getut(3C)
getw .....see getc(3S)
getwd .....get current working directory pathname
gmtime.....see ctime(3)
gsignal.....see ssignal(3C)
hasmntopt .....see getmntent(3)
hcreate .....see hsearch(3C)
hdestroy .....see hsearch(3C)
hsearch.....manage hash search tables
htonl .....see byteorder(3N)
htons.....see byteorder(3N)
hypot .....Euclidean distance function
iabs.....see abs(3F)
iargc.....return command line arguments
ichar .....see ftype(3F)
idim .....see dim(3F)
idint.....see ftype(3F)
idnint.....see round(3F)
ifix .....see ftype(3F)
index.....return location of Fortran substring
inet .....Internet address manipulation routines

inet_addr ..... see inet(3N)
inet_lnaof ..... see inet(3N)
inet_makeaddr ..... see inet(3N)
inet_netof ..... see inet(3N)
inet_network ..... see inet(3N)
inet_ntoa ..... see inet(3N)
initgroups ..... initialize group access list
innetgr ..... see getnetgrent(3N)
insque ..... insert/remove element from a queue
int ..... see ftype(3F)
irand ..... see rand(3F)
isalnum ..... see ctype(3C)
isalpha ..... see ctype(3C)
isascii ..... see ctype(3C)
isatty ..... see ttyname(3C)
iscntrl ..... see ctype(3C)
isdigit ..... see ctype(3C)
isgraph ..... see ctype(3C)
isign ..... see sign(3F)
islower ..... see ctype(3C)
isprint ..... see ctype(3C)
ispunct ..... see ctype(3C)
isspace ..... see ctype(3C)
isupper ..... see ctype(3C)
isxdigit ..... see ctype(3C)
j0 ..... see bessel(3M)
j1 ..... see bessel(3M)
jn ..... see bessel(3M)
jrand48 ..... see drand48(3C)
killpg ..... send signal to a process group
l3tol ..... convert between 3-byte integers and long integers
l64a ..... see a64l(3C)
lcong48 ..... see drand48(3C)
ldaclose ..... see ldclose(3X)
ldahread ..... read the archive header of a member of an archive file
ldaopen ..... see ldopen(3X)
ldclose ..... close a common object file
ldexp ..... see frexp(3C)

ldfcn .....common object file access routines
ldfhread.....read the file header of a common object file
ldgetname.....retrieve symbol name for object file
ldlinit .....see ldlread(3X)
ldlitem.....see ldlread(3X)
ldlread.....manipulate line no. entries of a common object file function
ldlseek.....seek to line no. entries of a section of a common object file
ldnlseek .....see ldlseek(3X)
ldnrseek.....see ldrseek(3X)
ldnshread.....see ldshread(3X)
ldnsseek .....see ldsseek(3X)
ldohseek .....seek to the optional file header of a common object file
ldopen.....open a common object file for reading
ldrseek.....seek to relocation entries of a section of a common object file
ldshread.....read indexed/named section header of a common object file
ldsseek .....seek to an indexed/named section of a common object file
ldtbindex..compute index of symbol table entry of a common object file
ldtbread.....read indexed symbol table entry of a common object file
ldtbseek.....seek to the symbol table of a common object file
len .....return length of Fortran string
lfind.....see lsearch(3C)
lge.....string comparision intrinsic functions
lgt.....see lge(3F)
line_push .....routine used to push streams line disciplines
lle.....see lge(3F)
llt .....see lge(3F)
localtime .....see ctime(3)
lockf.....record locking on files
log.....Fortran natural logarithm intrinsic function
log .....see exp(3M)
log10.....Fortran common logarithm intrinsic function
log10 .....see exp(3M)
logname .....return login name of user
longjmp .....see setjmp(3C)
lrand48 .....see drand48(3C)
lsearch .....linear search and update
lshift .....see bool(3F)
ltol3.....see l3tol(3C)

pow.....see exp(3M)
printf.....print formatted output
putc.....put character or word on a stream
putchar.....see putc(3S)
putenv.....change or add value to environment
putpwent.....write password file entry
puts.....put a string on a stream
pututline.....see getut(3C)
putw.....see putc(3S)
qsort.....quicker sort
rand.....simple random-number generator
rand.....Fortran uniform random-number generator
rcmd.....routines for returning a stream to a remote command
readdir.....see directory(3)
real.....see ftype(3F)
realloc.....see malloc(3C)
realloc.....see malloc(3X)
regcmp.....compile and execute a regular expression
regex.....see regcmp(3X)
remque.....see insque(3N)
res_mkquery.....see resolver(3N)
res_send.....see resolver(3N)
res_init.....see resolver(3N)
resolver.....resolver routines
rewind.....see fseek(3S)
rewinddir.....see directory(3)
rexec.....return stream to a remote command
round.....Fortran nearest integer functions
rpc.....library routines for remote procedure calls
rresvport.....see rcmd(3N)
rshift.....see bool(3F)
ruserok.....see rcmd(3N)
scandir.....scan a directory
scanf.....convert formatted input
seed48.....see drand48(3C)
seekdir.....see directory(3)
set42sig.....set 4.2 BSD signal interface
setbuf.....assign buffering to a stream

setgid.....see setuid(3)
setgrent.....see getgrent(3C)
sethostent.....see gethostent(3N)
setjmp.....non-local goto
setkey.....see crypt(3C)
setmntent.....see getmntent(3)
setnetent.....see getnetent(3N)
setnetgrent.....see getnetgrent(3N)
setprotoent.....see getprotoent(3N)
setptabent.....see getptabent(3)
setpwent.....see getpwent(3C)
setservent.....see getservent(3N)
setuid.....set user and group IDs
setutent.....see getut(3C)
setvbuf.....see setbuf(3S)
sgetl.....see sputl(3X)
sign.....Fortran transfer-of-sign intrinsic function
signal.....specify what to do upon receipt of a signal
signal.....specify Fortran action on receipt of a system signal
sin.....Fortran sine intrinsic function
sin.....see trig(3M)
sinh.....Fortran hyperbolic sine intrinsic function
sinh.....hyperbolic functions
sleep.....suspend execution for interval
slots.....ROM library functions
sngl.....see ftype(3F)
sprintf.....see printf(3S)
sputl.....access long integer data in a machine independent fashion
sqrt.....Fortran square root intrinsic function
sqrt.....see exp(3M)
srand.....see rand(3C)
srand.....see rand(3F)
srand48.....see drand48(3C)
sscanf.....see scanf(3S)
ssignal.....software signals
store.....see dbm(3X)
strcat.....see string(3C)
strchr.....see string(3C)

strcmp ..... see string(3C)
strcpy ..... see string(3C)
strcspn ..... see string(3C)
string ..... string operations
strlen ..... see string(3C)
strncat ..... see string(3C)
strncmp ..... see string(3C)
strncpy ..... see string(3C)
strpbrk ..... see string(3C)
strrchr ..... see string(3C)
strspn ..... see string(3C)
strtod ..... convert string to double-precision number
strtok ..... see string(3C)
strtol ..... convert string to integer
swab ..... swap bytes
sys_errlist ..... see perror(3C)
sys_nerr ..... see perror(3C)
system ..... issue a shell command from Fortran
system ..... issue a shell command
tan ..... Fortran tangent intrinsic function
tan ..... see trig(3M)
tanh ..... Fortran hyperbolic tangent intrinsic function
tanh ..... see sinh(3M)
tdelete ..... see tsearch(3C)
telldir ..... see directory(3)
tempnam ..... see tmpnam(3S)
termcap ..... terminal independent operation routines
tfind ..... see tsearch(3C)
tgetent ..... see termcap(3X)
tgetflag ..... see termcap(3X)
tgetnum ..... see termcap(3X)
tgetstr ..... see termcap(3X)
tgoto ..... see termcap(3X)
tmpfile ..... create a temporary file
tmpnam ..... create a name for a temporary file
toascii ..... see conv(3C)
tolower ..... see conv(3C)
toupper ..... see conv(3C)

tputs .....see termcap(3X)
trig .....trigonometric functions
tsearch.....manage binary search trees
ttyname .....find name of a terminal
ttyslot .....find the slot in the utmp file of the current user
twalk.....see tsearch(3C)
tzset.....see ctime(3)
tzsetwall.....see ctime(3)
umount .....unmount a file system
ungetc.....push character back into input stream
utmpname .....see getut(3C)
varargs .....handle variable argument list
vfprintf.....see vprintf(3S)
vprintf .....print formatted output of a varargs argument list
vsprintf .....see vprintf(3S)
xdr.....library routines for external data representation
xor .....see bool(3F)
y0.....see bessel(3M)
y1.....see bessel(3M)
yn.....see bessel(3M)
yp_all .....see ypclnt(3N)
yp_bind.....see ypclnt(3N)
yp_first.....see ypclnt(3N)
yp_get_default_domain .....see ypclnt(3N)
yp_master .....see ypclnt(3N)
yp_match .....see ypclnt(3N)
yp_next.....see ypclnt(3N)
yp_order .....see ypclnt(3N)
yp_unbind.....see ypclnt(3N)
ypclnt.....yellow pages client interface
yperr_string .....see ypclnt(3N)
ypprot_err .....see ypclnt(3N)
zabs .....see abs(3F)

## NAME

intro – introduction to subroutines and libraries

## SYNOPSIS

```
#include <stdio.h>
#include <math.h>
```

## DESCRIPTION

This section describes functions found in various libraries, other than those functions that directly invoke system primitives (described in Section 2 of this volume). Major collections are identified by a letter after the section number:

(3C) These functions, together with those of Section 2 (and those marked (3S)), constitute the Standard C Library, `libc`, which is automatically loaded by the C compiler, `cc`(1). The link editor `ld`(1) searches this library under the `-lc` flag option. Some functions require declarations that can be included in the program being compiled by adding the line

`#include <`*header-filename*`>`

The appropriate header file is indicated in the SYNOPSIS part of a function description.

(3F) These functions constitute the Fortran intrinsic function library, `libF77` and are automatically available to the Fortran programmer, requiring no special invocation of the compiler. These functions are flagged with the (3F) suffix on the associated manual page entries and appear in their own alphabetically organized subsection at the end of this section.

(3M) These functions constitute the Math Library, `libm`. They are automatically loaded as needed by the Fortran compiler `f77`(1). They are not automatically loaded by the C compiler, `cc`(1); however, the link editor searches this library under the `-lm` flag option. Declarations for these functions may be obtained from the header file `<math.h>`.

(3N) These functions are networking routines and, unless otherwise noted, are found in the Standard C Library `libc.a`.

(3X) Various specialized libraries. The files in which these libraries are found are given on the appropriate pages.

(3S) These functions constitute the standard I/0 package; An introduction to this package follows under the heading "STANDARD I/O." The functions are in the library `libc`, already mentioned. Declarations should be obtained from the `#include` file `<stdio.h>`.

DEFINITIONS

A **character** is any bit pattern able to fit into a byte on the machine. The **null character** is a character with value 0, represented in the C language as "`\0`". A **character array** is a sequence of characters. A **null-terminated character array** is a sequence of characters, the last of which is the null character. A **string** is a designation for a null-terminated character array. The **null string** is a character array containing only the null character. A **null pointer** is the value that is obtained by casting `0` into a pointer. The C language guarantees that this value will not match that of any legitimate pointer, so many functions that return pointers return it to indicate an error. `NULL` is defined as `0` in `<stdio.h>`; the user can include his own definition if he is not using `<stdio.h>`.

Many groups of Fortran intrinsic functions have "generic" function names that do not require explicit or implicit type declaration. The type of the function is determined by the type of its argument(s). For example, the generic function `max` returns an integer value if given integer arguments (`max0`), a real value if given real arguments (`amax1`), or a double-precision value if given double-precision arguments (`dmax1`).

STANDARD I/O

The functions described in the entries of subclass (3S) in this manual provide an efficient, user level I/O buffering scheme. The functions are in the library `libc` and declarations should be obtained from the header file `<stdio.h>`.

The input/output function may be grouped into the following categories: file access, file status, input, output, and miscellaneous. For lists of the functions in each category, refer to the "Libraries" sections of *A/UX Programming Languages and Tools, Volume 1*. The inline macros `getc`(3S) and `putc`(3S) handle characters quickly. The macros `getchar` and `putchar`, and the higher-level routines `fgetc`, `fgets`, `fprintf`, `fputc`, `fputs`, `fread`, `fscanf`, `fwrite`, `gets`, `getw`, `printf`,

puts, putw, and scanf 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. Normally, there are three open streams with constant pointers declared in the <stdio.h> header file and associated with the standard open files:

| | |
|---|---|
| stdin | standard input file |
| stdout | standard output file |
| stderr | standard error file. |

A constant NULL (0) designates a nonexistent pointer.

An integer constant EOF (–1) is returned upon end-of-file or error by most integer functions that deal with streams (see the individual descriptions for details).

An integer constant BUFSIZ specifies the size of the buffers used by the particular implementation.

Any program that uses this package must include the header file of pertinent macro definitions, as follows:

```
#include <stdio.h>
```

The functions and constants mentioned in the (3S) entries are declared in that header file <stdio.h> and need no further declaration. The constants and the following functions are implemented as macros: getc, getchar, putc, putchar, feof, ferror, clearerr, and fileno. Redeclaration of these names is perilous.

The <stdio.h> file is illustrated in the "Libraries" sections of the *A/UX Programming Languages and Tools, Volume 1*.

*Note:* Invalid stream pointers cause serious errors, possibly including program termination. Individual function descriptions describe the possible error conditions.

For descriptions and examples of header files, refer to "The Standard C Library (libc)," "The C Math Library," and "The C Object Library" in *A/UX Programming Languages and Tools, Volume 1*.

**FILES**

/lib/libc.a
/usr/lib/libF77.a

/lib/libm.a

**SEE ALSO**

ar(1), cc(1), f77(1), ld(1), lint(1), nm(1), open(2), close(2), lseek(2), pipe(2), read(2), write(2), ctermid(3S), cuserid(3S), fclose(3S), ferror(3S), fopen(3S), fread(3S), fseek(3S), getc(3S), gets(3S), popen(3S), printf(3S), putc(3S), puts(3S), scanf(3S), setbuf(3S), system(3S), tmpfile(3S), tmpnam(3S), ungetc(3S), math(5). *A/UX Programming Languages and Tools, Volume 1*.

**RETURN VALUE**

Functions in the C and Math Libraries (3C and 3M) may return the conventional values 0 or ± HUGE (the largest-magnitude single-precision floating-point numbers; HUGE is defined in the <math.h> header file) when the function is undefined for the given arguments or when the value is not representable. In these cases, the external variable errno (see intro(2)) is set to the value EDOM or ERANGE. Because many of the Fortran intrinsic functions use the routines found in the Math Library, the same conventions apply.

**WARNING**

Many of the functions in the libraries call and/or refer to other functions and external variables described in this section and in Section 2 (System Calls). If a program inadvertantly defines a function or external variable with the same name, the presumed library version of the function or external variable may not be loaded. The lint(1) program checker reports name conflicts of this kind as "multiple declarations" of the names in question. Definitions for sections 2, 3C, and 3S are checked automatically. Other definitions can be included by using the −l option (for example, −lm includes definitions for libm, the Math Library, section 3M). Use of lint is highly recommended.

## NAME

a64l, l64a – convert between long integer and base-64 ASCII string

## SYNOPSIS

```
long  a64l(s)
char  *s;

char  *l64a(l)
long  l;
```

## DESCRIPTION

These functions are used to maintain numbers stored in base-64 ASCII characters. This is a notation by which long integers can be represented by up to 6 characters; each character represents a "digit" in a radix-64 notation.

The characters used to represent "digits" are . for 0, / for 1, 0 through 9 for 2–11, A through Z for 12–37, and a through z for 38–63.

a64l takes a pointer to a null-terminated base-64 representation and returns a corresponding long value. If the string pointed to by *s* contains more than 6 characters, uses the first 6.

l64a takes a long argument and returns a pointer to the corresponding base-64 representation. If the argument is 0, l64a returns a pointer to a null string.

## BUGS

The value returned by l64a is a pointer into a static buffer, the contents of which are overwritten by each call.

**NAME**

abort – generate an IOT fault

**SYNOPSIS**

```
int abort()
```

**DESCRIPTION**

abort first closes all open files if possible, then causes an IOT signal to be sent to the process. This usually results in termination with a core dump.

It is possible for abort to return control if SIGIOT is caught or ignored, in which case the value returned is that of the kill(2) system call.

**DIAGNOSTICS**

If SIGIOT is neither caught nor ignored, and the current directory is writable, a core dump is produced and the message abort – core dumped is written by the shell.

**SEE ALSO**

sdb(1), exit(2), kill(2), signal(3).

## NAME

abort – terminate Fortran program

## SYNOPSIS

```
call abort()
```

## DESCRIPTION

abort terminates the program which calls it, closing all open files truncated to the current position of the file pointer.

## DIAGNOSTICS

When invoked, prints "Fortran abort routine called" on the standard error output.

## SEE ALSO

abort(3C).

**NAME**

abs – return integer absolute value

**SYNOPSIS**

```
int  abs(i)
int  i;
```

**DESCRIPTION**

abs returns the absolute value of its integer operand.

**BUGS**

In two's-complement representation, the absolute value of the negative integer with largest magnitude is returned.

Some implementations trap this error, but others simply ignore it.

**SEE ALSO**

floor(3M).

**NAME**

abs, iabs, dabs, cabs, zabs – Fortran absolute value

**SYNOPSIS**

```
integer i1, i2
real r1, r2
double precision dp1, dp2
complex cx1, cx2
double complex dx1, dx2

r2=abs(r1)

i2=iabs(i1)
i2=abs(i1)

dp2=dabs(dp1)
dp2=abs(dp1)

cx2=cabs(cx1)
cx2=abs(cx1)

dx2=zabs(dx1)
dx2=abs(dx1)
```

**DESCRIPTION**

abs is the family of absolute value functions. iabs returns the integer absolute value of its integer argument. dabs returns the double-precision absolute value of its double-precision argument. cabs returns the complex absolute value of its complex argument. zabs returns the double-complex absolute value of its double-complex argument. The generic form abs returns the type of its argument.

**SEE ALSO**

floor(3M).

## NAME

acos, dacos – Fortran arccosine intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2

r2=acos(r1)

dp2=dacos(dp1)
dp2=acos(dp1)
```

## DESCRIPTION

acos returns the real arccosine of its real argument. dacos returns the double-precision arccosine of its double-precision argument. The generic form acos may be used with impunity because its argument determines the type of the returned value.

## SEE ALSO

trig(3M).

## NAME

aimag, dimag – Fortran imaginary part of complex argument

## SYNOPSIS

```
real r
complex cxr
double precision dp
double complex cxd

r=aimag(cxr)

dp=dimag(cxd)
```

## DESCRIPTION

aimag returns the imaginary part of its single-precision complex argument. dimag returns the double-precision imaginary part of its double-complex argument.

## NAME

aint, dint – Fortran integer part intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2

r2=aint(r1)

dp2=dint(dp1)
dp2=aint(dp1)
```

## DESCRIPTION

aint returns the truncated value of its real argument in a real. dint returns the truncated value of its double-precision argument as a double-precision value. aint may be used as a generic function name, returning either a real or double-precision value depending on the type of its argument.

**NAME**

asin, dasin – Fortran arcsine intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2

r2=asin(r1)

dp2=dasin(dp1)
dp2=asin(dp1)
```

**DESCRIPTION**

asin returns the real arcsine of its real argument. dasin returns the double-precision arcsine of its double-precision argument. The generic form asin may be used with impunity as it derives its type from that of its argument.

**SEE ALSO**

trig(3M).

**NAME**

assert – verify program assertion

**SYNOPSIS**

```
#include <assert.h>

assert(expression)
int expression;
```

**DESCRIPTION**

This macro is useful for putting diagnostics into programs. If *expression* is false (zero) when assert is executed, assert prints

Assertion failed: *expression, file xyz, line nnn*

on the standard error output and aborts. In the error message, *xyz* is the name of the source file and *nnn* is the source line number of the assert statement.

Compiling with the preprocessor option -DNDEBUG (see cpp(1)) or with the preprocessor control statement #define NDEBUG ahead of the #include <assert.h> statement, stops assertions from being compiled into the program.

**NOTE**

assert cannot be used in an expression since it turns into and if statement.

**SEE ALSO**

cpp(1), abort(3C).

## NAME

atan, datan – Fortran arctangent intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2

r2=atan(r1)

dp2=datan(dp1)
dp2=atan(dp1)
```

## DESCRIPTION

atan returns the real arctangent of its real argument. datan returns the double-precision arctangent of its double-precision argument. The generic form atan may be used with a double-precision argument returning a double-precision value.

## SEE ALSO

trig(3M).

**NAME**

atan2, datan2 – Fortran arctangent intrinsic function

**SYNOPSIS**

```
real r1, r2, r3
double precision dp1, dp2, dp3

r3=atan2(r1, r2)

dp3=datan2(dp1, dp2)
dp3=atan2(dp1, dp2)
```

**DESCRIPTION**

atan2 returns the arctangent of *arg1/arg2* as a real value. datan2 returns the double-precision arctangent of its double-precision arguments. The generic form atan2 may be used with impunity with double-precision arguments.

**SEE ALSO**

trig(3M).

**NAME**

atof – convert ASCII string to floating-point number

**SYNOPSIS**

```
double atof(nptr)
char *nptr;
```

**DESCRIPTION**

atof converts a character string pointed to by *nptr* to a double-precision floating point number. The first unrecognized character ends the conversion. atof recognizes an optional string of white space characters (blanks or tabs), 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. If the string begins with an unrecognized character, atof returns the value zero.

```
atof(str)
```

is equivalent to

```
strtod(str, (char **)NULL)
```

**ERRORS**

When the correct value would overflow, atof returns HUGE, and sets errno to ERANGE. Zero is returned on underflow.

**SEE ALSO**

scanf(3S), strtod(3C), strtol(3C).

**NAME**

j0, j1, jn, y0, y1, yn – Bessel functions

**SYNOPSIS**

```
#include <math.h>

double j0(x)
double x;

double j1(x)
double x;

double jn(n, x)
int n;
double x;

double y0(x)
double x;

double y1(x)
double x;

double yn(n, x)
int n;
double x;
```

**DESCRIPTION**

j0 and j1 return Bessel functions of *x* of the first kind of orders 0 and 1 respectively. jn returns the Bessel function of *x* of the first kind of order *n*.

y0 and y1 return the Bessel functions of *x* of the second kind of orders 0 and 1 respectively. yn returns the Bessel function of *x* of the second kind of order *n*. The value of *x* must be positive.

**ERRORS**

Nonpositive arguments cause y0, y1, and yn to return the value −HUGE and to set errno to EDOM. In addition, a message indicating DOMAIN error is printed on the standard error output.

Arguments too large in magnitude cause j0, j1, y0 and y1 to return zero and set errno to ERANGE. In addition, a message indicating TLOSS error is printed on the standard error output.

**NOTE**

These error-handling procedures may be changed with the function matherr(3M).

**SEE ALSO**

matherr(3M).

**NAME**

blt, blt512 – block transfer data

**SYNOPSIS**

```
int blt(to, from, count)
char *to;
char *from;
int count;

int blt512(to, from, count)
char *to;
char *from;
int count;
```

**DESCRIPTION**

blt does a fast copy of *count* bytes of data starting at address *from* to address *to*.

blt512 does a fast copy of *count* number of consecutive 512 byte units starting at address *from* to address *to*.

**SEE ALSO**

memory(3).

## NAME

and, or, xor, not, lshift, rshift – Fortran bitwise boolean functions

## SYNOPSIS

```
integer i, j, k
real a, b, c
double precision dp1, dp2, dp3

k=and(i, j)
c=or(a, b)
j=xor(i, a)
j=not(i)
k=lshift(i, j)
k=rshift(i, j)
```

## DESCRIPTION

The generic intrinsic boolean functions and, or, and xor return the value of the binary operations on their arguments. not is a unary operator returning the one's complement of its argument. lshift and rshift return the value of the first argument shifted left or right, respectively, the number of times specified by the second (integer) argument.

The boolean functions are generic, i.e., defined for all data types as arguments and return values. Where required, the compiler generates appropriate type conversions.

## NOTE

Although defined for all data types, use of boolean functions on non-integer data is not productive.

## BUGS

The implementation of the shift functions may cause large shift values to deliver unexpected results.

## NAME

bsearch – binary search a sorted table

## SYNOPSIS

```
#include <search.h>

char *bsearch(key, base, nel, width, compar)
char *key;
char *base;
unsigned nel; width;
int (*compar)();
```

## DESCRIPTION

bsearch is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer into a table indicating where a datum may be found. The table must be previously sorted in increasing order according to a provided comparison function. *key* points to a datum instance to be sought in the table. *base* points to the element at the base of the table. *nel* is the number of elements in the table. *width* is the width of an element in bytes; *sizeof*(**key*) should be used. *compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero as accordingly the first argument is to be considered less than, equal to, or greater than the second.

## EXAMPLE

The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

This code fragment reads in strings and either finds the corresponding node and prints out the string and its length, or prints an error message.

```
#include <stdio.h>
#include <search.h>

#define    TABSIZE    1000

struct node {           /* these are stored in the table */
    char *string;
    int length;
};
struct node table[TABSIZE];   /* table to be searched */
```

```
        .
        .
        .
{
        struct node /*node_ptr, node;
        int node_compare( );   /* routine to compare 2 nodes */
        char str_space[20];    /* space to read string into */
        .
        .
        .
        node.string = str_space;
        while (scanf("%s", node.string) != EOF) {
            node_ptr = (struct node *)bsearch((char *)(&node),
                    (char *)table, TABSIZE,
                    sizeof(struct node), node_compare);
            if (node_ptr != NULL) {
                (void)printf("string = %20s, length = %d\n",
                    node_ptr->string, node_ptr->length);
            } else {
                (void)printf("not found: %s\n", node.string);
            }
        }
}
/*
        This routine compares two nodes based on an
        alphabetical ordering of the string field.
*/
int
node_compare(node1, node2)
struct node *node1, *node2;
{
        return strcmp(node1->string, node2->string);
}
```

**NOTES**

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

**RETURN VALUE**

A NULL pointer is returned if the key cannot be found in the table.

**SEE ALSO**

hsearch(3C), lsearch(3C), qsort(3C), tsearch(3C).

## NAME

bcopy, bcmp, bzero, ffs – bit and byte string operations

## SYNOPSIS

```
int bcopy(b1, b2, length)
char *b1, *b2;
int length;

int bcmp(b1, b2, length)
char *b1, *b2;
int length;

int bzero(b, length)
char *b;
int length;

int ffs(i)
int i;
```

## DESCRIPTION

The macro bcopy, and the functions bcmp, and bzero operate on variable length strings of bytes. They do not check for null bytes as the routines in string(3C) do.

bcopy copies *length* bytes from string *b1* to the string *b2*.

bcmp compares byte string *b1* against byte string *b2*, returning zero if they are identical, nonzero otherwise. Both strings are assumed to be *length* bytes long.

bzero places *length* 0 bytes in the string *b1*.

ffs finds 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 –1 indicates the value passed is zero.

## FILES

/usr/include/sys/param.h

## BUGS

The bcmp and bcopy routines take parameters backwards from strcmp and strcpy.

## SEE ALSO

memory(3C), string(3).

## NAME

htonl, htons, ntohl, ntohs – convert values between host and network byte order

## SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>

u_long htonl(hostlong);
u_long hostlong;

u_short htons(hostshort);
u_short hostshort;

u_long ntohl(netlong);
u_long netlong;

u_short ntohs(netshort);
u_short netshort;
```

## DESCRIPTION

These macros convert 16 and 32 bit quantities between network byte order and host byte order. On machines in the Motorola 68000 family such as the Macintosh II, 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 gethostent(3N) and getservent(3N).

## SEE ALSO

gethostent(3N), getservent(3N).

**NAME**

clock – report CPU time used

**SYNOPSIS**

```
long clock()
```

**DESCRIPTION**

`clock` returns the amount of CPU time (in microseconds) used since the first call to `clock`. The time reported is the sum of the user and system times of the calling process and its terminated child processes for which it has executed `wait`(2) or `system`(3S).

**SEE ALSO**

`times`(2), `wait`(2), `system`(3S).

**BUGS**

The value returned by `clock` is defined in microseconds for compatibility with systems that have CPU clocks with much higher resolution. Because of this, the value returned wraps around after accumulating only 2,147 seconds of CPU time (about 36 minutes).

**NAME**

conjg, dconjg – Fortran complex conjugate intrinsic function

**SYNOPSIS**

```
complex  cx1, cx2
double complex   dx1, dx2

cx2=conjg(cx1)

dx2=dconjg(dx1)
```

**DESCRIPTION**

conjg returns the complex conjugate of its complex argument. dconjg returns the double-complex conjugate of its double-complex argument.

## NAME

toupper, tolower, _toupper, _tolower, toascii – translate characters

## SYNOPSIS

```
#include <ctype.h>

int toupper(c)
int c;

int tolower(c)
int c;

int _toupper(c)
int c;

int _tolower(c)
int c;

int toascii(c)
int c;
```

## DESCRIPTION

toupper and tolower have as domain the range of getc(3S): the integers from –1 through 255. If the argument of toupper represents a lowercase letter, the result is the corresponding uppercase letter. If the argument of tolower represents an uppercase letter, the result is the corresponding lowercase letter. All other arguments in the domain are returned unchanged.

The macros _toupper and _tolower, are macros that accomplish the same thing as toupper and tolower but have restricted domains and are faster. _toupper requires a lowercase letter as its argument; its result is the corresponding uppercase letter. The macro _tolower requires an uppercase letter as its argument; its result is the corresponding lowercase letter. Arguments outside the domain cause undefined results.

The toascii macro yields its argument with all bits turned off that are not part of a standard ASCII character; it is intended for compatibility with other systems.

## SEE ALSO

ctype(3C), getc(3S).

**NAME**

cos, dcos, ccos – Fortran cosine intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2
complex cx1, fx2

r2=cos(r1)

dp2=dcos(dp1)
dp2=cos(dp1)

cx2=ccos(cx1)
cx2=cos(cx1)
```

**DESCRIPTION**

cos returns the real cosine of its real argument. dcos returns the double-precision cosine of its double-precision argument. ccos returns the complex cosine of its complex argument. The generic form cos may be used with impunity because its returned type is determined by that of its argument.

**SEE ALSO**

trig(3M).

**NAME**

cosh, dcosh – Fortran hyperbolic cosine intrinsic function

**SYNOPSIS**

```
real r1, r2
double precision dp1, dp2

r2=cosh(r1)

dp2 =dcosh(dp1)
dp2=cosh(dp1)
```

**DESCRIPTION**

cosh returns the real hyperbolic cosine of its real argument. dcosh returns the double-precision hyperbolic cosine of its double-precision argument. The generic form cosh may be used to return the hyperbolic cosine in the type of its argument.

**SEE ALSO**

sinh(3M).

## NAME

crypt, setkey, encrypt – generate DES encryption

## SYNOPSIS

```
char *crypt(key, salt)
char *key, *salt;

void setkey(key)
char *key;

void encrypt(block, edflag)
char *block;
int edflag;
```

## DESCRIPTION

crypt is the password encryption function. It is based on the NBS Data Encryption Standard (DES), with variations intended to frustrate use of hardware implementations of the DES for key search.

*key* is a user's typed password. *salt* is a 2-character string chosen from the set [a-zA-Z0-9./]; this string is used to perturb the DES algorithm in one of 4,096 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. The first 2 characters are the *salt* itself.

The setkey and encrypt 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; this gives a 56-bit key which is set into the machine. The 56-bit key is used with the above-mentioned algorithm to encrypt or decrypt the string *block* with the function encrypt.

The argument to the encrypt entry is a character array of length 64 containing only the characters with numerical value 0 and 1. 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 zero, the argument is encrypted; if nonzero, it is decrypted.

## SEE ALSO

crypt(1), login(1), passwd(1), getpass(3C), passwd(4).

## BUGS

The return value points to static data that is overwritten by each call.

## NAME

ctermid – generate filename for terminal

## SYNOPSIS

```
#include <stdio.h>

char *ctermid(s)
char *s;
```

## DESCRIPTION

ctermid generates the pathname of the controlling terminal for the current process, and stores it in a string.

If *s* is a NULL pointer, the string is stored in an internal static area, the contents of which are overwritten at the next call to ctermid, and the address of which is returned. Otherwise, *s* is assumed to point to a character array of at least L_ctermid elements; the pathname is placed in this array and the value of *s* is returned. The constant L_ctermid is defined in the <stdio.h> header file.

## NOTES

The difference between ctermid and ttyname(3C) is that ttyname must be handed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while ctermid returns a string (/dev/tty) that refers to the terminal if used as a filename. For this reason, ttyname is useful only if the process already has at least one file open to a terminal.

## SEE ALSO

ttyname(3C).

**NAME**

ctime, localtime, gmtime, asctime, tzset, tzsetwall – convert date and time to ASCII

**SYNOPSIS**

```
extern char *tzname[2];

void tzset()

void tzsetwall()

char *ctime(clock)
long *clock;

#include <time.h>

char *asctime(tm)
struct tm *tm;

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

extern long timezone;
extern int daylight;
```

**DESCRIPTION**

`tzset` uses the value of the environment variable `TZ` to set time conversion information used by *localtime*. If `TZ` does not appear in the environment, the best available approximation to local wall clock time is used by *localtime*. If `TZ` appears in the environment but its value is a null string, Greenwich Mean Time is used; if `TZ` appears and begins with a slash, it is used as the absolute pathname of the `tzfile`(4)-format file from which to read the time conversion information; if `TZ` appears and begins with a character other than a slash, it's used as a pathname relative to a system time conversion information directory.

`tzsetwall` sets things up so that `localtime` returns the best available approximation of local wall clock time.

`ctime` converts a long integer, pointed to by *clock*, representing the time in seconds since 00:00:00 GMT, January 1, 1970, and returns a pointer to a 26-character string of the form

```
Thu Nov 24 18:22:48 1986\n\0
```

All the fields have constant width.

`localtime` and `gmtime` return pointers to "`tm`" structures, described below. `localtime` corrects for the time zone and any time zone adjustments (such as Daylight Savings time in the U.S.A.). Before doing so, `localtime` calls `tzset` (if `tzset` has not been called in the current process). After filling in the "`tm`" structure, `localtime` sets the `tm_isdst`'th element of `tzname` to a pointer to an ASCII string that's the time zone abbreviation to be used with `localtime`'s return value.

`gmtime` converts to Greenwich Mean Time (GMT).

`asctime` converts a time value contained in a "`tm`" structure to a 26-character string, as shown in the above example, and returns a pointer to the string.

Declarations of all the functions and externals, and the "`tm`" structure, are in the `<time.h>` header file. The structure (of type) `struct tm` includes the following fields:

```
int tm_sec;     /* seconds (0 - 59) */
int tm_min;     /* minutes (0 - 59) */
int tm_hour;    /* hours (0 - 23) */
int tm_mday;    /* day of month (1 - 31) */
int tm_mon;     /* month of year (0 - 11) */
int tm_year;    /* year - 1900 */
int tm_wday;    /* day of week (Sunday = 0) */
int tm_yday;    /* day of year (0 - 365) */
int tm_isdst;   /* is DST in effect? */
```

`tm_isdst` is nonzero if a time zone adjustment such as Daylight Savings time is in effect.

The external `long` variable *timezone* contains the difference, in seconds, between GMT and local standard time (in EST, *timezone* is 5*60*60); the external variable *daylight* is nonzero if, and only if, the standard U.S.A. Daylight Savings Time conversion should be applied. The program knows about the peculiarities of this conversion in 1974 and 1975; if necessary, a table for these years can be extended.

If an environment variable named `TZ` is present, `asctime` uses the contents of the variable to override the default time zone. The value of `TZ` must be a 3-letter time zone name, followed by a number representing the difference between local time and Greenwich Mean Time in hours, followed by an optional 3-letter name for a daylight time zone. For example, the setting for New Jersey would be `EST5EDT`. The effects of setting `TZ` are thus to

change the values of the external variables *timezone* and *daylight*; in addition, the time zone names contained in the external variable

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

are set from the environment variable `TZ`. The function `tzset` sets these external variables from `TZ`; `tzset` is called by `asctime` and may also be called explicitly by the user.

Note that in most installations, `TZ` is set by default when the user logs on, to a value in the local `/etc/profile` file (see `profile(4)`).

**FILES**

| | |
|---|---|
| `/etc/zoneinfo` | time zone information directory |
| `/etc/zoneinfo/localtime` | local time zone file |

**SEE ALSO**

`time(2)`, `getenv(3)`, `tzfile(4)`, `profile(4)`, `environ(5)`.

**NOTE**

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

**NAME**

isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii – classify characters

**SYNOPSIS**

```
#include <ctype.h>

int isalpha(c)
int c;

...
```

**DESCRIPTION**

These macros classify character-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. `isascii` is defined on all integer values; the rest are defined only where `isascii` is true and on the single non-ASCII value EOF (–1); see `intro`(3)).

| | |
|---|---|
| `isalpha` | *c* is a letter. |
| `isupper` | *c* is an upper-case letter. |
| `islower` | *c* is a lower-case letter. |
| `isdigit` | *c* is a digit [0-9]. |
| `isxdigit` | *c* is a hexadecimal digit [0-9], [A-F] or [a-f]. |
| `isalnum` | *c* is an alphanumeric (letter or digit). |
| `isspace` | *c* is a space, tab, carriage return, *newline*, vertical tab, or form-feed. |
| `ispunct` | *c* is a punctuation character (neither control nor alphanumeric). |
| `isprint` | *c* is a printing character, code 040 (space) through 0176 (tilde). |
| `isgraph` | *c* is a printing character, similar to `isprint` except false for space. |
| `iscntrl` | *c* is a delete character (0177) or an ordinary control character (less than 040). |
| `isascii` | *c* is an ASCII character, code less than 0200. |

**RETURN VALUE**

If the argument to any of these macros is not in the domain of the

function, the result is undefined.

**SEE ALSO**

intro(3), ascii(5).

## NAME

curses – CRT screen handling and optimization package

## SYNOPSIS

```
#include <curses.h>
cc [flags] files -lcurses [libraries]
```

## DESCRIPTION

These routines give the user a method of updating screens with reasonable optimization. In order to initialize the routines, the routine `initscr()` must be called before any of the other routines that deal with windows and screens are used. The routine `endwin()` should be called before exiting. To get character-at-a-time input without echoing, (most interactive, screen oriented-programs want this) after calling `initscr()` you should call ``nonl(); cbreak(); noecho();''

The full curses interface permits manipulation of data structures called "windows" which can be thought of as two dimensional arrays of characters representing all or part of a terminal screen. A default window called `stdscr` is supplied, and others can be created with `newwin`. Windows are referred to by variables declared "`WINDOW *`," the type `WINDOW` is defined in `curses.h` to be a C structure. These data structures are manipulated with functions described below, among which the most basic are `move`, and `addch`. (More general versions of these functions are included with names beginning with "`w`", allowing you to specify a window. The routines not beginning with "`w`", affect `stdscr`.) Then `refresh()` is called, telling the routines to make the users CRT screen look like `stdscr`.

"Mini-Curses" is a subset of curses which does not allow manipulation of more than one window. To invoke this subset, use `-DMINICURSES` as a `cc` option. This level is smaller and faster than full curses.

If the environment variable `TERMINFO` is defined, any program using curses will check for a local terminal definition before checking in the standard place. For example, if the standard place is `/usr/lib/terminfo`, and `TERM` is set to "`vt100`", then normally the compiled file is found in `/usr/lib/terminfo/v/vt100`. (The "`v`" is copied from the first letter of "`vt100`" to avoid creation of huge directories.) However, if `TERMINFO` is set to `/usr/paul/myterms`, `curses` will first check `/usr/paul/myterms/v/vt100`, and if that fails, will then check

/usr/lib/terminfo/v/vt100. This is useful for developing experimental definitions or when write permission in /usr/lib/terminfo is not available.

## SEE ALSO

terminfo(4).

## FUNCTIONS

Routines listed here may be called when using the full curses. Those marked with a plus (+) are macros. Those marked with an asterisk (*) may be called when using Mini-Curses.

| | |
|---|---|
| addch (*ch*) * | add a character to stdscr (like putchar) (wraps to next line at end of line) |
| addstr (*str*) * | calls addch with each character in *str* |
| attroff (*attrs*) * | turn off attributes named |
| attron (*attrs*) * | turn on attributes named |
| attrset (*attrs*) * | set current attributes to *attrs* |
| baudrate () * | current terminal speed |
| beep () * | sound beep on terminal |
| box (*win, vert, hor*) | draw a box around edges of *win* *vert* and *hor* are chars to use for vertical and horizontal edges of box |
| clear () | clear stdscr |
| clearok (*win, bf*) | clear screen before next redraw of *win* |
| clrtobot () | clear to bottom of stdscr |
| clrtoeol () | clear to end of line on stdscr |
| cbreak () * | set cbreak mode |
| delay_output (*ms*) * | insert *ms* millisecond pause in output |
| delch () | delete a character |
| deleteln () | delete a line |
| delwin (*win*) | delete *win* |
| doupdate () | update screen from all wnooutrefresh |
| echo () * | set echo mode |
| endwin () * | end window modes |
| erase () | erase stdscr |
| erasechar () | return user's erase character |
| fixterm () | restore tty to "in curses" state |
| flash () | flash screen or beep |
| flushinp () * | throw away any typeahead |
| getch () * | get a char from tty |
| getstr (str) | get a string through stdscr |
| gettmode () | establish current tty modes |
| getyx (*win, y, x*) + | get (y,x) coordinates |

| | |
|---|---|
| has_ic() | true if terminal can do insert character |
| has_il() | true if terminal can do insert line |
| idlok (*win, bf*) * | use terminal's insert/delete line if *bf* != 0 |
| inch() | get char at current (y,x) coordinates |
| initscr()* | initialize screens |
| insch (*c*) | insert a char |
| insertln() | insert a line |
| intrflush (*win, bf*) | interrupts flush output if *bf* is TRUE |
| keypad (*win, bf*) | enable keypad input |
| killchar() | return current user's kill character |
| leaveok (*win, flag*) | OK to leave cursor anywhere after refresh if flag !=0 for *win*, otherwise cursor must be left at current position. |
| longname() | return verbose name of terminal |
| meta (*win, flag*) * | allow meta characters on input if flag != 0 |
| move (*y, x*) * | move to (y,x) on stdscr |
| mvaddch (*y, x, ch*) + | move(y,x) then addch (*ch*) |
| mvaddstr (*y, x, str*) + | move(y,x) then addstr(*str*) |
| mvcur (*oldrow, oldcol, newrow, newcol*) | low level cursor motion |
| mvdelch (*y, x*) + | like delch, but move(y,x) first |
| mvgetch (*y, x*) + | like getch, but move(y,x) first |
| mvgetstr (*y, x*) + | like getstr, but move(y,x) first |
| mvinch (*y, x*) + | like inch, but move(y,x) first |
| mvinsch (*y, x,* c) | like insch, but move(y,x) first |
| mvprintw (*y, x, fmt, args*) + | like printw, but move(y,x) first |
| mvscanw (*y, x, fmt, args*) | like scanw, but move(y,x) first |
| mvwaddch (*win, y, x, ch*) + | like addch, but move(y,x) first |
| mvwaddstr (*win, y, x, str*) + | like waddstr, but move(y,x) first |
| mvwdelch (*win, y, x*) + | like wdelch, but move(y,x) first |
| mvwgetch (*win, y, x*) + | like wgetch, but move(y,x) first |
| mvwgetstr (*win, y, x*) + | like wgetstr, but move(y,x) |
| mvwin (*win, by, bx*) | like win, but move(y,x) |
| mvwinch (*win, y, x*) + | like winch, but move(y,x) |
| mvwinsch (*win, y, x, c*) + | like winsch, but move(y,x) |
| mvwprintw (*win, y, x, fmt, args*) + | like wprintw, but move(y,x) |
| mvwscanw (*win, y, x, fmt, args*) + | like wscanw, but move(y,x) |
| newpad (*nlines, ncols*) | create a new pad with given dimensions |

| | |
|---|---|
| newterm (*type*, *fd*) | set up new terminal of given type to output on *fd* |
| newin (*lines*, *cols*, *begin_y*, *begin_x*) | create a new window |
| nl ()* | set newline mapping |
| nocbreak ()* | unset cbreak mode |
| nodelay (*win*, *bf*) | enable nodelay input mode through getch |
| noecho ()* | unset echo mode |
| nonl ()* | unset newline mapping |
| noraw ()* | unset raw mode |
| overlay (*win*1, *win*2) | overlay *win*1 on *win*2 |
| overwrite (*win*1, *win*2) | overwrite *win*1 on top of *win*2 |
| pnoutrefresh (*pad*, *pminrow*, *pmincol*, *sminrow*, *smincol*, *smaxrow*, *smaxcol*) | like prefresh but with no output until doupdate called |
| prefresh (*pad*, *pminrow*, *pmincol*, *sminrow*, *smincol*, *smaxrow*, *smaxcol*) | refresh from pad starting with given upper left corner of pad with output to given portion of screen |
| printw (*fmt arg1*, *arg2*, ...) | printf on stdscr |
| raw ()* | set raw mode |
| refresh ()* | make current screen look like stdscr |
| resetterm ()* | set tty modes to "out of curses" state |
| resetty ()* | reset tty flags to stored value |
| saveterm ()* | save current modes as "in curses" state |
| savetty ()* | store current tty flags |
| scanw (*fmt*, *arg1*, *arg2*, ...) | scanf through stdscr |
| scroll (*win*) | scroll *win* one line |
| scrollok (*win*, *flag*) | allow terminal to scroll if *flag* != 0 |
| set_term (*new*) | now talk to terminal *new* |
| setscrreg (*t*, *b*) | set user scrolling region to lines t through b |
| setterm (*type*) | establish terminal with given type |
| setupterm (*term*, *filenum*, *errret*) | |
| standend ()* | clear standout mode attribute |
| standout ()* | set standout mode attribute |
| subwin (*win*, *lines*, *cols*, *begin_y*, *begin_x*) | create a subwindow |
| touchwindow (*win*) | change all of *win* |

| | |
|---|---|
| `traceoff()` | turn off debugging trace output |
| `traceon()` | turn on debugging trace output |
| `typeahead(`*fd*`)` | use file descriptor *fd* to check typeahead |
| `unctrl(`*ch*`)*` | printable version of *ch* |
| `waddch(`*win*`,`*ch*`)` | add char to *win* |
| `waddstr(`*win*`,`*str*`)` | add string to *win* |
| `wattroff(`*win*`,`*attrs*`)` | turn off *attrs* in *win* |
| `wattron(`*win*`,`*attrs*`)` | turn on *attrs* in *win* |
| `wattrst(`*win*`,`*attrs*`)` | set attrs in *win* to *attrs* |
| `wclear(`*win*`)` | clear *win* |
| `wclrtobot(`*win*`)` | clear to bottom of *win* |
| `wclrtoeol(`*win*`)` | clear to end of line on *win* |
| `wdelch(`*win*`,`*c*`)` | delete char from *win* |
| `wdeleteln(`*win*`)` | delete line from *win* |
| `werase(`*win*`)` | erase *win* |
| `wgetch(`*win*`)` | get a char through *win* |
| `wgetstr(`*win*`,`*str*`)` | get a string through *win* |
| `winch(`*win*`)+` | get char at current (y,x) in *win* |
| `winsch(`*win*`,`*c*`)` | insert char into *win* |
| `winsertln(`*win*`)` | insert line into *win* |
| `wmove(`*win*`,`*y*`,`*x*`)` | set current (y,x) coordinates on *win* |
| `wnoutrefresh(`*win*`)` | refresh but no screen output |
| `wprintw(`*win*`,`*fmt*`,`*arg1*`,`*arg2*`, ...)` | `printf` on *win* |
| `wrefresh(`*win*`)` | make screen look like *win* |
| `wscanw(`*win*`,`*fmt*`,`*arg1fC*`,`*arg2*`, ...)` | `scanf` through *win* |
| `wsetscrreg(`*win*`,`*t*`,`*b*`)` | set scrolling region of *win* |
| `wstandend(`*win*`)` | clear standout attribute in *win* |
| `wstandout(`*win*`)` | set standout attribute in *win* |

## TERMINFO LEVEL ROUTINES

These routines should be called by programs wishing to deal directly with the `terminfo` database. Due to the low level of this interface, it is discouraged. Initially, `setupterm` should be called. This will define the set of terminal dependent variables defined in `terminfo`(4). The include files `<curses.h>` and `<term.h>` should be included to get the definitions for these strings, numbers, and flags. Parmeterized strings should be passed through `tparm` to instantiate them. All `terminfo` strings (including the output of `tparm`) should be printed with `tputs` or `putp`. Before exiting, `resetterm` should be called to restore the tty modes. (Programs desiring shell escapes or suspending with CONTROL-Z can call `resetterm` before the

shell is called and `fixterm` after returning from the shell.)

`fixterm()` restore tty modes for `terminfo` use
(called by `setupterm`)
`resetterm()` reset tty modes to state before program entry
`setupterm(`*term, fd, rc*`)`read in database. Terminal type is the
character string *term*, all output is to UNIX
System file descriptor *fd*. A status value is
returned in the integer pointed to by *rc*: 1
is normal. The simplest call would be
`setupterm(0, 1, 0)` which uses all defaults.
`tparm(`*str, p1, p2, ..., p9*`)`
instantiate string *str* with parms $p_i$.
`tputs(`*str, affcnt, putc*`)` apply padding info to string *str*.
*affcnt* is the number of lines affected,
or 1 if not applicable. *putc* is a
`putchar`-like function to which the characters
are passed, one at a time.
`putp(`*str*`)` handy function that calls `tputs`
`(`*str*`, 1, putchar)`
`vidputs(`*attrs, putc*`)` output the string to put terminal in video
attribute mode *attrs*, which is any
combination of the attributes listed below.
Chars are passed to `putchar`-like
function *putc*.
`vidattr(`*attrs*`)` Like `vidputs` but outputs through
`putchar`

## TERMCAP COMPATIBILITY ROUTINES

These routines were included as a conversion aid for programs that use termcap. Their parameters are the same as for `termcap`. They are emulated using the `terminfo` database. They may go away at a later date.

| | |
|---|---|
| `tgetent(`*bp, name*`)` | look up `termcap` entry for *name* |
| `tgetflag(`*id*`)` | get boolean entry for *id* |
| `tgetnum(`*id*`)` | get numeric entry for *id* |
| `tgetstr(`*id, area*`)` | get string entry for *id* |
| `tgoto(`*cap, col, row*`)` | apply parms to given *cap* |
| `tputs(`*cap, affcnt, fn*`)` | apply padding to *cap* calling *fn* as putchar |

## ATTRIBUTES

The following video attributes can be passed to the functions `attron, attroff, attrset`.

| | |
|---|---|
| A_STANDOUT | Terminal's best highlighting mode |
| A_UNDERLINE | Underlining |
| A_REVERSE | Reverse video |
| A_BLINK | Blinking |
| A_DIM | Half bright |
| A_BOLD | Extra bright or bold |
| A_BLANK | Blanking (invisible) |
| A_PROTECT | Protected |
| A_ALTCHARSET | Alternate character set |

## FUNCTION KEYS

The following function keys might be returned by `getch` if `keypad` has been enabled. Note that not all of these are currently supported, due to lack of definitions in `terminfo` or the terminal not transmitting a unique code when the key is pressed.

| NAME | VALUE | KEY NAME |
|---|---|---|
| KEY_BREAK | 0401 | break key (unreliable) |
| KEY_DOWN | 0402 | The four arrow keys ... |
| KEY_UP | 0403 | |
| KEY_LEFT | 0404 | |
| KEY_RIGHT | 0405 | ... |
| KEY_HOME | 0406 | Home key (upward+left arrow) |
| KEY_BACKSPACE | 0407 | backspace (unreliable) |
| KEY_F0 | 0410 | Function keys. Space for 64 is reserved. |
| KEY_F(*n*) | (KEY_F0+(*n*)) | Formula for *fn*. |
| KEY_DL | 0510 | Delete line |
| KEY_IL | 0511 | Insert line |
| KEY_DC | 0512 | Delete character |
| KEY_IC | 0513 | Insert char or enter insert mode |
| KEY_EIC | 0514 | Exit insert char mode |
| KEY_CLEAR | 0515 | Clear screen |
| KEY_EOS | 0516 | Clear to end of screen |
| KEY_EOL | 0517 | Clear to end of line |
| KEY_SF | 0520 | Scroll 1 line forward |
| KEY_SR | 0521 | Scroll 1 line backwards (reverse) |
| KEY_NPAGE | 0522 | Next page |
| KEY_PPAGE | 0523 | Previous page |
| KEY_STAB | 0524 | Set tab |
| KEY_CTAB | 0525 | Clear tab |
| KEY_CATAB | 0526 | Clear all tabs |
| KEY_ENTER | 0527 | Enter or send (unreliable) |
| KEY_SRESET | 0530 | soft (partial) reset (unreliable) |

| | | |
|---|---|---|
| KEY_RESET | 0531 | reset or hard reset (unreliable) |
| KEY_PRINT | 0532 | print or copy |
| KEY_LL | 0533 | home down or bottom (lower left) |

**WARNING**

The plotting library plot(3X) and the curses library curses(3X) both use the names erase() and move(). The curses versions are macros. If you need both libraries, put the plot(3X) code in a different source file than the curses(3X) code and/or #undef move() and erase() in the plot(3X)) code.

## NAME

curses5.0 – BSD-style screen functions with "optimal" cursor motion

## SYNOPSIS

cc [*flags*] *files* -lcurses5.0 -ltermcap [*libraries*]

## DESCRIPTION

These routines are a subset of the routines provided in the new curses library. They are provided for compatibility with programs that use the old curses and termcap libraries. These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen, and the user sets up an image of a new one. Then the refresh() tells the routines to make the current screen look like the new one. In order to initialize the routines, the routine initscr() must be called before any of the other routines that deal with windows and screens are used. The routine endwin() should be called before exiting.

## SEE ALSO

ioctl(2), curses(3X), getenv(3), termcap(4), terminfo(4), tty(4).

## FUNCTIONS

| | |
|---|---|
| addch (*ch*) | add a character to stdscr |
| addstr (*str*) | add a string to stdscr |
| box (*win,vert,hor*) | draw a box around a window |
| crmode() | set cbreak mode |
| clear() | clear stdscr |
| clearok (*scr,boolf*) | set clear flag for *scr* |
| clrtobot() | clear to bottom on stdscr |
| clrtoeol() | 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 |
| getch() | get a char through stdscr |
| getcap (*name*) | get terminal capability *name* |
| getstr (*str*) | get a string through stdscr |
| gettmode() | get tty modes |
| getyx (*win,y,x*) | get (*y,x*) co-ordinates |
| inch() | get char at current (*y,x*) co-ordinates |
| initscr() | initialize screens |

| | |
|---|---|
| `insch` (*c*) | insert a char |
| `insertln` () | insert a line |
| `leaveok` (*win,boolf*) | set leave flag for *win* |
| `longname` (*termbuf,name*) | get long name from *termbuf* |
| `move` (*y,x*) | move to (*y,x*) on `stdscr` |
| `mvcur` (*lasty,lastx,newy,newx*) | actually move cursor |
| `newwin` (*lines,cols,begin_y,begin_x*) | create a new window |
| `nl` () | set newline mapping |
| `nocrmode` () | unset cbreak mode |
| `noecho` () | unset echo mode |
| `nonl` () | unset newline mapping |
| `noraw` () | unset raw mode |
| `overlay` (*win1,win2*) | overlay *win1* on *win2* |
| `overwrite` (*win1,win2*) | overwrite *win1* on top of *win2* |
| `printw` (*fmt,arg1,arg2,...*) | printf on `stdscr` |
| `raw` () | set raw mode |
| `refresh` () | make current screen look like `stdscr` |
| `resetty` () | reset tty flags to stored value |
| `savetty` () | stored current tty flags |
| `scanw` (*fmt,arg1,arg2,...*) | `scanf` through `stdscr` |
| `scroll` (*win*) | scroll *win* one line |
| `scrollok` (*win,boolf*) | set scroll flag |
| `setterm` (*name*) | set term variables for name |
| `standend` () | end standout mode |
| `standout` () | start standout mode |
| `subwin` (*win,lines,cols,begin_y,begin_x*) | create a subwindow |
| `touchwin` (*win*) | change all of *win* |
| `unctrl` (*ch*) | printable version of *ch* |
| `waddch` (*win,ch*) | add char to *win* |
| `waddstr` (*win,str*) | add string to *win* |
| `wclear` (*win*) | clear *win* |
| `wclrtobot` (*win*) | clear to bottom of *win* |
| `wclrtoeol` (*win*) | clear to end of line on *win* |
| `wdelch` (*win,c*) | delete char from *win* |
| `wdeleteln` (*win*) | delete line from *win* |
| `werase` (*win*) | erase *win* |
| `wgetch` (*win*) | get a char through *win* |
| `wgetstr` (*win,str*) | get a string through *win* |
| `winch` (*win*) | get char at current (*y,x*) in *win* |
| `winsch` (*win,c*) | insert char into *win* |
| `winsertln` (*win*) | insert line into *win* |
| `wmove` (*win,y,x*) | set current (*y,x*) co-ordinates on *win* |
| `wprintw` (*win,fmt,arg1,arg2,...*) | `printf` on *win* |

| | |
|---|---|
| wrefresh (*win*) | make screen look like *win* |
| wscanw (*win,fmt,arg1,arg2,...*) | scanf through *win* |
| wstandend (*win*) | end standout mode on *win* |
| wstandout (*win*) | start standout mode on *win* |

## NAME

cuserid – get character login name of the user

## SYNOPSIS

```
#include <stdio.h>

char *cuserid(s)
char *s;
```

## DESCRIPTION

cuserid generates a character string representation of the login name of the owner of the current process. If *s* is a NULL pointer, this representation is generated in an internal static area, the address of which is returned. Otherwise, *s* is assumed to point to an array of at least L_cuserid characters; the representation is left in this array. The constant L_cuserid is defined in the <stdio.h> header file.

## RETURN VALUE

If the login name cannot be found, cuserid returns a NULL pointer; if *s* is not a NULL pointer, a null character (\0) is placed at *s*[0].

## SEE ALSO

getlogin(3C), getpwent(3C).

## BUGS

cuserid uses getpwnam(3C); thus the results of a user's call to the latter will be obliterated by a subsequent call to the former.

The name cuserid is rather a misnomer.

## NAME

dbminit, fetch, store, delete, firstkey, nextkey – data base subroutines

## SYNOPSIS

```
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

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`.

*key*s and *content*s 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))
```

RETURN VALUE

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`.

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 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

dial – establish an out-going terminal line connection

## SYNOPSIS

```
#include <dial.h>

int dial(call)
CALL call;

void undial(fd)
int fd;
```

## DESCRIPTION

dial returns a file descriptor for a terminal line open for read/write. The argument to dial is a CALL structure (defined in the <dial.h> header file).

When finished with the terminal line, the calling program must invoke undial to release the semaphore that has been set during the allocation of the terminal device.

The CALL typedef in the <dial.h> header file is:

```
typedef struct {
      struct termio *attr;  /* pointer to termio
                               attribute struct */
      int           baud;   /* transmission data rate */
      int           speed;  /* 212A modem: low=300,
                               high=1200 */
      char          *line;  /* device name for
                               out-going line */
      char          *telno  /* pointer to tel-no digits
                               string */
      int           modem;  /* specify modem control for
                               direct lines */
      char         *device; /* Will hold the name of the
                               device used to make a
                               connection */
      int           dev_len /* The length of the device
                               used to make connection */
} CALL;
```

The CALL element speed is intended only for use with an out-going dialed call, in which case its value should be either 300 or 1200 to identify the 113A modem, or the high-speed or low-speed setting on the 212A modem. Note that the 113A modem or the low-speed setting of the 212A modem will transmit at any rate between 0 and 300 bits per second. However, the high-speed

setting of the 2121 modem transmits and receives at 1200 bits per second only. The CALL element baud is for the desired transmission baud rate. For example, one might set baud to 110 and speed to 300 (or 1200). However, if speed is set to 1200 baud must be set to high (1200).

If the desired terminal line is a direct line, a string pointer to its device name should be placed in the line element in the CALL structure. Legal values for such terminal device names are kept in the L-devices file. In this case, the value of the baud element need not be specified as it will be determined from the L-devices file.

The telno element is for a pointer to a character string representing the telephone number to be dialed. The termination symbol will be supplied by the dial function, and should not be included in the telno string passed to dial in the CALL structure.

The CALL element modem is used to specify modem control for direct lines. This element should be nonzero if modem control is required. The CALL element *attr* is a pointer to a termio structure, as defined in the <termio.h> header file. A NULL value for this pointer element may be passed to the dial function, but if such a structure is included, the elements specified in it will be set for the outgoing terminal line before the connection is established. This is important for attributes such as parity and baud rate.

The CALL element device is used to hold the device name (cul..) that establishes the connection.

The CALL element dev_len is the length of the device name that is copied into the array device.

ERRORS

On failure, a negative value indicating the reason for the failure is returned. Mnemonics for these negative indices as listed here are defined in the <dial.h> header file.

```
INTRPT  -1  /* interrupt occurred */
D_HUNG  -2  /* dialer hung (no return from write) */
NO_ANS  -3  /* no answer within 10 seconds */
ILL_BD  -4  /* illegal baud-rate */
A_PROB  -5  /* acu problem (open() failure) */
L_PROB  -6  /* line problem (open() failure) */
NO_Ldv  -7  /* can't open LDEVS file */
```

```
DV_NT_A -8  /* requested device not available */
DV_NT_K -9  /* requested device not known */
NO_BD_A -10 /* no device available at requested baud */
NO_BD_K -11 /* no device known at requested baud */
```

**FILES**

/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..*tty-device*

**SEE ALSO**

uucp(1C), alarm(2), read(2), write(2), termio(7).

**WARNINGS**

Including the <dial.h> header file automatically includes the <termio.h> header file.

Because the above routine uses <stdio.h>, the size of programs not otherwise using standard I/O is increased more than might be expected.

**BUGS**

An alarm(2) system call for 3,600 seconds is made (and caught) within the dial module for the purpose of "touching" the LCK.. file and constitutes the device allocation semaphore for the terminal device. Otherwise, uucp(1C) may simply delete the LCK.. entry on its 90-minute clean-up rounds. The alarm may go off while the user program is in a read(2) or write(2) system call, causing an apparent error return. If the user program is to run for an hour or more, error returns from reads should be checked for (errno==EINTR), and the read possibly reissued.

**NAME**

dim, ddim, idim – Fortran positive difference intrinsic functions

**SYNOPSIS**

```
integer a1, a2, a3
a3=idim(a1, a2)

real a1, a2, a3
a3=dim(a1, a2)

double precision a1, a2, a3
a3=ddim(a1, a2)
```

**DESCRIPTION**

These functions return:

| | |
|---|---|
| *a1-a2* | if *a1* > *a2* |
| 0 | if *a1* <= *a2* |

## NAME

opendir, readdir, telldir, seekdir, rewinddir, closedir – directory operations

## SYNOPSIS

```
#include <sys/dir.h>

DIR *opendir(filename)
char *filename;

struct direct *readdir(dirp)
DIR *dirp;

long telldir(dirp)
DIR *dirp;

seekdir(dirpb loc)
DIR *dirp;
long loc;

int rewinddir(dirp)
DIR *dirp;

int 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.

The `rewinddir` macro 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 searchs 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**

`ls(1)`, `open(2)`, `close(2)`, `getdirentries(2)`, `read(2)`, `lseek(2)`, `dir(4)`.

**NAME**

dprod – Fortran double precision product intrinsic function

**SYNOPSIS**

```
real a1, a2
double precision a3
a3=dprod(a1, a2)
```

**DESCRIPTION**

dprod returns the double precision product of its real arguments.

## NAME

drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 – generate uniformly distributed pseudo-random numbers

## SYNOPSIS

```
double drand48()

double erand48(xsubi)
unsigned short xsubi[3];

long lrand48()

long nrand48(xsubi)
unsigned short xsubi[3];

long mrand48()

long jrand48(xsubi)
unsigned short xsubi[3];

void srand48(seedval)
long seedval;

unsigned short *seed48(seed16v)
unsigned short seed16v[3];

void lcong48(param)
unsigned short param[7];
```

## DESCRIPTION

This family of functions generates pseudo-random numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic.

Functions drand48 and erand48 return non-negative double-precision floating-point values uniformly distributed over the interval [0.0, 1.0).

Functions lrand48 and nrand48 return non-negative long integers uniformly distributed over the interval $[0, 2^{31})$.

Functions mrand48 and jrand48 return signed long integers uniformly distributed over the interval $[-2^{31}, 2^{31})$. Functions srand48,seed48, and lcong48 are initialization entry points, one of which should be invoked before drand48,lrand48, or mrand48 is called. (Although it is not recommended practice, constant default initializer values are supplied automatically if drand48,lrand48, or mrand48 is called without a prior call to an initialization entry point.) Functions erand48, nrand48, and jrand48 do not require an

initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, $X_i$, according to the linear congruential formula

$$X_{n+1} = (aX + c)_{\bmod m} \qquad n \geq 0$$

The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless `lcong48` has been invoked, the multiplier value $a$ and the addend value $c$ are given by

$$a = 5DEECE66D_{16} = 273673163155_8$$
$$c = B_{16} = 13_8.$$

The value returned by any of the functions `drand48`, `erand48`, `lrand48`, `nrand48`, `mrand48`, or `jrand48` is computed by first generating the next 48-bit $X_i$ in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of $X_i$ and transformed into the returned value.

The functions `drand48`, `lrand48`, and `mrand48` store the last 48-bit $X_i$ generated in an internal buffer; that is why they must be initialized prior to being invoked. The functions `erand48`, `nrand48`, and `jrand48` require the calling program to provide storage for the successive $X_i$ values in the array specified as an argument when the functions are invoked. That is why these routines do not have to be initialized; the calling program merely has to place the desired initial value of $X_i$ into the array and pass it as an argument. By using different arguments, functions `erand48`, `nrand48`, and `jrand48` allow separate modules of a large program to generate several *independent* streams of pseudo-random numbers, i.e., the sequence of numbers in each stream does *not* depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function `srand48` sets the high-order 32 bits of $X_i$ to the 32 bits contained in its argument. The low-order 16 bits of $X_i$ are set to the arbitrary value $330E_{16}$.

The initializer function `seed48` sets the value of $X_i$ to the 48-bit value specified in the argument array. The previous value of $X_i$ is copied into a 48-bit internal buffer, used only by `seed48`. A pointer to this buffer is the value returned by `seed48`. The returned pointer, which can be ignored if not needed, is useful if a program is to be restarted from a given point at some future time. Use the pointer to get and store the last $X_i$ value; then use this value to reinitialize via `seed48` when the program is restarted.

The initialization function `lcong48` allows the user to specify the initial $X_i$, the multiplier value $a$, and the addend value $c$. Argument array elements *param*[0-2] specify $X_i$, elements *param*[3-5] specify the multiplier $a$, and *param*[6] specifies the 16-bit addend $c$. After `lcong48` has been called, a subsequent call to either `srand48` or `seed48` will restore the "standard" multiplier and addend values, $a$ and $c$, specified on the previous page.

NOTES

The routines are coded in portable C. The source code for the portable version can even be used on computers which do not have floating-point arithmetic. In such a situation, functions `drand48` and `erand48` do not exist; instead, they are replaced by the following two functions:

```
long irand48 (m)
unsigned short m;

long krand48 (xsubi,m)
unsigned short xsubi[3],m;
```

Functions `irand48` and `krand48` return non-negative long integers uniformly distributed over the interval [0, $m$–1].

SEE ALSO

`rand`(3C).

**NAME**

dup2 – duplicate a descriptor

**SYNOPSIS**

```
dup2(oldd, newd)
int oldd, newd;
```

**DESCRIPTION**

dup2 causes *newd* to become a duplicate of *oldd*. If *newd* is already in use, the descriptor is first deallocated as if a close(2) call had been done first.

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. If a separate pointer into the file is desired, a different object reference to the file must be obtained by issuing an additional open(2) call.

**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**

dup2 fails if:

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

**SEE ALSO**

accept(2N), close(2), dup(2), fcntl(2), getdtablesize(2N), open(2), pipe(2), socket(2N).

## NAME

ecvt, fcvt, gcvt – convert floating-point number to string

## 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;
int ndigit;
char *buf;
```

## DESCRIPTION

ecvt converts *value* to a null-terminated string of *ndigit* digits and returns a pointer to this string. The high-order digit is non-zero, unless the value is zero. The low-order digit is rounded. 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). The decimal point is not included in the returned string. If the sign of the result is negative, the word pointed to by *sign* is non-zero; otherwise it is zero.

fcvt is identical to ecvt, except that the correct digit has been rounded for printf "%f" (Fortran F-format) output of the number of digits specified by *ndigit*.

gcvt converts the *value* to a null-terminated string in the array pointed to by *buf* and returns *buf*. It attempts to produce *ndigit* significant digits in Fortran F-format, ready for printing; E-format is produced when F-format is not possible. A minus sign, if there is one, or a decimal point is included as part of the returned string. Trailing zeros are suppressed.

## SEE ALSO

printf(3S).

## BUGS

The values returned by ecvt and fcvt point to a single static data array.

## 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 (the first location beyond the data) coincides with end, but the program break may be reset by the routines of brk(2), malloc(3C), standard input/output, the profile (–p) option of cc(1), and so on. Thus, the current value of the program break should be determined by sbrk(0) (see brk(2)).

## SEE ALSO

cc(1), brk(2), intro(3), malloc(3C).

## NAME

erf, erfc – error function and complementary error function

## SYNOPSIS

```
#include <math.h>

double erf(x)
double x;

double erfc(x)
double x;
```

## DESCRIPTION

The erf function returns the error function of *x* (the precise formula is available in at standard calculus text).

erfc, which returns 1.0 – erf(*x*), is provided because of the extreme loss of relative accuracy if erf(*x*) is called for large *x* and the result subtracted from 1.0 (e.g. for *x* = 5, 12 places are lost).

## SEE ALSO

exp(3M).

## NAME

exp, dexp, cexp – Fortran exponential intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2
complex cx1, cx2

r2=exp(r1)

dp2=dexp(dp1)
dp2=exp(dp1)

cx2=cexp(cx1)
cx2=exp(cx1)
```

## DESCRIPTION

exp returns the real exponential function $e^x$ of its real argument. dexp returns the double-precision exponential function of its double-precision argument. cexp returns the complex exponential function of its complex argument. The generic function exp becomes a call to dexp or cexp, as required, depending on the type of its argument.

## SEE ALSO

exp(3M).

## NAME

exp, log, log10, pow, sqrt – exponential, logarithm, power, square root functions

## SYNOPSIS

```
#include <math.h>

double exp(x)
double x;

double log(x)
double x;

double log10(x)
double x;

double pow(x, y)
double x, y;

double sqrt(x)
double x;
```

## DESCRIPTION

The exp function returns e raised to the power of x.

log returns the natural logarithm of *x*. The value of *x* must be positive.

log10 returns the logarithm base ten of *x*. The value of *x* must be positive.

The pow function returns x raised to the power of y. If *x* is zero, *y* must be positive. If *x* is negative, *y* must be an integer.

sqrt returns the nonnegative square root of *x*. The value of *x* may not be negative.

## RETURN VALUE

exp returns HUGE when the correct value would overflow, or 0 when the correct value would underflow, and sets errno to ERANGE.

log and log10 return –HUGE and set errno to EDOM when *x* is nonpositive. A message indicating DOMAIN error (or SING error when *x* is 0) is printed on the standard error output.

pow returns 0 and sets errno to EDOM when *x* is 0 and *y* is nonpositive, or when *x* is negative and *y* is not an integer. In these cases a message indicating DOMAIN error is printed on the standard error output. When the correct value for pow would overflow or underflow, pow returns ±HUGE or 0 respectively, and sets errno to ERANGE.

`sqrt` returns 0 and sets `errno` to `EDOM` when $x$ is negative. A message indicating `DOMAIN` error is printed on the standard error output.

These error-handling procedures may be changed with the function `matherr`(3M).

**SEE ALSO**

`intro`(2), `hypot`(3M), `matherr`(3M), `sinh`(3M).

## NAME

fclose, fflush – close or flush a stream

## SYNOPSIS

```
#include <stdio.h>

int fclose(stream)
FILE *stream;

int fflush(stream)
FILE *stream;
```

## DESCRIPTION

fclose causes any buffered data for the named *stream* to be written out and the *stream* to be closed.

fclose is performed automatically for all open files upon calling exit(2).

fflush causes any buffered data for the named *stream* to be written to that file. The *stream* remains open.

## RETURN VALUE

These functions return 0 for success, and EOF if any error (such as trying to write to a file that has not been opened for writing) was detected.

## SEE ALSO

close(2), exit(2), fopen(3S), setbuf(3S).

## NAME

ferror, feof, clearerr, fileno – stream status inquiries

## SYNOPSIS

```
#include <stdio.h>

int feof(stream)
FILE *stream;

int ferror(stream)
FILE *stream;

void clearerr(stream)
FILE *stream;

int fileno(stream)
FILE *stream;
```

## DESCRIPTION

feof returns nonzero when EOF has previously been detected reading the named input *stream*; otherwise, it returns zero.

ferror returns nonzero when an I/O error has previously occurred reading from or writing to the named *stream*; otherwise, it returns zero.

clearerr resets the error indicator and EOF indicator to zero on the named *stream*.

fileno returns the integer file descriptor associated with the named *stream*; see open(2).

## NOTE

All these functions are implemented as macros; they cannot be declared or redeclared.

## SEE ALSO

open(2), fopen(3S).

## NAME

floor, ceil, fmod, fabs – floor, ceiling, remainder, absolute value functions

## SYNOPSIS

```
#include <math.h>

double floor(x)
double x;

double ceil(x)
double x;

double fmod(x, y)
double x, y;

double fabs(x)
double x;
```

## DESCRIPTION

floor returns the largest integer (as a double-precision number) not greater than $x$.

ceil returns the smallest integer not less than $x$.

returns the floating-point remainder of the division of $x$ by $y$: zero if $y$ is zero or if $x/y$ would overflow; otherwise the number $f$ with the same sign as $x$, such that $x = iy + f$ for some integer $i$, and $|f| < |y|$.

fabs returns the absolute value of $|x|$.

## SEE ALSO

abs(3C).

## 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)
int fildes;
char *type;
```

## DESCRIPTION

fopen opens the file named by *filename* and associates a *stream* with it. fopen returns a pointer to the FILE structure associated with the *stream*.

*filename* points to a character string that contains the name of the file to be opened.

*type* is a character string having one of the following values:

| | |
|---|---|
| r | open for reading |
| w | truncate or create for writing |
| a | append; open for writing at end of file, or create for writing |
| r+ | open for update (reading and writing) |
| w+ | truncate or create for update |
| a+ | append; open or create for update at end-of-file |

freopen substitutes the named file in place of the open *stream*. The original *stream* is closed, regardless of whether the open ultimately succeeds. freopen returns a pointer to the FILE structure associated with *stream*.

freopen is typically used to attach the preopened *streams* associated with stdin, stdout, and stderr to other files.

fdopen associates a *stream* with a file descriptor by formatting a file structure from the file descriptor. Thus, fdopen can be used to access the file descriptors returned by open(2), dup(2), creat(2), or pipe(2). (These calls open files but do not return pointers to a FILE structure.) The *type* of *stream* must agree

with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting *stream*. However, output may not be directly followed by input without an intervening `fseek` or `rewind`, and input may not be directly followed by output without an intervening `fseek`, `rewind`, or an input operation which encounters end-of-file.

When a file is opened for append (i.e., when *type* is "a" or "a+" it is impossible to overwrite information already in the file. `fseek` may be used to reposition the file pointer to any position in the file, but when output is written to the file the current file pointer is disregarded. All output is written at the end of the file and causes the file pointer to be repositioned at the end of the output. If two separate processes open the same file for append, each process may write freely to the file without fear of destroying output being written by the other. The output from the two processes will be intermixed in the file in the order in which it is written.

**RETURN VALUE**

`fopen` and `freopen` return a NULL pointer on failure.

**SEE ALSO**

`creat`(2), `dup`(2), `open`(2), `pipe`(2), `fclose`(3S), `fseek`(3S).

## NAME

fread, fwrite – binary input/output

## SYNOPSIS

```
#include <stdio.h>

int fread(ptr, size, nitems, stream)
char *ptr;
int  size, nitems;
FILE *stream;

int fwrite(ptr, size, nitems, stream)
char *ptr;
int size, nitems;
FILE *stream;
```

## DESCRIPTION

fread copies *nitems* items of data from the named input *stream* into an array beginning at *ptr*. An item of data is a sequence of bytes (not necessarily terminated by a null byte) of length *size*. fread stops appending bytes if an end-of-file or error condition is encountered while reading *stream* or if *nitems* items have been read. fread leaves the file pointer in *stream*, if defined, pointing to the byte following the last byte read if there is one. fread does not change the contents of *stream*.

fwrite appends at most *nitems* items of data from the the array pointed to by *ptr* to the named output *stream*. fwrite stops appending when it has appended *nitems* items of data or if an error condition is encountered on *stream*. fwrite does not change the contents of the array pointed to by *ptr*.

The variable *size* is typically *sizeof*(**ptr*) where the pseudo-function *sizeof* specifies the length of an item pointed to by *ptr*. If *ptr* points to a data type other than *char* it should be cast into a pointer to *char*.

## RETURN VALUE

fread and fwrite return the number of items read or written. If *size* or *nitems* is non-positive, no characters are read or written and 0 is returned by both fread and fwrite.

## SEE ALSO

read(2), write(2), fopen(3S), getc(3S), gets(3S), printf(3S), putc(3S), puts(3S), scanf(3S).

## NAME

frexp, ldexp, modf – manipulate parts of floating-point numbers

## SYNOPSIS

```
double frexp(value, eptr)
double value;
int *eptr;

double ldexp(value, exp)
double value;
int exp;

double modf(value, iptr)
double value, *iptr;
```

## DESCRIPTION

Every nonzero number can be written uniquely as $x$* pow(2,$n$), where the "mantissa" (fraction) $x$ is in the range $0.5 \le |x| < 1.0$, and the "exponent" $n$ is an integer. frexp returns the mantissa of a double *value*, and stores the exponent indirectly in the location pointed to by *eptr*. If *value* is zero, both results returned by frexp are zero.

ldexp returns the quantity *value** pow(2,*exp*).

modf returns the signed fractional part of *value* and stores the integral part indirectly in the location pointed to by *iptr*.

## ERRORS

If ldexp would cause overflow, ± HUGE is returned (according to the sign of *value*), and errno is set to ERANGE.

If ldexp would cause underflow, zero is returned and errno is set to ERRANGE.

## SEE ALSO

exp(3M).

## NAME

fseek, rewind, ftell – reposition a file pointer in a stream

## SYNOPSIS

```
#include <stdio.h>

int fseek(stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

void rewind(stream)
FILE *stream;

long ftell(stream)
FILE *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, when the value of *ptrname* is 0, 1, or 2, respectively.

rewind(*stream*) is equivalent to fseek(*stream*, 0L, 0), except that no value is returned.

fseek and rewind undo any effects of ungetc(3S).

After fseek or rewind, the next operation on a file opened for update may be either input or output.

ftell returns the offset of the current byte relative to the beginning of the file associated with the named *stream*.

## RETURN VALUE

fseek returns non-zero for improper seeks; otherwise it returns zero.

An improper seek can be, for example, an fseek done on a file that has not been opened via fopen; in particular, fseek may not be used on a terminal or on a file opened via popen(3S).

## SEE ALSO

lseek(2), fopen(3S), popen(3S), ungetc(3S).

**WARNING**

On A/UX an offset returned by `ftell` is measured in bytes, and it is permissible to seek to positions relative to that offset; however, portability to systems other than A/UX requires that an offset be used by `fseek` directly. Arithmetic may not meaningfully be performed on such an offset, which is not necessarily measured in bytes.

## NAME

ftok – standard interprocess communication package

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(path, id)
char *path;
char id;
```

## DESCRIPTION

All interprocess communication facilities require the user to supply a key to be used by the msgget(2), semget(2), and shmget(2) system calls to obtain interprocess communication identifiers. One method for forming a key is to use the ftok subroutine described below. Another way to compose keys is to include the project ID in the most significant byte and to use the remaining portion as a sequence number. There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If a standard is not adhered to, unrelated processes may interfere with each other's operation. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys do not conflict across a given system.

ftok returns a key based on *path* and *id* that is usable in subsequent msgget, semget, and shmget system calls. *path* must be the pathname of an existing file that is accessible to the process. *id* is a character that uniquely identifies a project. ftok returns the same key for linked files when called with the same *id*; it returns different keys when called with the same filename but different *id*s.

## SEE ALSO

intro(2), msgget(2), semget(2), shmget(2).

## DIAGNOSTICS

ftok returns (key_t) −1 if *path* does not exist or if it is not accessible to the process.

## WARNING

If the file whose *path* is passed to ftok is removed when keys still refer to the file, future calls to ftok with the same *path* and *id* will return an error. If the same file is recreated, ftok is likely to return a different key than it did the original time it was called.

## NAME

ftw – walk a file tree

## SYNOPSIS

```
#include <ftw.h>

int ftw(path, fn, depth)
char *path;
int (*fn) ();
int depth;
```

## DESCRIPTION

ftw recursively descends the directory hierarchy rooted in *path*. For each object in the hierarchy, ftw calls *fn*, passing it a pointer to a nullterminated character string containing the name of the object, a pointer to a stat structure (see stat(2)) containing information about the object, and an integer. Possible values of the integer, defined in the <ftw.h> header file, are FTW_F for a file, FTW_D for a directory, FTW_DNR for a directory that cannot be read, and FTW_NS for an object for which stat could not be executed successfully. If the integer is FTW_DNR, descendants of that directory will not be processed. If the integer is FTW_NS, the stat structure will contain garbage. An example of an object that would cause FTW_NS to be passed to *fn* is a file in a directory with read permission but not execute (search) permission.

ftw visits a directory before visiting any of its descendants.

The tree traversal continues until the tree is exhausted, an invocation of *fn* returns a nonzero value, or an error is detected within ftw (such as an I/O error). If the tree is exhausted, ftw returns zero. If *fn* returns a nonzero value, ftw stops its tree traversal and returns whatever value was returned by *fn*. If ftw detects an error, it returns −1, and sets the error type in errno.

ftw uses one file descriptor for each level in the tree. The *depth* argument limits the number of file descriptors so used. If *depth* is zero or negative, the effect is the same as if it were 1. *depth* must not be greater than the number of file descriptors currently available for use. ftw runs more quickly if *depth* is at least as large as the number of levels in the tree.

## RETURN VALUE

The tree traversal continues until the tree is exhausted, and invocation of *fn* returns a nonzero value or some error is detected within ftw (such as an I/O error). If the tree is exahusted, ftw returns 0. If *fn* returns a nonzero value, ftw stops its tree

traversal and returns whatever value was returned by *fn*.

If `ftw` encounters an error other than `EACCESS`, it returns –1 and `errno` is set to indicate the error. The external variable `errno` may contain the error values that are possible when a directory is opened or when `stat`(2) is executed on a directory or file.

**SEE ALSO**

`stat`(2), `malloc`(3C).

**BUGS**

Because `ftw` is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

`ftw` could be made to run faster and use less storage on deep structures at the cost of considerable complexity.

`ftw` uses `malloc`(3C) to allocate dynamic storage during its operation. If `ftw` is forcibly terminated, such as by `longjmp` being executed by *fn* or an interrupt routine, `ftw` does not have a chance to free that storage, so it remains permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have *fn* return a nonzero value at its next invocation.

**NAME**

int, ifix, idint, real, float, sngl, dble, cmplx, dcmplx, ichar, char – explicit Fortran type conversion

**SYNOPSIS**

```
integer i, j
real r, s
double precision dp, dq
complex cx
double complex dcx
character *1 ch

i=int(r)
i=int(dp)
i=int(cx)
i=int(dcx)
i=ifix(r)
i=idint(dp)

r=real(i)
r=real(dp)
r=real(cx)
r=real(dcx)
r=float(i)
r=sngl(dp)

dp=dble(i)
dp=dble(r)
dp=dble(cx)
dp=dble(dcx)

cx=cmplx(i)
cx=cmplx(i, j)
cx=cmplx(r)
cx=cmplx(r, s)
cx=cmplx(dp)
cx=cmplx(dp, dq)
cx=cmplx(dcx)

dcx=dcmplx(i)
dcx=dcmplx(i, j)
dcx=dcmplx(r)
dcx=dcmplx(r, s)
dcx=dcmplx(dp)
dcx=dcmplx(dp, dq)
dcx=dcmplx(cx)
```

*i*=ichar(*ch*)
*ch*=char(*i*)

DESCRIPTION

These functions perform conversion from one data type to another.

int converts to integer form its real, double precision, complex, or double complex argument. If the argument is real or double precision, int returns the integer whose magnitude is the largest integer that does not exceed the magnitude of the argument and whose sign is the same as the sign of the argument (i.e., truncation). For complex types, the above rule is applied to the real part. ifix and idint convert only real and double precision arguments respectively.

real converts to real form an integer, double precision, complex, or double complex argument. If the argument is *double precision* or *double complex*, as much precision is kept as is possible. If the argument is one of the complex types, the real part is returned. float and sngl convert only integer and double precision arguments, respectively.

dble converts any integer, real, complex, or double complex argument to double precision form. If the argument is of a complex type, the real part is returned.

cmplx converts its integer, real, double precision, or double complex argument(s) to complex form.

dcmplx converts its integer, real, double precision, or complex argument(s) to double complex form.

Either one or two arguments may be supplied to cmplx and dcmplx. If there is only one argument, it is taken as the real part of the complex type and a imaginary part of zero is supplied. If two arguments are supplied, the first is taken as the real part and the second as the imaginary part.

ichar converts from a character to an integer depending on the character's position in the collating sequence.

char returns the character in the *i*th position in the processor collating sequence, where *i* is the supplied argument.

For a processor capable of representing *n* characters,

ichar(char(*i*)) = *i* for $0 <= i < n$, and

char(ichar(*ch*)) = *ch* for any representable character *ch*.

## NAME

gamma – log gamma function

## SYNOPSIS

```
#include <math.h>

extern int signgam;

double gamma(x)
double x;
```

## DESCRIPTION

gamma returns the natural log of gamma as a function of the absolute value of a given value. gamma returns $\ln(|\Gamma(x)|)$, where $\Gamma(x)$ is defined as

$$\int_0^\infty e^{-t} t^{x-1} dt .$$

The sign of $\Gamma(x)$

is returned in the external integer signgam. The argument $x$ may not be a nonpositive integer.

The following C program fragment might be used to calculate $\Gamma$:

```
if ((y = gamma(x)) > LN_MAXDOUBLE)
        error();
y = signgam * exp(y);
```

where LN_MAXDOUBLE is the least value that causes exp(3M) to return a range error, and is defined in the <values.h> header file.

## RETURN VALUE

For non-negative integer arguments HUGE is returned, and errno is set to EDOM. A message indicating SING error is printed on the standard error output.

If the correct value would overflow, gamma returns HUGE and sets errno to ERANGE.

These error-handling procedures may be changed with the function matherr(3M).

## SEE ALSO

exp(3M), matherr(3M), values(5).

**NAME**

getarg – return Fortran command-line argument

**SYNOPSIS**

```
character*N c
integer i
getarg(i, c)
```

**DESCRIPTION**

getarg returns the *i*th command-line argument of the current process. Thus, if a program were invoked with:

foo *arg*1 *arg*2 *arg*3

getarg(2, c) would return the string *arg2* in the character variable *c*.

**SEE ALSO**

getopt(3C).

## NAME

getc, getchar, fgetc, getw – get character or word from a stream

## SYNOPSIS

```
#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar()

int fgetc(stream)
FILE *stream;

int getw(stream)
FILE *stream;
```

## DESCRIPTION

The `getc` macro returns the next character (i.e., byte) from the named input *stream*, as an integer. It also moves the file pointer, if defined, ahead one character in *stream*. The `getchar` macro is defined as `getc(stdin)`.

`fgetc` behaves like `getc`, but is a function rather than a macro. `fgetc` runs more slowly than `getc`, but takes less space per invocation and its name can be passed as an argument to a function.

`getw` returns the next word (32-bit integer on a Macintosh II) from the named input *stream*. `getw` increments the associated file pointer, if defined, to point to the next word. `getw` assumes no special alignment in the file.

## RETURN VALUE

These functions return the constant EOF at end-of-file or upon an error. Because EOF is a valid integer, `ferror`(3S) should be used to detect `getw` errors.

## SEE ALSO

`fclose`(3S), `ferror`(3S), `fopen`(3S), `fread`(3S), `gets`(3S), `putc`(3S), `scanf`(3S), `ungetc`(3S).

## WARNING

If the integer value returned by `getc`, `getchar`, or `fgetc` is stored into a character variable and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a character on widening to integer is machine-dependent.

BUGS

Because it is implemented as a macro, `getc` treats incorrectly a *stream* argument with side effects. In particular, `getc(*f++)` does not work sensibly. `fgetc` should be used instead.
Because of possible differences in word length and byte ordering, files written using `putw` are machine-dependent, and may not be read using `getw` on a different processor.

## NAME

getcwd – get pathname of current working directory

## SYNOPSIS

```
char *getcwd(buf, size)
char *buf;
int size;
```

## DESCRIPTION

getcwd returns a pointer to the current directory pathname. The value of *size* must be at least two greater than the length of the pathname to be returned.

If *buf* is a NULL pointer, getcwd obtains *size* bytes of space using malloc(3C). In this case, the pointer returned by getcwd may be used as the argument in a subsequent call to free.

The function is implemented by using popen(3S) to pipe the output of the pwd(1) command into the specified string space.

## EXAMPLE

```
char *cwd, *getcwd();
.
.
.
if ((cwd=getcwd((char *)NULL, 64))==NULL) {
    perror(''pwd'');
    exit(1);
}
printf(''%s\n'', cwd);
```

## RETURN VALUE

Returns NULL with errno set if *size* is not large enough, or if an error occurs in a lower-level function.

## SEE ALSO

pwd(1), malloc(3C), popen(3S).

**NAME**

getenv – return value for environment name

**SYNOPSIS**

```
char  *getenv(name)
char  *name;
```

**DESCRIPTION**

getenv searches the environment list (see environ(5)) for a string of the form *name=value*, and returns a pointer to the *value* in the current environment if such a string is present; otherwise a NULL pointer is returned.

**SEE ALSO**

exec(2), putenv(3C), environ(5).

**NAME**

getenv – return Fortran environment variable

**SYNOPSIS**

```
character *N c
getenv(tmpdir, c)
```

**DESCRIPTION**

getenv returns the character-string value of the environment variable represented by its first argument into the character variable of its second argument. If no such environment variable exists, all blanks are returned.

**SEE ALSO**

getenv(3C), environ(5).

## NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent – obtain group file entry from a group file

## SYNOPSIS

```
#include <grp.h>

struct group *getgrent()

struct group *getgrgid(gid)
int gid;

struct group *getgrnam(name)
char *name;

void setgrent()

struct group *fgetgrent(f)
FILE *f;

void 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 /etc/group file. Each line contains a group structure, defined in the <grp.h> header file.

```
struct group {
        char    *gr_name;    /* the name of the group */
        char    *gr_passwd;  /* the encrypted group
                                password */
        int      gr_gid;     /* the numeric group ID */
        char    **gr_mem;    /* vector of pointers to
                                member names */
};
```

When first called, getgrent returns a pointer to the first group structure in the file; thereafter, it returns a pointer to the next group structure in the file; therefore, successive calls may be used to search the entire file. getgrgid searches from the beginning of the file until a numeric group ID matching *gid* is found; it returns a pointer to the particular structure in which the match was found. getgrnam searches from the beginning of the file until a group name matching *name* is found; it returns a pointer to the particular structure in which the match was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

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.

`fgetgrent` returns a pointer to the next group structure in the stream *f*, which matches the format of `/etc/group`.

**RETURN VALUE**

A NULL pointer is returned on EOF or error.

**FILES**

`/etc/group`

**SEE ALSO**

`getlogin`(3C), `getpwent`(3C), `group`(4).

**WARNING**

The above routines use `<stdio.h>`. This causes them to increase the size of programs not otherwise using standard I/O more than might be expected.

**BUGS**

All information is contained in a static area, so it must be copied if it is to be saved.

## NAME

gethostent, gethostbyaddr, gethostbyname, sethostent, endhostent – get network host entry

## SYNOPSIS

```
#include <netdb.h>

struct hostent *gethostent()

struct hostent *gethostbyname(name)
char *name;

struct hostent *gethostbyaddr(addr, len, type)
char *addr;
int len, type;

int sethostent(stayopen)
int stayopen

int endhostent()
```

## DESCRIPTION

gethostent, gethostbyname, and gethostbyaddr each return a pointer to an object with the following structure containing the broken-out fields of a line in the network host data base, /etc/hosts.

```
struct hostent {
    char  *h_name;        /*official name of host*/
    char  **h_aliases;    /*alias list*/
    int   h_addrtype;     /*address type*/
    int   h_length;       /*length of address*/
    char  *h_addr;        /*address*/
};
```

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 | A pointer to the network address for the host. Host addresses are returned in network byte order. |

`gethostent` reads the next line of the file, opening the file if necessary.

`sethostent` opens and rewinds the file. If the *stayopen* flag is non-zero, the host data base will not be closed after each call to `gethostent` (either directly, or indirectly through one of the other "gethost" calls).

`endhostent` closes the file.

`gethostbyname` and `gethostbyaddr` sequentially search from the beginning of the file until a matching host name or host address is found, or until EOF is encountered. Host addresses are supplied in network order.

**RETURN VALUE**

NULL pointer (0) returned on EOF or error.

**FILES**

`/etc/hosts`

**SEE ALSO**

`hosts`(4N).

**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 user ID is shared by several login names.

If getlogin is called within a process that is not attached to a terminal, it returns a NULL pointer. The correct procedure for determining the login name is to call cuserid or getlogin. If getlogin fails, call getpwuid.

RETURN VALUE

getlogin returns the NULL pointer if *name* is not found.

FILES

/etc/utmp

SEE ALSO

cuserid(3S), getgrent(3C), getpwent(3C), utmp(4).

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(3) routines for accessing the file system description file /etc/fstab, and the mounted file system description file /etc/mtab.

setmntent opens a file system description file and returns a file pointer for use 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 broken-out fields of a line in the file system description file, <mntent.h>. The fields have meanings described in fstab(4).

```
struct mntent {
    char    *mnt_fsname; /* file system name */
    char    *mnt_dir;    /* file system path prefix */
    char    *mnt_type;   /* 4.2, 5.2, nfs, swap, or ignore */
    char    *mnt_opts;   /* ro, rw, quota, noquota, hard, soft */
    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.

**RETURN VALUE**

NULL pointer (0) returned on EOF or error.

**FILES**

`/etc/fstab`
`/etc/mtab`

**SEE ALSO**

`fstab`(4), `mtab`(4).

**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)
long net;

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 */
        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 nonzero, the net data base will not be closed after each call to getnetent (either directly, or indirectly through one of the

other "getnet" calls).

endnetent closes the file.

getnetbyname and getnetbyaddr sequentially search from the beginning of the file until a matching net name or net address is found, or until EOF is encountered. Network numbers are supplied in host order.

RETURN VALUE

NULL pointer (0) returned on EOF or error.

FILES

/etc/networks

SEE ALSO

networks(4N).

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;

int setnetgrent(netgroup)
char *netgroup

int endnetgrent()

getnetgrent(machinep, userp, domainp)
char **machinep, **userp, **domainp;
```

## DESCRIPTION

inngetgr 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*. getnetgrent will malloc space for the name. This space is released when a endnetgrent call is made. getnetgrent returns 1 if it succeeding in obtaining another member of the network group, 0 if it has reached the end of the group.

setnetgrent establishes the network group from which getnetgrent will obtain members, and also restarts calls to getnetgrent from the beginning of the list. If the previous setnetgrent call was to a different network group, a endnetgrent call is implied. endnetgrent frees the space allocated during the getnetgrent calls.

## FILES

/etc/netgroup

## NAME

getopt – get option letter from argument vector

## SYNOPSIS

```
int getopt(argc, argv, optstring)
int argc;
char **argv, *optstring;

extern char *optarg;
extern int optind, opterr;
```

## 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*. This error message may be disabled by setting opterr to 0.

## EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options a and b, and the options f and o, both of which require arguments:

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

```
...
while ((c = getopt (argc, argv, "abf:o:")) != EOF)
     switch (c) {
     case 'a':
          if (bflg)
               errflg++;
          else
               aflg++;
          break;
     case 'b':
          if (aflg)
               errflg++;
          else
               bproc( );
          break;
     case 'f':
          ifile = optarg;
          break;
     case 'o':
          ofile = optarg;
          break;
     case '?':
          errflg++;
     }
if (errflg) {
     fprintf (stderr, "usage: ..fl. ");
     exit (2);
}
for ( ; optind < argc; optind++) {
     if (access (argv[optind], 4)) {
...
```

**SEE ALSO**

getopt(1).

## NAME

getpass – read a password

## SYNOPSIS

```
char *getpass (prompt)
char *prompt;
```

## DESCRIPTION

getpass reads up to a newline or EOF from the file /dev/tty, after prompting on the standard error output with the null-terminated string *prompt* and disabling echo. A pointer is returned to a null-terminated string of at most 8 characters. If /dev/tty cannot be opened, a NULL pointer is returned. An interrupt terminates input and sends an interrupt signal to the calling program before returning.

## FILES

/dev/tty

## SEE ALSO

crypt(3C).

## WARNING

The above routine uses <stdio.h>. This causes the size of programs not otherwise using standard I/O to increase more than might be expected.

## BUGS

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

## NAME

getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent – get protocol entry

## SYNOPSIS

```
#include <netdb.h>

struct protoent *getprotoent()

struct protoent *getprotobyname(name)
char *name;

struct protoent *getprotobynumber(proto)
int proto;

int setprotoent(stayopen)
int stayopen

int endprotoent()
```

## DESCRIPTION

getprotoent, getprotobyname, and getprotobynumber each return a pointer to an object with the following structure containing the broken-out fields of a line in the network protocol data base, /etc/protocols.

```
struct protoent {
        char *p_name;       /* official name of protocol */
        char **p_aliases;   /* alias list */
        long p_proto;       /* protocol number */
};
```

The members of this structure are:

p_name — The official name of the protocol.

p_aliases — A zero terminated list of alternate names for the protocol.

p_proto — The protocol number.

getprotoent reads the next line of the file, opening the file if necessary.

setprotoent opens and rewinds the file. If the *stayopen* flag is nonzero, the net data base will not be closed after each call to getprotoent (either directly, or indirectly through one of the other "getproto" calls).

endprotoent closes the file.

getprotobyname and getprotobynumber sequentially search from the beginning of the file until a matching protocol

name or protocol number is found, or until EOF is encountered.

RETURN VALUE

NULL pointer (0) returned on EOF or error.

FILES

/etc/protocols

SEE ALSO

protocols(4N).

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet protocols are currently understood.

## NAME

getptabent, addptabent, endptabent, setptabent, numbptabent – get partition table file entry

## SYNOPSIS

```
#include <stdio.h>
#include <apple/ptabent.h>

struct ptabent *getptabent(filep)
FILE *filep;

int addptabent(filep, ptab)
FILE *filep;
struct ptabent *ptab;

int endptabent(filep)
FILE *filep;

FILE *setptabent(fname, type)
char *fname;
char *type;

int numptabent(filep)
FILE *filep;

cc [flags] files -lptab [libraries]
```

## DESCRIPTION

`setptabent` opens a partition table file and returns a file pointer which can then be used with `getptabent` or `addptabent`. The *type* argument is the same as in `fopen`(3). `getptabent` returns a pointer to an object with the following structure containing the broken-out fields of a line in the partition table file. The fields have meanings described in `ptab`(4).

```
struct ptabent {
        char *ptab_name;     /* partition name */
        char *ptab_type;     /* partition type */
        int  ptab_ctrl;      /* controller number */
        int  ptab_disk;      /* disk number */
        int  ptab_part;      /* partition number */
};
```

`addptabent` adds the `ptabent` structure `ptab` to the end of the open file *filep*. `numptabent` returns the number of partition table file entries and has the effect of rewinding the partition table file to allow repeated searches. `endptabtent` closes the file.

**FILES**

/etc/ptab

**RETURN VALUE**

A NULL pointer (0) is returned on EOF or error by setptabent and getptabent. addptabent, endptabent, and numbptabent return EOF on error.

**BUGS**

The returned ptabent structure points to static information that is overwritten in each call.

**SEE ALSO**

pname(1M), ptab(4).

## NAME

getpw – get name from UID

## SYNOPSIS

```
int  getpw(uid,  buf)
int  uid;
char  *buf;
```

## DESCRIPTION

getpw searches the password file for a user ID number that equals *uid*, copies the line of the password file in which *uid* was found into the array pointed to by *buf*, and returns 0. The line is null terminated. getpw returns nonzero if *uid* cannot be found.

This routine is included only for compatibility with prior systems and should not be used; see getpwent(3C) for routines to use instead.

## RETURN VALUE

getpw returns nonzero on error.

## FILES

/etc/passwd

## SEE ALSO

getpwent(3C), passwd(4).

## WARNING

The above routine uses <stdio.h>. Therefore, the size of programs not otherwise using standard I/O is increased more than might be expected.

## NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent – get password file entry

## SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent()

struct passwd *getpwuid(uid)
int uid;

struct passwd *getpwnam(name)
char *name;

void setpwent()

void endpwent()

struct passwd *fgetpwent(f)
FILE *f;
```

## 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 /etc/passwd file. Each line in the file contains a passwd structure, declared in the <pwd.h> header file:

```
struct passwd {
        char    *pw_name;
        char    *pw_passwd;
        int     pw_uid;
        int     pw_gid;
        char    *pw_age;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};
```

Because this structure is declared in <pwd.h>, it is not necessary to redeclare it.

The pw_comment field is unused; the others have meanings described in passwd(4).

When first called, getpwent returns a pointer to the first passwd structure in the file; thereafter, it returns a pointer to the next passwd structure in the file; therefore, successive calls can

be used to search the entire file. `getpwuid` searches from the beginning of the file until a numerical user id matching *uid* is found; it returns a pointer to the particular structure in which the match was found. `getpwnam` searches from the beginning of the file until a login name matching *name* is found; it returns a pointer to the particular structure in which the match was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to `setpwent` has the effect of rewinding the password file to allow repeated searches. `endpwent` may be called to close the password file when processing is complete.

`fgetpwent` returns a pointer to the next `passwd` structure in the stream *f*, which matches the format of `/etc/passwd`.

**RETURN VALUE**

A NULL pointer is returned on EOF or error.

**FILES**

`/etc/passwd`

**SEE ALSO**

`cuserid`(3S), `getlogin`(3C), `getgrent`(3C), `putpwent`(3C), `passwd`(4).

**WARNING**

The above routines use `<stdio.h>`. Therefore the size of programs not otherwise using standard I/O is increased more than might be expected.

**BUGS**

All information is contained in a static area, so it must be copied if it is to be saved.

## 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;
int n;
FILE *stream;
```

## DESCRIPTION

gets reads characters from the standard input stream, stdin, into the array pointed to by *s*, until a newline character is read or an end-of-file condition is encountered. The newline character is discarded and the string is terminated with a null character.

fgets reads characters from the *stream* into the array pointed to by *s* until *n*−1 characters are read, or a *newline* character is read and transferred to *s*, or an end-of-file condition is encountered. The string is then terminated with a null character.

## RETURN VALUE

If end-of-file is encountered and no characters have been read, no characters are transferred to *s* and a NULL pointer is returned. If a read error (e.g., trying to use these functions on a file that has not been opened for reading) occurs, a NULL pointer is returned. Otherwise *s* is returned.

## SEE ALSO

ferror(3S), fopen(3S), fread(3S), getc(3S), scanf(3S).

## NOTE

gets deletes the newline ending its input, but fgets keeps it.

## 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;

int setservent(stayopen)
int stayopen

int endservent()
```

## DESCRIPTION

`getservent`, `getservbyname`, and `getservbyport` each return a pointer to an object with the following structure containing the broken-out fields of a line in the network services data base, `/etc/services`.

```
struct servent {
    char *s_name;     /* official name of service */
    char **s_aliases; /* alias list */
    long 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

`getservent` (either directly, or indirectly through one of the other "getserv" calls).

`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.

**RETURN VALUE**

NULL pointer (0) returned on EOF or error.

**FILES**

`/etc/services`

**SEE ALSO**

`getprotoent`(3N), `services`(4N).

**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

getutent, getutid, getutline, pututline, setutent, endutent, utmpname – access utmp file entry

## SYNOPSIS

```
#include <sys/types.h>
#include <utmp.h>

struct utmp *getutent()

struct utmp *getutid(id)
struct utmp *id;

struct utmp *getutline(line)
struct utmp *line;

void pututline(utmp)
struct utmp *utmp;

void setutent()

void endutent()

ivoid utmpname(file)
char *file;
```

## DESCRIPTION

getutent, getutid, and getutline each return a pointer to a structure of the following type:

```
struct utmp {
     char   ut_user[8];     /* User login name */
     char   ut_id[4];       /* /etc/inittab ID
                               (usually line#) */
     char   ut_line[12];    /* device name (console, lnxx) */
     short  ut_pid;         /* process ID */
     short  ut_type;        /* type of entry */
     struct exit_status {
     short  e_termination; /* Process termination status */
     short  e_exit;         /* Process exit status */
     } ut_exit;             /* Exit status of a process
                            /* marked as DEAD_PROCESS */
     time_t  ut_time;       /* time entry was made */
     char    ut_host[16];   /* host name, if remote */
};
```

getutent reads in the next entry from a utmp-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.

`getutid` searches forward from the current point in the `utmp` file until it finds an entry with a `ut_type` matching `id->ut_type` if the type specified is `RUN_LVL`, `BOOT_TIME`, `OLD_TIME`, or `NEW_TIME`. If the type specified in *id* is `INIT_PROCESS`, `LOGIN_PROCESS`, `USER_PROCESS`, or `DEAD_PROCESS`, `getutid` will return a pointer to the first entry whose type is one of these four and whose `ut_id` field matches `id->ut_id`. `getutid` fails if the end of file is reached without a match.

`getutline` searches forward from the current point in the `utmp` file until it finds an entry of the type `LOGIN_PROCESS` or `USER_PROCESS` which also has a `ut_line` string matching the `line->ut_line` string. If the end of file is reached without a match, it fails.

`pututline` writes out the supplied `utmp` structure into the `utmp` file. It uses `getutid` to search forward for the proper place if it finds that it is not already at the proper place. It is assumed that the user of `pututline` has searched for the proper entry using one of the `getut` routines. If this has been done, `pututline` will not search. If `pututline` does not find a matching slot for the new entry, it will add a new entry to the end of the file.

`setutent` resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.

`endutent` closes the currently open file.

`utmpname` allows the user to change the name of the file examined from `/etc/utmp` to any other filename. It is expected that most often this other file will be `/etc/wtmp`. If the file doesn't exist, this will not be apparent until the first attempt to reference the file is made. `utmpname` does not open the file. It just closes the old file, if it is currently open, and saves the new filename.

**RETURN VALUE**

A NULL pointer is returned upon failure to read or write. Failure to read may be due to permissions or because end-of-file has been reached.

**FILES**

`/etc/utmp`
`/etc/wtmp`

**SEE ALSO**

ttyslot(3C), utmp(4).

**COMMENTS**

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. Each call to either getutid or getutline sees the routine examine the static structure before performing more I/O. If the search of the static structure results in a match, no further search is performed. To use getutline to search for multiple occurences, zero out the static structure after each success; otherwise getutline will just return the same pointer over and over again. There is one exception to the rule about removing the structure before further reads are done. If the implicit read done by pututline finds that it isn't already at the correct place in the file, the contents of the static structure returned by the getutent, getutid, or getutline routines are not harmed, if the user has just modified those contents and passed the pointer back to pututline.

These routines use buffered standard I/O for input, but pututline uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the utmp and wtmp files.

## 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.

Maximum pathname length is PATH_MAX characters (see intro(2)).

## DIAGNOSTICS

getwd returns zero and places a message in *pathname* if an error occurs.

## NAME

hsearch, hcreate, hdestroy – manage hash search tables

## SYNOPSIS

```
#include <search.h>

ENTRY *hsearch(item, action)
ENTRY item;
ACTION action;

int hcreate(nel)
unsigned nel;

void hdestroy()
```

## DESCRIPTION

hsearch is a hash-table search routine generalized from Knuth (6.4) Algorithm D. It returns a pointer into a hash table indicating the location at which an entry can be found. *item* is a structure of type ENTRY (defined in the <search.h> header file) containing two pointers: *item.key* points to the comparison key, and *item.data* points to any other data to be associated with that key. (Pointers to types other than character should be cast to pointer-to-character.) *action* is a member of an enumeration type ACTION indicating the disposition of the entry if it cannot be found in the table. ENTER indicates that the item should be inserted in the table at an appropriate point. FIND indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a NULL pointer.

hcreate allocates sufficient space for the table, and must be called before hsearch is used. *nel* is an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

hdestroy destroys the search table, and may be followed by another call to hcreate.

## NOTES

hsearch uses "open addressing" with a "multiplicative" hash function. However, its source code has many other options available which the user may select by compiling the hsearch source with the following symbols defined to the preprocessor:

DIV — Use the *remainder modulo table size* as the hash function instead of the multiplicative algorithm.

USCR Use a User Supplied Comparison Routine for ascertaining table membership. The routine should be named `hcompar` and should behave in a mannner similar to `strcmp` (see `string`(3C)).

CHAINED Use a linked list to resolve collisions. If this option is selected, the following other options become available.

START Place new entries at the beginning of the linked list (default is at the end).

SORTUP Keep the linked list sorted by key in ascending order.

SORTDOWN Keep the linked list sorted by key in descending order.

Additionally, there are preprocessor flags for obtaining debugging printout (-DDEBUG) and for including a test driver in the calling routine (-DDRIVER). The source code should be consulted for further details.

**RETURN VALUE**

`hsearch` returns a NULL pointer if either the action is `FIND` and the item could not be found or the action is `ENTER` and the table is full.

`hcreate` returns zero if it cannot allocate sufficient space for the table.

**EXAMPLE**

The following example will read in strings followed by two numbers and store them in a hash table, discarding duplicates. It will then read in strings and find the matching entry in the hash table and print it out.

```
#include <stdio.h>
#include <search.h>

struct info {       /* this is the info stored in
   int age, room;       the table other than the key */
};
#define NUM_EMPL 5000 /* # of elements in search
                                    table */

main( )
{
      /* space to store strings */
```

```
    char string_space[NUM_EMPL*20];

    /* space to store employee info */
    struct info info_space[NUM_EMPL];

    /* next avail space in string_space */
    char *str_ptr = string_space;

    /* next avail space in info_space */
    struct info *info_ptr = info_space;
    ENTRY item, *found_item, *hsearch( );

    /* name to look for in table */
    char name_to_find[30];
    int i = 0;

    /* create table */
    (void) hcreate(NUM_EMPL);
    while (scanf("%s%d%d", str_ptr, &info_ptr->age,
        &info_ptr->room) != EOF && i++ < NUM_EMPL) {

       /* put info in structure,
            and structure in item */
       item.key = str_ptr;
       item.data = (char *)info_ptr;
       str_ptr += strlen(str_ptr) + 1;
       info_ptr++;

       /* put item into table */
       (void) hsearch(item, ENTER);
}

/* access table */
item.key = name_to_find;
while (scanf("%s", item.key) != EOF) {
    if ((found_item = hsearch(item, FIND)) != NULL) {

       /* if item is in the table */
       (void)printf("found %s, age = %d, room = %d\n",
            found_item->key,
           ((struct info *)found_item->data)->age,
           ((struct info *)found_item->data)->room);
     } else {
```

```
            (void)printf("no such employee %s\n",
                  name_to_find)
            }
        }
    }
```

**SEE ALSO**

bsearch(3C), lsearch(3C), malloc(3C), malloc(3X), string(3C), tsearch(3C).

**WARNING**

hsearch and hcreate use malloc(3C) to allocate space.

**BUGS**

Only one hash search table may be active at any given time.

**NAME**

hypot – Euclidean distance function

**SYNOPSIS**

```
#include <math.h>

double hypot(x, y)
double x, y;
```

**DESCRIPTION**

hypot returns the following, taking precautions against unwarranted overflows:

```
sqrt(x * x + y * y)
```

**RETURN VALUE**

When the correct value would overflow, hypot returns HUGE and sets errno to ERANGE.

These error-handling procedures may be changed with the function matherr(3M).

**SEE ALSO**

matherr(3M).

**NAME**

iargc – return command line arguments

**SYNOPSIS**

```
integer i
i=iargc()
```

**DESCRIPTION**

The iargc function returns the number of command line arguments passed to the program. Thus, if a program were invoked via

foo *arg1 arg2 arg3*

iargc() would return "3".

**SEE ALSO**

getarg(3F).

**NAME**

index – return location of Fortran substring

**SYNOPSIS**

```
character *N1 ch1
character *N2 ch2
integer i

i=index(ch1, ch2)
```

**DESCRIPTION**

index returns the location of substring *ch2* in string *ch1*. The value returned is either the position at which substring *ch2* starts or 0 if *ch2* is not present in string *ch1*.

## 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>

struct in_addr inet_addr(cp)
char *cp;

int inet_network(cp)
char *cp;

char *inet_ntoa(in)
struct inet_addr in;

struct in_addr inet_makeaddr(net, lna)
int net, lna;

int inet_lnaof(in)
struct in_addr in;

int inet_netof(in)
struct in_addr in;
```

## DESCRIPTION

The routines `inet_addr` and `inet_network` each interpret character strings representing numbers expressed in the Internet standard "." notation, returning numbers suitable for use as Internet addresses and Internet network numbers, respectively. The routine `inet_ntoa` takes an Internet address and returns an ASCII string representing the address in "." notation. The routine `inet_makeaddr` takes an Internet network number and a local network address and constructs an Internet address from it. The routines `inet_netof` and `inet_lnaof` break apart Internet host addresses, returning the network number and local network address part, respectively.

All Internet address are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

## INTERNET ADDRESSES

Values specified using the "." notation take one of the following forms:

a.b.c.d

a.b.c
a.b
a

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.

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).

## RETURN VALUE

The value –1 is returned by `inet_addr` and `inet_network` for malformed requests.

## SEE ALSO

`gethostent`(3N), `getnetent`(3N), `hosts`(4N), `networks`(4N)

## 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.

## 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.

## RETURN VALUE

initgroups returns –1 if it was not invoked by the superuser.

## FILES

/etc/group
/etc/passwd

## SEE ALSO

setgroups(2).

## 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[];
};

int insque(elem, pred)
struct qelem *elem, *pred;

int remque(elem)
struct qelem *elem;
```

## DESCRIPTION

The `insque` and `remque` macros 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.

## FILES

`/usr/include/?/insque.h`

## NAME

killpg – send signal to a process group

## SYNOPSIS

```
int killpg(pgrp, sig)
int pgrp, sig;
```

## DESCRIPTION

killpg sends the signal *sig* to the process group *pgrp*.

The sending process and members of the process group must have the same effective user ID, otherwise this call is restricted to the superuser. As a single special case the continue signal SIGCONT may be sent to any process which 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 corresponding to that specified by *pgrp*. |
| [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), getpid(2).

## NAME

l3tol, ltol3 – convert between 3-byte integers and long integers

## SYNOPSIS

```
void  l3tol(lp, cp, n)
long  *lp;
char  *cp;
int    n;

void  ltol3(cp, lp, n)
char  *cp;
long  *lp;
int  n;
```

## DESCRIPTION

l3tol converts a list of *n* 3-byte integers (packed into a character string pointed to by *cp*) into a list of long integers pointed to by lp.

ltol3 performs the reverse conversion from long integers (*lp*) to 3-byte integers (*cp*).

These functions are useful for file system maintenance where the block numbers are 3 bytes long.

## SEE ALSO

fs(4).

## BUGS

Because of possible differences in byte ordering, the numerical values of the long integers are machine-dependent.

## NAME

ldahread – read the archive header of a member of an archive file

## SYNOPSIS

```
#include <stdio.h>
#include <ar.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldahread(ldptr, arhead
LDFILE *ldptr;
ARCHDR *arhead;
```

## DESCRIPTION

If TYPE(*ldptr*) is the archive file magic number, ldahread reads the archive header of the common object file currently associated with *ldptr* into the area of memory beginning at *arhead*.

Programs using this routine should be loaded with the object file access library libld.a.

## RETURN VALUE

ldahread returns SUCCESS or FAILURE. ldahread fails if TYPE(*ldptr*) does not represent an archive file or if it cannot read the archive header.

## SEE ALSO

ldclose(3X), ldopen(3X), ldfcn(3X), ar(4).

**NAME**

ldclose, ldaclose – close a common object file

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldclose(ldptr)
LDFILE *ldptr;

int ldaclose(ldptr)
LDFILE *ldptr;
```

**DESCRIPTION**

ldopen(3X) and ldclose are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of common object files can be processed as if it were a series of simple common object files.

If TYPE(*ldptr*) does not represent an archive file, ldclose closes the file and frees the memory allocated to the LDFILE structure associated with *ldptr*. If TYPE(*ldptr*) is the magic number of an archive file, and if there are any more files in the archive, ldclose reinitializes OFFSET(*ldptr*) to the file address of the next archive member and returns FAILURE. The LDFILE structure is prepared for a subsequent ldopen(3X). In all other cases, ldclose returns SUCCESS.

ldaclose closes the file and frees the memory allocated to the LDFILE structure associated with *ldptr* regardless of the value of TYPE(*ldptr*). ldaclose always returns SUCCESS. The function is often used in conjunction with ldaopen.

Programs using this routine must be loaded with the object file access library libld.a.

**SEE ALSO**

fclose(3S), ldfcn(3X), ldopen(3X).

## NAME

ldfcn – common object file access routines

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

## DESCRIPTION

The common object file access routines are a collection of functions for reading an object file that is in common object file form. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines effectively insulate the calling program from knowledge of the overall structure of the object file.

The interface between the calling program and the object file access routines is based on the defined type LDFILE (defined as struct ldfile), which is declared in the header file <ldfcn.h>. The primary purpose of this structure is to provide uniform access to both simple object files and object files that are members of an archive file.

The function ldopen(3X) allocates and initializes the LDFILE structure and returns a pointer to the structure to the calling program. The fields of the LDFILE structure may be accessed individually through macros defined in <ldfcn.h> and contain the following information:

LDFILE **ldptr;*

TYPE(*ldptr*) The file magic number, used to distinguish between archive members and simple object files.

IOPTR(*ldptr*) The file pointer returned by fopen(3S) and used by the standard input/output functions.

OFFSET(*ldptr*) The file address of the beginning of the object file; the offset is nonzero if the object file is a member of an archive file.

HEADER(*ldptr*) The file header structure of the object file.

The object file access functions may be divided into four categories:

(1) functions that open or close an object file

`ldopen`(3X) and `ldaopen`
open a common object file

`ldclose`(3X) and `ldaclose`
close a common object file

(2) functions that read header or symbol table information

`ldahread`(3X) read the archive header of a member of an archive file

`ldfhread`(3X) read the file header of a common object file

`ldshread`(3X) and `ldnshread`
read a section header of a common object file

`ldtbread`(3X) read a symbol table entry of a common object file

`ldgetname`(3X) retrieve a symbol name from a symbol table entry or from the string table

(3) functions that position an object file at (seek to) the start of the section, relocation, or line number information for a particular section.

`ldohseek`(3X) seek to the optional file header of a common object file

`ldsseek`(3X) and `ldnsseek`
seek to a section of a common object file

`ldrseek`(3X) and `ldnrseek`
seek to the relocation information for a section of a common object file

`ldlseek`(3X) and `ldnlseek`
seek to the line number information for a section of a common object file

`ldtbseek`(3X) seek to the symbol table of a common object file

(4) the function `ldtbindex`(3X) which returns the index of a particular common object file symbol table entry

These functions are described in detail in the manual pages identified for each function.

All the functions except ldopen, ldgetname(3X), ldaopen, and ldtbindex return either SUCCESS or FAILURE, which are constants defined in <ldfcn.h>. ldopen and ldaopen both return pointers to a LDFILE structure.

Programs using this routine must be loaded with the object file access library libld.a.

## MACROS

Additional access to an object file is provided through a set of macros defined in <ldfcn.h>. These macros parallel the standard input/output file reading and manipulating functions, translating a reference of the LDFILE structure into a reference to its file descriptor field.

The following macros are provided:

GETC(*ldptr*)
FGETC(*ldptr*)
GETW(*ldptr*)
UNGETC(*c, ldptr*)
FGETS(*s, n, ldptr*)
FREAD(*ptr, size, nitems, ldptr*)
FSEEK(*ldptr, offset, ptrname*)
FTELL(*ldptr*)
REWIND(*ldptr*)
FEOF(*ldptr*)
FERROR(*ldptr*)
FILENO(*ldptr*)
SETBUF(*ldptr, buf*)
STROFFSET(*ldptr*)

The STROFFSET macro calculates the address of the string table in an object file. See the manual entries for the corresponding standard input/output library functions for details on the use of these macros. (The functions are identified as 3S in this manual.)

## WARNINGS

The macro FSEEK defined in the header file <ldfcn.h> translates into a call to the standard input/output function fseek(3S). FSEEK should not be used to seek from the end of an archive file since the end of an archive file may not be the same as the end of one of its object file members.

## SEE ALSO

fopen(3S), fseek(3S), ldahread(3X), ldclose(3X), ldfhread(3X), ldgetname(3X), ldlread(3X),

ldlseek(3X), ldohseek(3X), ldopen(3X), ldrseek(3X), ldlseek(3X), ldshread(3X), ldtbindex(3X), ldtbread(3X), ldtbseek(3X).
"COFF Reference" and "C Object Library" *A/UX Programming Languages and Tools, Volume 1.*

## NAME

ldfhread – read the file header of a common object file

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldfhread(ldptr, filehead)
LDFILE *ldptr;
FILHDR *filehead;
```

## DESCRIPTION

ldfhread reads the file header of the common object file currently associated with *ldptr* into the area of memory beginning at *filehead*.

ldfhread returns SUCCESS or FAILURE. ldfhread fails if it cannot read the file header.

In most cases the use of ldfhread can be avoided by using the macro HEADER(*ldptr*) defined in <ldfcn.h> (see ldfcn(3)). The information in any field of the file header may be accessed by applying the dot operator to the value returned by the HEADER macro; for example: HEADER(*ldptr*).f_timdat. .

The program using this routine must be loaded with the object file access library libld.a.

## SEE ALSO

ldclose(3X), ldfcn(3X), ldopen(3X), filehdr(4).

## NAME

ldgetname – retrieve symbol name for object file symbol table entry

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

char *ldgetname(ldptr, symbol)
LDFILE *ldptr;
SYMENT *symbol;
```

## DESCRIPTION

ldgetname returns a pointer to the name associated with *symbol* as a string. The string is contained in a static buffer local to ldgetname. Because the buffer is overwritten by each call to ldgetname, it must be copied by the caller if the name is to be saved.

The common object file format has been extended to handle arbitrary length symbol names with the addition of a "string table." ldgetname returns the symbol name associated with a symbol table entry for either an object file or a preobject file. Thus, ldgetname can be used to retrieve names from object files without any backward compatibility problems.

Typically, ldgetname is called immediately after a successful call to ldtbread to retrieve the name associated with the symbol table entry filled by ldtbread.

Programs using this routine should be loaded with the object file access library libld.a.

## ERRORS

ldgetname returns NULL (defined in <stdio.h>) for an object file if the name cannot be retrieved. This occurs when:

the string table cannot be found.

not enough memory can be allocated for the string table.

the string table appears not to be a string table (e.g., if an auxiliary entry is handed to ldgetname that looks like a reference to a name in a nonexistent string table).

the name's offset into the string table is beyond the end of the string table.

**SEE ALSO**

ldclose(3X), ldfcn(3X), ldopen(3X), ldtbseek(3X), ldtbread(3X).

## NAME

ldlread, ldlinit, ldlitem – manipulate line number entries of a common object file function

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <linenum.h>
#include <ldfcn.h>

int ldlread(ldptr, fcnindx, linenum, linent)
LDFILE *ldptr;
long fcnindx;
unsigned short linenum;
LINENO linent;

int ldlinit(ldptr, fcnindx)
LDFILE *ldptr;
long fcnindx;

int ldlitem(ldptr, linenum, linent)
LDFILE *ldptr;
unsigned short linenum;
LINENO linent;
```

## DESCRIPTION

ldlread searches the line number entries of the common object file currently associated with *ldptr*. ldlread begins its search with the line number entry for the beginning of a function and confines its search to the line numbers associated with a single function. The function is identified by *fcnindx*, the index of its entry in the object file symbol table. ldlread reads the entry with the smallest line number equal to or greater than *linenum* into *linent*.

ldlinit and ldlitem together perform exactly the same function as ldlread. After an initial call to ldlread or ldlinit, ldlitem may be used to retrieve a series of line number entries associated with a single function. ldlinit simply locates the line number entries for the function identified by *fcnindx*. ldlitem finds and reads the entry with the smallest line number equal to or greater than *linenum* into *linent*.

Programs using this routine should be loaded with the object file access library libld.a.

## ERRORS

ldlread, ldlinit, and ldlitem each return either

SUCCESS or FAILURE. ldlread fails if there are no line number entries in the object file, if *fcnindx* does not index a function entry in the symbol table, or if it finds no line number equal to or greater than *linenum*.

ldlinit fails if there are no line number entries in the object file or if *fcnindx* does not index a function entry in the symbol table. ldlitem fails if it finds no line number equal to or greater than *linenum*.

**SEE ALSO**

ldclose(3X), ldfcn(3X), ldopen(3X), ldtbindex(3X).

## NAME

ldlseek, ldnlseek – seek to line number entries of a section of a common object file

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldlseek(ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnlseek(ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

## DESCRIPTION

ldlseek seeks to the line number entries of the section specified by *sectindx* of the common object file currently associated with *ldptr*.

ldnlseek seeks to the line number entries of the section specified by *sectname*.

ldlseek and ldnlseek return SUCCESS or FAILURE. ldlseek fails if *sectindx* is greater than the number of sections in the object file; ldnlseek fails if there is no section name corresponding to **sectname*. Either function fails if the specified section has no line number entries or if it cannot seek to the specified line number entries.

Note that the first section has an index of one.

Programs using this routine must be loaded with the object file access library libld.a.

## SEE ALSO

ldclose(3X), ldfcn(3X), ldopen(3X), ldshread(3X).

## NAME

ldohseek – seek to the optional file header of a common object file

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldohseek(ldptr)
LDFILE *ldptr;
```

## DESCRIPTION

ldohseek seeks to the optional file header of the common object file currently associated with *ldptr*.

ldohseek returns SUCCESS or FAILURE. ldohseek fails if the object file has no optional header or if it cannot seek to the optional header.

Programs using this routine should be loaded with the object file access routine library libld.a.

## SEE ALSO

ldclose(3X), ldfcn(3X), ldopen(3X), ldfhread(3X).

## NAME

ldopen, ldaopen – open a common object file for reading

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

LDFILE *ldopen(filename, ldptr)
char *filename;
LDFILE *ldptr;

LDFILE *ldaopen(filename, oldptr)
char *filename;
LDFILE *oldptr;
```

## DESCRIPTION

ldopen and ldclose(3X) are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus, an archive of common object files can be processed as if it were a series of simple common object files.

If ldptr has the value NULL, ldopen opens *filename*, allocates and initializes the LDFILE structure, and returns a pointer to the structure to the calling program.

If *ldptr* is valid and TYPE(*ldptr*) is the archive magic number, ldopen reinitializes the LDFILE structure for the next archive member of *filename*.

ldopen and ldclose are designed to work in concert. ldclose returns FAILURE only when TYPE(*ldptr*) is the archive magic number and there is another file in the archive to be processed. Only then should ldopen be called with the current value of *ldptr*. In all other cases, in particular whenever a new *filename* is opened, ldopen should be called with a NULL ldptr argument.

The following is a prototype for the use of ldopen and ldclose.

```
/* for each filename to be processed */

ldptr = NULL;
do
  if ((ldptr = ldopen(filename, ldptr)) != NULL )

  {
          /* check magic number */
          /* process the file */
  }
} while (ldclose(ldptr) == FAILURE );
```

If the value of *oldptr* is not NULL, ldaopen opens *filename* anew and allocates and initializes a new LDFILE structure, copying the TYPE, OFFSET, and HEADER fields from *oldptr*. ldaopen returns a pointer to the new LDFILE structure. This new pointer is independent of the old pointer, *oldptr*. The two pointers may be used concurrently to read separate parts of the object file. For example, one pointer may be used to step sequentially through the relocation information, while the other is used to read indexed symbol table entries.

Both ldopen and ldaopen open *filename* for reading. Both functions return NULL if *filename* cannot be opened or if memory for the LDFILE structure cannot be allocated. A successful open does not insure that the given file is a common object file or an archived object file.

Programs using this routine must be loaded with the object file access library libld.a.

**SEE ALSO**

fopen(3S), ldclose(3X), ldfcn(3X).

## NAME

ldrseek, ldnrseek – seek to relocation entries of a section of a common object file

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldrseek(ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnrseek(ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

## DESCRIPTION

ldrseek seeks to the relocation entries of the section specified by *sectindx* of the common object file currently associated with *ldptr*.

ldnrseek seeks to the relocation entries of the section specified by *sectname*.

The routines ldrseek and ldnrseek return SUCCESS or FAILURE. ldrseek fails if *sectindx* is greater than the number of sections in the object file; ldnrseek fails if there is no section name corresponding with *sectname*. Either function fails if the specified section has no relocation entries or if it cannot seek to the specified relocation entries.

Note that the first section has an index of one.

Programs using this routine should be loaded with the object file access library libld.a.

## SEE ALSO

ldclose(3X), ldfcn(3X), ldopen(3X), ldshread(3X).

## NAME

ldshread, ldnshread – read an indexed/named section header of a common object file

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <scnhdr.h>
#include <ldfcn.h>

int ldshread(ldptr, sectindx, secthead)
LDFILE *ldptr;
unsigned short sectindx;
SCNHDR *secthead;

int ldnshread(ldptr, sectname, secthead)
LDFILE *ldptr;
char *sectname;
SCNHDR *secthead;
```

## DESCRIPTION

ldshread reads the section header specified by *sectindx* of the common object file currently associated with *ldptr* into the area of memory beginning at *secthead*.

ldnshread reads the section header specified by *sectname* into the area of memory beginning at *secthead*.

ldshread and ldnshread return SUCCESS or FAILURE. ldshread fails if *sectindx* is greater than the number of sections in the object file; ldnshread fails if there is no section name corresponding with *sectname*. Either function fails if it cannot read the specified section header.

Note that the first section header has an index of one.

Programs using this routine must be loaded with the object file access library libld.a.

## SEE ALSO

ldclose(3X), ldfcn(3X), ldopen(3X).

## NAME

ldsseek, ldnsseek – seek to an indexed/named section of a common object file

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldsseek(ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnsseek(ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

## DESCRIPTION

ldsseek seeks to the section specified by *sectindx* of the common object file currently associated with *ldptr*.

ldnsseek seeks to the section specified by *sectname*.

ldsseek and ldnsseek return SUCCESS or FAILURE. ldsseek fails if *sectindx* is greater than the number of sections in the object file; ldnsseek fails if there is no section name corresponding with *sectname*. Either function fails if there is no section data for the specified section or if it cannot seek to the specified section.

Note that the first section has an index of one.

Programs using this routine should be loaded with the object file access library libld.a.

## SEE ALSO

ldclose(3X), ldfcn(3X), ldopen(3X), ldshread(3X).

## NAME

ldtbindex – compute the index of a symbol table entry of a common object file

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

long ldtbindex(ldptr)
LDFILE *ldptr;
```

## DESCRIPTION

ldtbindex returns the (long) index of the symbol table entry at the current position of the common object file associated with *ldptr*.

The index returned by ldtbindex may be used in subsequent calls to ldtbread(3X). However, since ldtbindex returns the index of the symbol table entry that begins at the current position of the object file, if ldtbindex is called immediately after a particular symbol table entry has been read, it returns the the index of the next entry.

ldtbindex fails if there are no symbols in the object file or if the object file is not positioned at the beginning of a symbol table entry.

Note that the first symbol in the symbol table has an index of zero.

Programs using this routine should be loaded with the object file access library libld.a.

## SEE ALSO

ldclose(3X), ldfcn(3X), ldopen(3X), ldtbread(3X), ldtbseek(3X).

## NAME

ldtbread – read an indexed symbol table entry of a common object file

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldtbread(ldptr, symindex, symbol)
LDFILE *ldptr;
long symindex;
SYMENT *symbol;
```

## DESCRIPTION

ldtbread reads the symbol table entry specified by *symindex* of the common object file currently associated with *ldptr* into the area of memory beginning at *symbol*.

ldtbread returns SUCCESS or FAILURE. ldtbread fails if *symindex* is greater than the number of symbols in the object file or if it cannot read the specified symbol table entry.

Note that the first symbol in the symbol table has an index of zero.

Programs using this routine must be loaded with the object file access library libld.a.

## SEE ALSO

ldclose(3X), ldfcn(3X), ldgetname(3X), ldopen(3X), ldtbseek(3X).

**NAME**

ldtbseek – seek to the symbol table of a common object file

**SYNOPSIS**

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldtbseek(ldptr)
LDFILE *ldptr;
```

**DESCRIPTION**

ldtbseek seeks to the symbol table of the common object file currently associated with *ldptr*.

ldtbseek returns SUCCESS or FAILURE. ldtbseek fails if the symbol table has been stripped from the object file or if it cannot seek to the symbol table.

Programs using this routine must be loaded with the object file access library libld.a.

**SEE ALSO**

ldclose(3X), ldfcn(3X), ldopen(3X), ldtbread(3X).

**NAME**

len – return length of Fortran string

**SYNOPSIS**

```
character *N ch
integer i
i=len(ch)
```

**DESCRIPTION**

len returns the length of string *ch*.

**NAME**

lge, lgt, lle, llt – string comparision intrinsic functions

**SYNOPSIS**

```
character *N a1, a2
logical l

l=lge(a1, a2)
l=lgt(a1, a2)
l=lle(a1, a2)
l=llt(a1, a2)
```

**DESCRIPTION**

These functions return TRUE if the inequality holds and FALSE otherwise.

**NAME**

line_push – routine used to push streams line disciplines

**SYNOPSIS**

```
line_push (fildes)
int fildes;
```

**DESCRIPTION**

line_push will push the streams line discipline "line" onto the stream referenced by the file descriptor *fildes*. If *fildes* does not reference a stream or it references a stream that already has a line discipline pushed onto it nothing will happen.

**SEE ALSO**

line_sane(1M), streams(7).

**NAME**

lockf – record locking on files

**SYNOPSIS**

```
#include <unistd.h>

int lockf(fildes, function, size)
long size;
int fildes, function;
```

**DESCRIPTION**

The lockf call will allow sections of a file to be locked (advisory write locks). (Mandatory locking is available via locking(2)). Locking calls from other processes which attempt to lock the locked file section will either return an error value or be put to sleep until the resource becomes unlocked. All the locks for a process are removed when the process terminates. (See fcntl(2) for more information about record locking.)

*fildes* is an open file descriptor. The file descriptor must have O_WRONLY or O_RDWR permission in order to establish lock with this function call.

*function* is a control value which specifies the action to be taken. The permissible values for *function* are defined in <unistd.h> as follows:

```
#define F_ULOCK 0   /* Unlock a previously
                       locked section */
#define F_LOCK  1   /* Lock a section for
                       exclusive use */
#define F_TLOCK 2   /* Test and lock a section
                       for exclusive use */
#define F_TEST  3   /* Test section for other
                       processes locks */
```

All other values of *function* are reserved for future extensions and will result in an error return if not implemented.

F_TEST is used to detect if a lock by another process is present on the specified section. F_LOCK and F_TLOCK both lock a section of a file if the section is available. F_ULOCK removes locks from a section of the file.

*size* is the number of contiguous bytes to be locked or unlocked. The resource to be locked starts at the current offset in the file and extends forward for a positive size and backward for a negative

size. If *size* is zero, the section from the current offset through the largest file offset is locked (i.e., from the current offset through the present or any future end-of-file). An area need not be allocated to the file in order to be locked, as such locks may exist past the end-of-file.

The sections locked with `F_LOCK` or `F_TLOCK` may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent sections occur, the sections are combined into a single section. If the request requires that a new element be added to the table of active locks and this table is already full, an error is returned, and the new section is not locked.

`F_LOCK` and `F_TLOCK` requests differ only by the action taken if the resource is not available. `F_LOCK` will cause the calling process to sleep until the resource is available. `F_TLOCK` will cause the function to return a −1 and set `errno` to `[EACCES]` error if the section is already locked by another process.

`F_ULOCK` requests may, in whole or in part, release one or more locked sections controlled by the process. When sections are not fully released, the remaining sections are still locked by the process. Releasing the center section of a locked section requires an additional element in the table of active locks. If this table is full, an `[EDEADLK]` error is returned and the requested section is not released.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by accessing another process's locked resource. Thus calls to `lock` or `fcntl` scan for a deadlock prior to sleeping on a locked resource. An error return is made if sleeping on the locked resource would cause a deadlock.

Sleeping on a resource is interrupted with any signal. The `alarm`(2) command may be used to provide a timeout facility in applications which require this facility.

## 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 `lockf` utility will fail if one or more of the following are true:

`[EBADF]` *fildes* is not a valid open descriptor.

| | |
|---|---|
| [EACCES] | `function` is `F_TLOCK` or `F_TEST` and the section is already locked by another process. |
| [EDEADLK] | `function` is `F_LOCK` or `F_TLOCK` and a deadlock would occur. Also the `function` is either of the above or `F_ULOCK` and the number of entries in the lock table would exceed the number allocated on the system. |
| [EREMOTE] | *fildes* is a file descriptor referring to a file on a remotely mounted file system. |

**CAVEATS**

Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The standard I/O package is the most common source of unexpected buffering.

**SEE ALSO**

`close(2)`, `creat(2)`, `fcntl(2)`, `intro(2)`, `locking(2)`, `open(2)`, `read(2)`, `write(2)`.

## NAME

log, alog, dlog, clog – Fortran natural logarithm intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2
complex cx1, cx2

r2=alog(r1)
r2=log(r1)

dp2=dlog(dp1)
dp2=log(dp1)

cx2=clog(cx1)
cx2=log(cx1)
```

## DESCRIPTION

alog returns the real natural logarithm of its real argument. dlog returns the double-precision natural logarithm of its double-precision argument. clog returns the complex logarithm of its complex argument. The generic function log becomes a call to alog, dlog, or clog depending on the type of its argument.

## SEE ALSO

exp(3M).

## NAME

log10, alog10, dlog10 – Fortran common logarithm intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2

r2=alog10(r1)
r2=log10(r1)

dp2=dlog10(dp1)
dp2=log10(dp1)
```

## DESCRIPTION

alog10 returns the real common logarithm of its real argument. dlog10 returns the double-precision common logarithm of its double-precision argument. The generic function log10 becomes a call to alog10 or dlog10 depending on the type of its argument.

## SEE ALSO

exp(3M).

**NAME**

logname – return login name of user

**SYNOPSIS**

```
char *logname()
```

**DESCRIPTION**

logname returns a pointer to the null-terminated login name; it extracts the $LOGNAME variable from the user's environment.

This routine is kept in /lib/libPW.a.

**FILES**

/etc/profile

**SEE ALSO**

env(1), login(1), profile(4), environ(5).

**BUGS**

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

This method of determining a login name is subject to forgery.

## NAME

lsearch, lfind – linear search and update

## SYNOPSIS

```
#include <stdio.h>
#include <search.h>

char *lsearch(key, base, nelp, width, compar)
char *key;
char *base;
unsigned *nelp;
unsigned *width;
int (*compar)();

char *lfind(key, base, nelp, width, compar)
char *key;
char *base;
unsigned *nelp;
unsigned *width;
int (*compar)();
```

## DESCRIPTION

lsearch is a linear search routine generalized from Knuth (6.1) Algorithm S. It returns a pointer into a table indicating where a datum may be found. If the datum does not occur, it is added at the end of the table. *key* points to the datum to be sought in the table. *base* points to the first element in the table. *nelp* points to an integer containing the current number of elements in the table. The integer at **nelp* is incremented if the datum is added to the table. *width* is the width of an element in bytes. *compar* is the name of the comparison function which the user must supply (strcmp, for example). It is called with two arguments that point to the elements being compared. The function must return zero if the elements are equal and non-zero otherwise.

lfind is the same as lsearch except that if the datum is not found, it is not added to the table. Instead, a −1 pointer is returned.

## RETURN VALUE

If the searched for datum is found, both lsearch and lfind return a pointer to it. Otherwise, lfind returns NULL and lsearch returns a pointer to the newly added element.

## NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-

character.
The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.
Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

**EXAMPLE**

This fragment will read in ≤ TABSIZE strings of length ≤ ELSIZE and store them in a table, eliminating duplicates.

```
#include <stdio.h>
#include <search.h>

#define TABSIZE 50
#define ELSIZE 120

    char line[ELSIZE], tab[TABSIZE][ELSIZE], *lsearch( );
    unsigned nel = 0;
    int strcmp( );
    . . .
    while (fgets(line, ELSIZE, stdin) != NULL &&
          nel < TABSIZE)
             (void) lsearch(line, (char *)tab, &nel,
                       ELSIZE, strcmp);
    . . .
```

**SEE ALSO**

bsearch(3C), hsearch(3C), tsearch(3C).

**BUGS**

Undefined results can occur if there is not enough room in the table to add a new item.

## NAME

malloc, free, realloc, calloc, cfree – main memory allocator

## SYNOPSIS

```
char *malloc(size)
unsigned size;

void free(ptr)
char * ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

cfree(ptr , nelem, elsize)
char *ptr,
unsigned nelem, elsize;
```

## DESCRIPTION

malloc and free provide a simple general-purpose memory allocation package. malloc returns a pointer to a block of at least *size* bytes suitably aligned for any use.

The argument to free is a pointer to a block previously allocated by malloc; after free is performed this space is made available for further allocation, but its contents are left undisturbed.

Undefined results occur if the space assigned by malloc is overrun or if some random number is handed to free.

malloc allocates the first contiguous reach of free space of sufficient size found in a circular search from the last block allocated or freed; it coalesces adjacent free blocks as it searches. 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 are unchanged up to the lesser of the new and old sizes. If no free block of *size* bytes is available in the storage arena, realloc asks malloc to enlarge the arena by *size* bytes and then moves the data to the new space.

realloc also works if *ptr* points to a block freed since the last call of malloc, realloc, or calloc; thus sequences of free,

malloc, and realloc can exploit the search strategy of malloc to do storage compaction.

calloc allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

The arguments to cfree are the pointer to a block previously allocated by calloc plus the parameters to calloc.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

**RETURN VALUE**

malloc, realloc, and calloc return a NULL pointer if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. When this happens the block pointed to by *ptr* may be destroyed.

**NOTE**

Search time increases when many objects have been allocated; i.e., if a program allocates space but never frees it, each successive allocation takes longer.

**SEE ALSO**

brk(2), malloc(3X).

## NAME

malloc, free, realloc, calloc, mallopt, mallinfo – fast main memory allocator

## SYNOPSIS

```
#include <malloc.h>

char *malloc(size)
unsigned size;

void free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

int mallopt(cmd, value)
int cmd, value;

struct mallinfo mallinfo(max)
int max;
```

## DESCRIPTION

malloc and free provide a simple general-purpose memory allocation package, which runs considerably faster than the malloc(3C) package. It is found in the library "malloc", and is loaded if the option "-lmalloc" is used with cc(1) or ld(1).

malloc returns a pointer to a block of at least *size* bytes suitably aligned for any use.

The argument to free is a pointer to a block previously allocated by malloc; after free is performed this space is made available for further allocation, and its contents have been destroyed (but see mallopt below for a way to change this behavior).

Undefined results will occur if the space assigned by malloc is overrun or if some random number is handed to 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.

calloc allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

`mallopt` provides for control over the allocation algorithm. The available values for *cmd* are:

`M_MXFAST` Set `maxfast` to *value*. The algorithm allocates all blocks below the size of `maxfast` in large groups and then doles them out very quickly. The default value for `maxfast` is 0.

`M_NLBLKS` Set `numlblks` to *value*. The above mentioned "large groups" each contain `numlblks` blocks. `numlblks` must be greater than 0. The default value for `numlblks` is 100.

`M_GRAIN` Set `grain` to *value*. The sizes of all blocks smaller than `maxfast` are considered to be rounded up to the nearest multiple of `grain`. `grain` must be greater than 0. The default value of `grain` is the smallest number of bytes which will allow alignment of any data type. Value will be rounded up to a multiple of the default when `grain` is set.

`M_KEEP` Preserve data in a freed block until the next `malloc`, `realloc`, or `calloc`. This option is provided only for compatibility with the old version of `malloc` and is not recommended.

These values are defined in the <`malloc.h`> header file.

`mallopt` may be called repeatedly, but may not be called after the first small block is allocated.

`mallinfo` provides instrumentation describing space usage. It returns the structure:

```
struct mallinfo  {
    int arena;    /* total space in arena */
    int ordblks;   /* number of ordinary blocks */
    int smblks;    /* number of small blocks */
    int hblkhd;    /* space in holding block headers */
    int hblks;     /* number of holding blocks */
    int usmblks;   /* space in small blocks in use */
    int fsmblks;   /* space in free small blocks */
    int uordblks;  /* space in ordinary blocks in use */
    int fordblks;  /* space in free ordinary blocks */
    int keepcost;  /* space penalty if keep option */
                   /* is used */
}
```

This structure is defined in the <malloc.h> header file.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

**RTURN VALUE**

malloc, realloc and calloc return a NULL pointer if there is not enough available memory. When realloc returns NULL, the block pointed to by *ptr* is left intact. If mallopt is called after any allocation or if *cmd* or *value* are invalid, non-zero is returned. Otherwise, it returns zero.

**SEE ALSO**

brk(2), malloc(3C).

**WARNINGS**

This package usually uses more data space than malloc(3C).
The code size is also bigger than malloc(3C).
Note that unlike malloc(3C), this package does not preserve the contents of a block when it is freed, unless the M_KEEP option of mallopt is used.
Undocumented features of malloc(3C) have not been duplicated.

## NAME

matherr – error-handling function

## SYNOPSIS

```
#include <math.h>

int matherr(x)
struct exception *x;
```

## DESCRIPTION

matherr is invoked by functions in the Math Library when errors are detected. Users may define their own procedures for handling errors, by including a function named matherr in their programs. matherr must be of the form described above. When an error occurs, a pointer to the exception structure *x* will be passed to the user-supplied matherr function. This structure, which is defined in the <math.h> header file, is as follows:

```
struct exception {
        int type;
        char *name;
        double arg1, arg2, retval;
};
```

The element *type* is an integer describing the type of error that has occurred, from the following list of constants (defined in the header file):

| | |
|---|---|
| DOMAIN | argument domain error |
| SING | argument singularity |
| OVERFLOW | overflow range error |
| UNDERFLOW | underflow range error |
| TLOSS | total loss of significance |
| PLOSS | partial loss of significance |

The element *name* points to a string containing the name of the function that incurred the error. The variables *arg1* and *arg2* are the arguments with which the function was invoked. *retval* is set to the default value that will be returned by the function unless the user's matherr sets it to a different value.

If the user's matherr function returns nonzero, no error message will be printed, and errno will not be set.

If matherr is not supplied by the user, the default error-handling procedures, described with the math functions involved, will be invoked upon error. These procedures are also summarized in the table below. In every case, errno is set to EDOM or

ERANGE and the program continues.

**EXAMPLE**

```
#include <math.h>

int
matherr(x)
register struct exception *x;
{
   switch (x->type) {
   case DOMAIN:
       /* change sqrt to return sqrt(-arg1), not 0 */
       if (!strcmp(x->name, "sqrt")) {
           x->retval = sqrt(-x->arg1);
           return (0); /* print message and set errno */
       }
   case SING:
       /* all other domain or sing errors,
          print message and abort */
       fprintf(stderr, "domain error in %s\n", x->name);
       abort( );
   case PLOSS:
       /* print detailed error message */
       fprintf(stderr, "loss of significance in %s(%g) = %g\n",
           x->name, x->arg1, x->retval);
       return (1); /* take no other action */
   }
       return (0); /* all other errors,
                      execute default procedure */
}
```

**DEFAULT ERROR HANDLING PROCEDURES**

| | *Types of Errors* | | | | | |
|---|---|---|---|---|---|---|
| type | DOMAIN | SING | OVERFLOW | UNDERFLOW | TLOSS | PLOSS |
| errno | EDOM | EDOM | ERANGE | ERANGE | ERANGE | ERANGE |
| BESSEL: | – | – | – | – | M, 0 | * |
| y0, y1, yn (arg ≤ 0) | M, –H | – | – | – | – | – |
| EXP: | – | – | H | 0 | – | – |
| LOG, LOG10: | | | | | | |
| (arg < 0) | M, –H | – | – | – | – | – |
| (arg = 0) | – | M, –H | – | – | – | – |
| POW: | – | – | ±H | 0 | – | – |
| neg ** nonint | M, 0 | – | – | – | – | – |
| 0 ** nonpos | | | | | | |
| SQRT: | M, 0 | – | – | – | – | – |
| GAMMA: | – | M, H | H | – | – | – |
| HYPOT: | – | – | H | – | – | – |
| SINH: | – | – | ±H | – | – | – |
| COSH: | – | – | H | – | – | – |
| SIN, COS, TAN: | – | – | – | – | M, 0 | * |
| ASIN, ACOS, ATAN2: | M, 0 | – | – | – | – | – |

**ABBREVIATIONS**

| | |
|---|---|
| * | As much as possible of the value is returned. |
| M | Message is printed (EDOM error). |
| H | HUGE is returned. |
| –H | –HUGE is returned. |
| ±H | HUGE or –HUGE is returned. |
| 0 | 0 is returned. |

## NAME

max, max0, amax0, max1, amax1, dmax1 – Fortran maximum-value functions

## SYNOPSIS

```
integer i, j, k, l
real a, b, c, d
double precision dp1, dp2, dp3

l=max(i, j, k)
c=max(a, b)
d=max(a, b, c)
k=max0(i, j)
a=amax0(i, j, k)
i=max1(a, b)
d=amax1(a, b, c)
dp3=dmax1(dp1, dp2)
```

## DESCRIPTION

The maximum-value functions return the largest of their arguments; there may be any number of arguments. max is the generic form which can be used for all data types and takes its return type from that of its arguments. All arguments must be of the same type. max0 returns the integer form of the maximum value of its integer arguments; amax0, the real form of its integer arguments; max1, the integer form of its real arguments; amax1, the real form of its real arguments; and dmax1, the double-precision form of its double-precision arguments.

## SEE ALSO

min(3F).

## NAME

mclock – return Fortran time accounting

## SYNOPSIS

```
integer i
i=mclock()
```

## DESCRIPTION

mclock returns time accounting information about the current process and its child processes. The value returned is the sum of the current process's user time and the user and system times of all child processes.

## SEE ALSO

times(2), clock(3C), system(3F).

## NAME

memccpy, memchr, memcmp, memcpy, memset – memory operations

## SYNOPSIS

```
#include <memory.h>

char *memccpy(s1, s2, c, n)
char *s1, *s2;
int c, n;

char *memchr(s, c, n)
char *s;
int c, n;

int memcmp(s1, s2, n)
char *s1, *s2;
int n;

char *memcpy(s1, s2, n)
char *s1, *s2;
int n;

char *memset(s, c, n)
char *s;
int c, n;
```

## DESCRIPTION

These functions operate efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

memccpy copies characters from memory area *s2* into *s1*, stopping after the first occurrence of character *c* has been copied or after *n* characters have been copied, whichever comes first. It returns either a pointer to the character after the copy of *c* in *s1* or a NULL pointer if *c* was not found in the first *n* characters of *s2*.

memchr returns either a pointer to the first occurrence of character *c* in the first *n* characters of memory area *s* or a NULL pointer if *c* does not occur.

memcmp compares its arguments, looking at the first *n* characters only. It returns an integer less than, equal to, or greater than 0, depending on whether *s1* is lexicographically less than, equal to, or greater than *s2*.

memcpy copies *n* characters from memory area *s2* to *s1*. It returns *s1*.

memset sets the first *n* characters in memory area *s* to the value of character *c*. It returns *s*.

**NOTE**

For user convenience, all these functions are declared in the optional <memory.h> header file.

**BUGS**

memcmp uses native character comparison.

Because character movement is performed differently in different implementations, overlapping moves may yield unexpected results.

## NAME

min, min0, amin0, min1, amin1, dmin1 – Fortran minimum-value functions

## SYNOPSIS

```
integer i, j, k, l
real a, b, c, d
double precision dp1, dp2, dp3

l=min(i, j, k)
c=min(a, b)
d=min(a, b, c)
k=min0(i, j)
a=amin0(i, j, k)
i=min1(a, b)
d=amin1(a, b, c)
dp3=dmin1(dp1, dp2)
```

## DESCRIPTION

The minimum-value functions return the minimum of their arguments. There may be any number of arguments. `min` is the generic form which can be used for all data types. It takes its return type from that of its arguments, which must all be of the same type. `min0` returns the integer form of the minimum value of its integer arguments; `amin0`, the real form of its integer arguments; `min1`, the integer form of its real arguments; `amin1`, the real form of its real arguments; and `dmin1`, the double-precision form of its double-precision arguments.

## SEE ALSO

`max`(3F).

## NAME

mktemp – make a unique filename

## SYNOPSIS

```
char  *mktemp(template)
char  *template;
```

## DESCRIPTION

The function mktemp alters the contents of the string referenced by **template* so that it becomes a unique filename. The string at **template* should be initialized to a filenamed with six trailing X characters; mktemp replaces the Xs with a letter and the current process ID. The letter is selected so that the resulting name is not a duplicate an existing file.

## RETURN VALUE

mktemp returns the address of the unique (altered) filename. If a unique name cannot be created, template will point to a null (empty) string.

## SEE ALSO

getpid(2), tmpfile(3S), tmpnam(3S).

## BUGS

It is possible to run out of letters.

## NAME

mod, amod, dmod – Fortran remaindering intrinsic functions

## SYNOPSIS

```
integer i, j, k
real r1, r2, r3
double precision dp1, dp2, dp3

k=mod(i, j)

r3=amod(r1, r2)
r3=mod(r1, r2)

dp3=dmod(dp1, dp2)
dp3=mod(dp1, dp2)
```

## DESCRIPTION

mod returns the integer remainder of its first argument divided by its second argument. amod and dmod return, respectively, the real and double-precision whole number remainder of the integer division of their two arguments. The generic version mod returns the data type of its arguments.

## NAME

monitor – prepare execution profile

## SYNOPSIS

```
#include <mon.h>

void monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc) (), (*highpc) ();
WORD *buffer;
int bufsize, nfunc;
```

## DESCRIPTION

An executable program created by `cc -p` automatically includes calls for `monitor` with default parameters; `monitor` needn't be called explicitly except to gain fine control over profiling.

`monitor` is an interface to `profil`(2). *lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* elements of type `WORD` (defined in the `<mon.h>` header file). `monitor` arranges to record a histogram in the buffer. This histogram shows periodically sampled values of the program counter and counts of calls of certain functions. The lowest address sampled is that of *lowpc*; the highest address is just below *highpc*. *lowpc* may not equal 0 for this use of `monitor`. *nfunc* is the maximum number of call counts that can be kept; only calls of functions compiled with the profiling option `-p` of `cc`(1) are recorded. (The C Library and Math Library supplied when `cc -p` is used also have call counts recorded.) 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.

To profile the entire program, it is sufficient to use:

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

`etext` lies just above all the program text; see `end`(3C).

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

```
monitor ((int (*)())0, 0, 0, 0, 0);
```

`prof`(1) can then be used to examine the results.

## FILES

```
mon.out
/lib/libp/libc.a
```

/lib/libp/libm.a

**SEE ALSO**

cc(1), prof(1), profil(2), end(3C).

## NAME

mount – mount a file system

## SYNOPSIS

```
int mount (spec, dir, rwflag)
char *spec, *dir;
int rwflag;
```

## DESCRIPTION

mount requests that a removable file system contained on the block special file identified by *spec* be mounted on the directory identified by *dir*. *spec* and *dir* are pointers to path names.

Upon successful completion, references to the file *dir* will refer to the root directory on the mounted file system.

The low-order bit of *rwflag* is used to control write permission on the mounted file system; if 1, writing is forbidden, otherwise writing is permitted according to individual file accessibility. 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.

mount may be invoked only by the superuser.

## ERRORS

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

| | |
|---|---|
| [EPERM] | The effective user ID is not superuser. |
| [ENOENT] | Any of the named files does not exist. |
| [ENOTDIR] | A component of a path prefix is not a directory. |
| [ENOTBLK] | *spec* is not a block special device. |
| [ENXIO] | The device associated with *spec* does not exist. |
| [ENOTDIR] | *dir* is not a directory. |
| [EFAULT] | *spec* or *dir* points outside the allocated address space of the process. |
| [EBUSY] | *dir* is currently mounted on, is someone's current working directory, or is otherwise busy. |
| [EPERM] | A pathname contains a character with the high-order bit set. |

| | |
|---|---|
| [ENAMETOOLONG] | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX. |
| [ELOOP] | Too many symbolic links were encountered in translating a pathname. |
| [EBUSY] | The device associated with *spec* is currently mounted. |
| [EBUSY] | There are no more mount table entries. |

**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.

**SEE ALSO**

`fsmount`(2), `unmount`(2), `umount`(3), `fstab`(4).

## NAME

nlist – get entries from name list

## SYNOPSIS

```
#include <a.out.h>

int nlist(filename, nl)
char *filename;
struct nlist *nl
```

## DESCRIPTION

nlist examines the name list in the executable file whose name is pointed to by *filename*; it selectively extracts a list of values and puts them in the array of nlist structures pointed to by *nl*. The name list *nl* consists of an array of structures containing names of variables, types, and values. The list is terminated with a null name; i.e., a null string is in the name position of the structure. Each variable 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. The type filed will be set to 0 unless the file was compiled with the –g option. If the name is not found, both entries are set to 0. See a.out(4) for a discussion of the symbol table structure.

This function is useful for examining the system name list kept in the file /unix. In this way programs can obtain system addresses that are up to date.

## RETURN VALUE

nlist returns –1 upon error; otherwise it returns 0.

All value entries are set to 0 if the file cannot be read or if it does not contain a valid name list.

## SEE ALSO

a.out(4).

## NAME

perror, errno, sys_errlist, sys_nerr – system error messages

## SYNOPSIS

```
void perror(s)
char *s;

extern int errno;

extern char *sys_errlist[];

extern int sys_nerr;
```

## DESCRIPTION

perror produces a message on the standard error output, describing the last error encountered during a call to a system or library function. The argument string *s* is printed first, then a colon and a blank, then the message and a newline. To be of most use, the argument string should include the name of the program that incurred the error. The error number is taken from the external variable errno, which is set when errors occur but not cleared when nonerroneous calls are made.

To simplify variant formatting of messages, the array of message strings sys_errlist is provided; errno can be used as an index in this table to get the message string without the newline. sys_nerr is the largest message number 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).

## NAME

plot – graphics interface subroutines

## SYNOPSIS

```
int openpl()

int erase()

int label(s)
char *s;

int line(x1, y1, x2, y2)
int x1, y1, x2, y2;

int circle(x, y, r)
int x, y, r;

int arc( x, y, x0, y0, x1, y1)
int x, y, x0, y0, x1, y1;

int move(x, y)
int x, y;

int cont(x, y)
int x, y;

int point(x, y)
int x, y;

int linemod(s)
char *s;

int space(x0, y0, x1, y1)
int x0, y0, x1, y1;

int closepl()
```

## DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. `space` must be used before any of these functions to declare the amount of space necessary; see `plot`(4). `openpl` must be used before any of the others to open the device for writing. `closepl` flushes the output.

`circle` draws a circle of radius *r* with center at the point (*x*,*y*).

`arc` draws an arc of a circle with center at the point (*x*,*y*) between the points (*x0*,*y0*) and (*x1*,*y1*).

String arguments to `label` and `linemod` are terminated by nulls and do not contain newlines.

See plot(4) for a description of the effect of the remaining functions.

The library files listed below provide several variations of these routines.

**FILES**

| | |
|---|---|
| /usr/lib/libplot.a | produces output for tplot(1G) filters |
| /usr/lib/lib300.a | for DASI 300 |
| /usr/lib/lib300s.a | for DASI 300s |
| /usr/lib/lib450.a | for DASI 450 |
| /usr/lib/lib4014.a | for Tektronix 4014 |

**WARNINGS**

To compile a program containing these functions in file.c, use cc file.c -lplot.

To execute it, use a.out | tplot.

The above routines use <stdio.h>. Therefore, the size of programs not otherwise using standard I/O is increased more than might be expected.

**SEE ALSO**

tplot(1G), plot(4).

## NAME

popen, pclose – initiate pipe to/from a process

## SYNOPSIS

```
#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

int pclose(stream)
FILE *stream;
```

## DESCRIPTION

The arguments to popen are pointers to null-terminated strings; one string contains a shell command line and the other contains an I/O mode. The mode may be either "r" for reading or "w" for writing. popen creates a pipe between the calling program and the command to be executed. The value returned is a stream pointer. If the I/O mode is w, one can write to the standard input of the command by writing to the file *stream*; if the I/O mode is "r", one can read from the standard output of the command, by reading from the file *stream*.

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.

## RETURN VALUE

popen returns a NULL pointer if files or processes cannot be created.

pclose returns –1 if *stream* is not associated with a command opened by popen.

## SEE ALSO

pipe(2), wait(2), fclose(3S), fopen(3S), system(3S).

## BUGS

If the original processes and processes opened by popen concurrently read or write a common file, neither should use buffered I/O, because the buffering gets all mixed up. Problems with an output filter may be forestalled by careful buffer flushing, e.g., by using fflush; see fclose(3S).

If an illegal type is passed, popen will fork and exec the command line passed to it before it discovers that the type was illegal. This will result in a NULL pointer being returned and a broken pipe (with the command executing in the background).

NAME

printf, fprintf, sprintf – print formatted output

SYNOPSIS

```
#include <stdio.h>

int printf(format [, arg]...)
char *format;

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

int sprintf(s, format [, arg]...)
char *s, format;
```

DESCRIPTION

printf places output on the standard output stream stdout. fprintf places output on the named output *stream*. sprintf places output, followed by the null character (\ 0) in consecutive bytes starting at *s*; it is the user's responsibility to ensure that enough storage is available.

Each of these functions converts, formats, and prints its *arg*s under control of the *format*. The *format* is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching zero or more *arg*s. The results are undefined if there are insufficient *arg*s for the format. If the format is exhausted while *arg*s remain, the excess *arg*s are simply ignored.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

Zero or more *flags*, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded to the field width on the left (default) or right (if the left-adjustment flag "–" has been given); see below for flag specification. If the field width for an s conversion is preceded by a 0, the string is right adjusted with zero padding on the left.

A *precision* that gives the minimum number of digits to appear for the d, o, u, x, or X conversions, the number of digits to appear after the decimal point for the e and f

conversions, the maximum number of significant digits for the `g` conversion, or the maximum number of characters to be printed from a string in `s` conversion. The format of the precision is a period (`.`) followed by a decimal digit string; a null digit string is treated as zero.

An optional `l` (ell) specifying that a following `d`, `o`, `u`, `x`, or `X` conversion character applies to a long integer *arg*. An `l` before any other conversion character is ignored.

A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (`*`) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen; therefore, the *arg*s specifying field width or precision must appear *before* the *arg* (if any) to be converted.

The flag characters and their meanings are:

– The result of the conversion will be left-justified within the field.

\+ The result of a signed conversion will always begin with a sign (+ or –).

blank If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.

\# This flag specifies that the value is to be converted to an "alternate form." For `c`, `d`, `s`, and `u` conversions, the flag has no effect. For `o` conversion, it increases the precision to force the first digit of the result to be a zero. For `x` (`X`) conversion, a non-zero result will have `0x` (`0X`) prefixed to it. For `e`, `E`, `f`, `g`, and `G` conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For `g` and `G` conversions, trailing zeroes will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:

`d,o,u,x,X` The integer *arg* is converted to signed decimal, unsigned octal, decimal, or hexadecimal notation (`x` and `X`), respectively; the letters `abcdef` are used for `x` conversion and the letters `ABCDEF` for `X` conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. (For compatibility with older versions, padding with leading zeroes may alternatively be specified by prefixing a zero to the field width.) This does not imply an octal value for the field width. The default precision is 1. The result of converting a zero value with a precision of zero is a null string.

`f` The float or double *arg* is converted to decimal notation in the style "`[-]ddd.ddd`", where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are output; if the precision is explicitly 0, no decimal point appears.

`e,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 of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The `E` format code produces a number with `E` instead of `e` introducing the exponent. The exponent always contains at least two digits.

`g,G` The float or double *arg* is printed in style `f` or `e` (or in style `E` in the case of a `G` format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style `e` is used only if the exponent resulting from the conversion is less than –4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

`c` The character *arg* is printed.

`s` The *arg* is taken to be a string (character pointer) and characters from the string are printed until a null character (\ 0) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A NULL value for *arg* yields undefined results.

`%` Print a %; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by `printf` and `fprintf` are printed as if `putc`(3S) had been called.

## RETURN VALUE

Each function returns the number of characters transmitted (not including the \ 0 in the case of `sprintf`), or a negative value if an output error was encountered.

## 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, %.2d:%.2d", weekday, month, day, hour, min);
```

To print *pi* to 5 decimal places:

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

## SEE ALSO

`ecvt`(3C), `intro`(3), `putc`(3S), `scanf`(3S).

## NAME

putc, putchar, fputc, putw – put character or word on a stream

## SYNOPSIS

```
#include <stdio.h>

int putc(c, stream)
int c;
FILE *stream;

int putchar(c)
int c;

int fputc(c, stream)
int c;
FILE *stream;

int putw(w, stream)
int w;
FILE *stream;
```

## DESCRIPTION

The putc macro writes the character *c* onto the output *stream* at the position where the file pointer, if defined, is pointing. The putchar macro is defined as putc(*c*, *stdout*).

fputc behaves like putc, but is a function rather than a macro. fputc runs more slowly than putc, but it takes less space per invocation and its name can be passed as an argument to a function.

putw writes the word (32-bit integer on the Macintosh II) *w* to the output *stream* at the position at which the file pointer, if defined, is pointing. putw neither assumes nor causes special alignment in the file.

Output streams, with the exception of the standard error stream stderr, are by default buffered if the output refers to a file and line-buffered if the output refers to a terminal. The standard error output stream stderr is by default unbuffered, but use of freopen (see fopen(3S)) causes it to become buffered or line-buffered. When an output stream is unbuffered information, it is queued for writing on the destination file or terminal as soon as written; when it is buffered, many characters are saved up and written as a block; when it is line-buffered, each line of output is queued for writing on the destination terminal as soon as the line is completed (i.e., as soon as a newline character is written or terminal input is requested). setbuf(3S) may be used to change

the stream's buffering strategy.

**RETURN VALUE**

On success, these functions each return the value they have written. On failure, they return the constant EOF. This occurs if the file *stream* is not open for writing or if the output file cannot be grown. Because EOF is a valid integer, `ferror`(3S) should be used to detect `putw` errors.

**SEE ALSO**

`fclose`(3S), `ferror`(3S), `fopen`(3S), `fread`(3S), `getc`(3S), `printf`(3S), `puts`(3S), `setbuf`(3S).

**BUGS**

Because it is implemented as a macro, `putc` treats incorrectly a *stream* argument with side effects. In particular, `putc(c, *f++);` doesn't work sensibly. `fputc` should be used instead. Because of possible differences in word length and byte ordering, files written using `putw` are machine-dependent and may not be read using `getw` on a different processor.

## NAME

putenv – change or add value to environment

## SYNOPSIS

```
int  putenv(string)
char  *string;
```

## DESCRIPTION

*string* points to a string of the form "*name=value*". putenv makes the value of the environment variable *name* equal to *value* by altering an existing variable or creating a new one. In either case, the string pointed to by *string* becomes part of the environment, so altering the string will change the environment. The space used by *string* is no longer used once a new string-defining *name* is passed to putenv.

## RETURN VALUE

putenv returns nonzero if it was unable to obtain enough space via malloc for an expanded environment, otherwise zero.

## SEE ALSO

exec(2), getenv(3C), malloc(3C), environ(5).

## WARNINGS

putenv manipulates the environment pointed to by environ, and can be used in conjunction with getenv. However, *envp* (the third argument to *main*) is not changed.

This routine uses malloc(3C) to enlarge the environment.

After putenv is called, environmental variables are not in alphabetical order.

A potential error is to call putenv with an automatic variable as the argument, then exit the calling function while *string* is still part of the environment.

## NAME

putpwent – write password file entry

## SYNOPSIS

```
#include <pwd.h>

int putpwent(p, f)
struct passwd *p;
FILE *f;
```

## DESCRIPTION

putpwent is the inverse of getpwent(3C). Given a pointer to a passwd structure created by getpwent (or getpwuid or getpwnam), putpwuid writes a line on the stream *f* which matches the format of /etc/passwd.

The <pwd.h> header file is described in getpwent(3C).

## RETURN VALUE

putpwent returns nonzero if an error was detected during its operation; otherwise it returns zero.

## SEE ALSO

getpwent(3C).

## WARNING

The above routine uses <stdio.h>. Therefore, the size of programs not otherwise using standard I/O is increased more than might be expected.

## NAME

puts, fputs – put a string on a stream

## SYNOPSIS

```
#include <stdio.h>

int puts(s)
char *s;

int fputs(s, stream)
char *s;
FILE *stream;
```

## DESCRIPTION

puts writes the null-terminated string referenced by *s*, followed by a newline character, to the standard output stream stdout.

fputs writes the null-terminated string pointed to by *s* to the named output *stream*.

Neither function writes the terminating null character.

## SEE ALSO

ferror(3S), fopen(3S), fread(3S), printf(3S), putc(3S).

## RETURN VALUE

On success, both routines return the number of characters written.

Both functions return EOF on error. This occurs if the routines try to write on a file that has not been opened for writing.

## NOTES

puts appends a newline character while fputs does not.

## NAME

qsort – quicker sort

## SYNOPSIS

```
void qsort(base, nel, width, compar)
char *base;
unsigned nel, width;
int(*compar)();
```

## DESCRIPTION

qsort is an implementation of the quicker-sort algorithm. It sorts a table of data in place.

*base* points to the element at the base of the table. *nel* is the number of elements in the table. *width* is the width of an element in bytes. *compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero according as the first argument is to be considered less than, equal to, or greater than the second.

## NOTES

The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared. The order in the output of the two items which compare as equal is unpredictable.

## EXAMPLE

```
struct    entry {
     char  *name;
     int   flags;
};

main()
{
     struct entry hp[100];
     int entcmp();
     int i, count;

     for (i = 0; i < (count = 100); i++) {
          /* fill the structure with the name
                    and flags */
          .
          .
          .
     }
     qsort( (char *) hp, count, sizeof (hp[0]), entcmp);
```

```
}
entcmp(ep,ep2)
struct entry *ep, *ep2;
{
     return (strcmp(ep->name, ep2->name));
}
```

will sort a set of names with associated flags in ASCII order.

**SEE ALSO**

sort(1), bsearch(3C), lsearch(3C), string(3C).

## NAME

rand, srand – simple random-number generator

## SYNOPSIS

```
int  rand()

void  srand(seed)
unsigned  seed;
```

## DESCRIPTION

rand uses a multiplicative congruential random-number generator with period 2 power of 32 that returns successive pseudo-random numbers in the range from 0 to 32767.

srand can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

## NOTE

The spectral properties of rand leave a great deal to be desired. drand48(3C) provides a much better, though more elaborate, random-number generator.

## SEE ALSO

drand48(3C).

## NAME

irand, srand, rand – Fortran uniform random-number generator

## SYNOPSIS

```
call srand(iseed)

i=irand()

x=rand()
```

## DESCRIPTION

irand generates successive pseudo-random numbers in the range from 0 to 2**15-1. rand generates pseudo-random numbers distributed in (0, 1.0). srand uses its integer argument to reinitialize the seed for successive invocations of irand and rand.

## SEE ALSO

rand(3C).

## NAME

rcmd, rresvport, ruserok – routines for returning a stream to a remote command

## SYNOPSIS

```
int rcmd(ahost, inport, locuser, remuser, cmd, fd2p);
char **ahost;
u_short inport;
char *locuser, *remuser, *cmd;
int *fd2p;

int rresvport(port);
int *port;

int ruserok(rhost, superuser, ruser, luser);
char *rhost;
int superuser;
char *ruser, *luser;
```

## DESCRIPTION

rcmd is a routine used by the superuser 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 remshd(1M) server (among others).

rcmd looks up the host **ahost* using gethostent(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 call succeeds, a socket of type SOCK_STREAM is returned to the caller, and given to the remote command as stdin and stdout. If *fd2p* is nonzero, 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 the stderr (descriptor 2 of the remote(1M) command) on this channel, and will accept bytes on this channel as A/UX signal numbers to be forwarded to the process group of the command. If *fd2p* is 0, then the stderr (descriptor 2 of the remote(1M) command) will be made the same as stdout; no provision will be 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 remshd(1M).

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 addresses consist of a port in the range 0 to 1023. Only the superuser is allowed to bind an address of this sort to a socket.

ruserok takes a remote host's name, as returned by a gethostent(3N) routine, two user names and a flag indicating if the local user's name is the superuser. 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 1 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 0. If the *superuser* flag is 1, the checking of the "host.equiv" file is bypassed.

**SEE ALSO**

remsh(1N), rlogin(1N), remshd(1M), rexecd(1M), rlogind(1M), rexec(3N).

**BUGS**

There is no way to specify options to the socket call which rcmd makes.

## NAME

regcmp, regex – compile and execute a regular expression

## SYNOPSIS

```
char *regcmp(string1 [,string2,...], (char *)0))
char *string1, *string2, ...;

char *regex(re, subject [, ret0,...])
char *re, *subject, *ret0 , ...;

extern char *loc1;
```

## DESCRIPTION

regcmp compiles a regular expression and returns a pointer to the compiled form. malloc(3C) is used to create space for the vector. It is the user's responsibility to free unneeded space that has been allocated by malloc. A NULL return from regcmp indicates an incorrect argument. regcmp(1) has been written to generally preclude the need for this routine at execution time.

regex executes a compiled pattern against the subject string. Additional arguments are passed to receive values back. regex returns NULL on failure or a pointer to the next unmatched character on success. A global character pointer loc1 points to where the match began. regcmp and regex were mostly borrowed from the editor, ed(1); however, the syntax and semantics have been changed slightly. The following are the valid symbols and their associated meanings.

`[ ] * . ^` These symbols retain their current meaning.

`$` This symbol matches the end of the string; `\n` matches the newline.

`-` Within brackets the minus means "through." For example, `[a-z]` is equivalent to `[abcd . . .xyz]`. The - can appear as itself only if used as the last or first character. For example, the character class expression `[]-]` matches the characters ] and -.

`+` A regular expression followed by + means "one or more times." For example, `[0-9]+` is equivalent to `[0-9] [0-9]*`.

`{m}` `{m,}` `{m,u}` Integer values enclosed in { } indicate the number of times the preceding regular expression is to be applied. The minimum number is *m* and the maximum number is *u*, which must be less than 256.

If only *m* is present (e.g., {*m*}), it indicates the exact number of times the regular expression is to be applied. {*m*,} is analogous to {*m*,infinity}. The plus (+) and star (*) operations are equivalent to {1,} and {0,}, respectively.

( . . . )$*n*
The value of the enclosed regular expression is to be returned. The value will be stored in the *(n+1)*th argument following the subject argument. At present, at most 10 enclosed regular expressions are allowed. regex makes its assignments unconditionally.

( . . . ) Parentheses are used for grouping. An operator (e.g., *, +, { }) can work on a single character or a regular expression enclosed in parentheses. For example, (a*(cb+)*)$0.

By necessity, all the above defined symbols are special. They must, therefore, be escaped to be used as themselves.

EXAMPLES

Example 1:

```
char *cursor, *newcursor, *ptr;
        . . .
newcursor = regex((ptr = regcmp("^\n", 0)), cursor);
free(ptr);
```

This example will match a leading newline in the subject string pointed at by cursor.

Example 2:

```
char ret0[9];
char *newcursor, *name;
        . . .
name = regcmp("([A-Za-z][A-za-z0-9_]{0,7})$0", 0);
newcursor = regex(name, "123Testing321", ret0);
```

This example will match through the string "Testing3" and will return the address of the character after the last matched character (cursor+11). The string "Testing3" will be copied to the character array ret0.

Example 3:

```
#include "file.i"
char *string, *newcursor;
        . . .
newcursor = regex(name, string);
```

This example applies a precompiled regular expression in `file.i` (see regcmp(1)) against *string*.

This routine is kept in `/lib/libPW.a`.

## SEE ALSO

ed(1), regcmp(1), malloc(3C).

## BUGS

The user program may run out of memory if regcmp is called iteratively without freeing the vectors no longer required. The following user-supplied replacement for malloc(3C) reuses the same vector, saving time and space:

```
/* user's program */
        . . .
char *
malloc(n)
unsigned n;
{
        static char rebuf[512];
        return (n <= sizeof rebuf) ? rebuf : NULL;
}
```

**NAME**

res_mkquery, res_send, res_init, dn_comp, dn_expand – resolver routines

**SYNOPSIS**

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

res_mkquery(op, dname, class, type, data, datalen,
            newrr, buf, buflen)
int op;
char *dname;
int class, type;
char *data;
int datalen;
struct rrec *newrr;
char *buf;
int buflen;

res_send(msg, msglen, answer, anslen)
char *msg;
int msglen;
char *answer;
int anslen;

res_init()

dn_comp(exp_dn, comp_dn, length, dnptrs, lastdnptr)
char *exp_dn, *comp_dn;
int length;
char **dnptrs, **lastdnptr;

dn_expand(msg, 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 authoritative answer or finds an error. Currently this is not implemented.

RES_USEVC
: Use TCP connections for queries instead of UDP.

RES_STAYOPEN
: Used with RES_USEVC to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used.

RES_IGNTC
: Unused currently (ignore truncation errors, i.e., don't retry with TCP).

RES_RECURSE
: Set the recursion desired bit in queries. This is the default. (res_send does not do iterative queries and expects the name server to handle recursion.)

RES_DEFNAMES
: Append the default domain name to single label queries. This is the default.

res_init

reads the initialization file to get the default domain name and the Internet address of the initial hosts running the name server. If this line does not exist, the host running the resolver is tried. res_mkquery makes a standard query message and places it in *buf*. res_mkquery will return the size of the query or -1 if the query is larger than *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 array pointed to by `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 ALSO**

`named`(1M), `resolver`(4).

## NAME

rexec – return stream to a remote command

## SYNOPSIS

```
int rexec(ahost, inport, user, passwd, cmd, fd2p);
char **ahost;
u_short inport;
char *user, *passwd, *cmd;
int *fd2p;
```

## DESCRIPTION

rexec looks up the host **ahost* using gethostent(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; it will normally be the value returned from the call "getservbyname("exec", "tcp")" (see getservent(3N)). The protocol for connection is described in detail in rexecd(1M).

If the call succeeds, a socket of type SOCK_STREAM is returned to the caller, and given to the remote command as stdin and stdout. If *fd2p* is nonzero, then a 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 A/UX 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.

## SEE ALSO

rcmd(3N), rexecd(1M).

## BUGS

There is no way to specify options to the socket call which rexec makes.

**NAME**

anint, dnint, nint, idnint – Fortran nearest integer functions

**SYNOPSIS**

```
integer i
real r1, r2
double precision dp1, dp2

r2=anint(r1)
i=nint(r1)

dp2=anint(dp1)
dp2=dnint(dp1)

i=nint(dp1)
i=idnint(dp1)
```

**DESCRIPTION**

anint returns the nearest whole real number to its real argument (i.e., int(a+0.5) if a ≥ 0, int(a–0.5) otherwise). dnint does the same for its double-precision argument. nint returns the nearest integer to its real argument. idnint is the double-precision version. anint is the generic form of anint and dnint, performing the same operation and returning the data type of its argument. nint is also the generic form of idnint.

## NAME

rpc – library routines for remote procedure calls

## DESCRIPTION

These routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a data packet to the server. Upon receipt of the packet, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

## FUNCTIONS

| | |
|---|---|
| `auth_destroy()` | destroy authentication information handle |
| `authnone_create()` | return RPC authentication handle with no checking |
| `authunix_create()` | return RPC authentication handle with A/UX permissions |
| `authunix_create_default()` | return default A/UX authentication handle |
| `callrpc()` | call remote procedure, given [*prognum,versnum,procnum*] |
| `clnt_broadcast()` | broadcast remote procedure call everywhere |
| `clnt_call()` | call remote procedure associated with client handle |
| `clnt_destroy()` | destroy client's RPC handle |
| `clnt_freeres()` | free data allocated by RPC/XDR system when decoding results |
| `clnt_geterr()` | copy error information from client handle to error structure |
| `clnt_pcreateerror()` | print message to stderr about why client handle creation failed |
| `clnt_perrno()` | print message to stderr corresponing to condition |

| | |
|---|---|
| | given |
| `clnt_perror()` | print message to stderr about why RPC call failed |
| `clnt_sperrno()` | print message to a string corresponding to condition given |
| `clnt_sperror()` | print message to a string |
| `clntraw_create()` | create toy RPC client for simulation |
| `clnttcp_create()` | create RPC client using TCP transport |
| `clntudp_create()` | create RPC client using UDP transport |
| `get_myaddress()` | get the machine's IP address |
| `pmap_getmaps()` | return list of RPC program-to-port mappings |
| `pmap_getport()` | return port number on which waits supporting service |
| `pmap_rmtcall()` | instructs portmapper to make an RPC call |
| `pmap_set()` | establish mapping between [prognum,versnum,procnum] and port |
| `pmap_unset()` | destroy mapping between [*prognum,versnum,procnum*] and port |
| `registerrpc()` | register procedure with RPC service package |
| `rpc_createerr` | global variable indicating reason why client creation failed |
| `svc_destroy()` | destroy RPC service transport handle |
| `svc_fds` | global variable with RPC service file descriptor mask |

| | |
|---|---|
| `svc_freeargs()` | free data allocated by RPC/XDR system when decoding arguments |
| `svc_getargs()` | decodes the arguments of an RPC request |
| `svc_getcaller()` | get the network address of the caller of a procedure |
| `svc_getreq()` | returns when all associated sockets have been serviced |
| `svc_register()` | associates prognum and versnum with service dispatch procedure |
| `svc_run()` | wait for RPC requests to arrive and call appropriate service |
| `svc_sendreply()` | send back results of a remote procedure call |
| `svc_unregister()` | remove mapping of [prognum,versnum] to dispatch routines |
| `svcerr_auth()` | called when refusing service because of authentication error |
| `svcerr_decode()` | called when service cannot decode its parameters |
| `svcerr_noproc()` | called when service hasn't implemented the desired procedure |
| `svcerr_noprog()` | called when program is not registered with RPC package |
| `svcerr_progvers()` | called when version is not registered with RPC package |
| `svcerr_systemerr()` | called when service detects system error |

| | |
|---|---|
| `svcerr_weakauth()` | called when refusing service because of insufficient authentication |
| `svcraw_create()` | creates a toy RPC service transport for testing |
| `svctcp_create()` | creates an RPC service based on TCP transport |
| `svcudp_create()` | creates an RPC service based on UDP transport |
| `xdr_accepted_reply()` | generates RPC-style replies without using RPC package |
| `xdr_authunix_parms()` | generates A/UX credentials without using RPC package |
| `xdr_callhdr()` | generates RPC-style headers without using RPC package |
| `xdr_callmsg()` | generates RPC-style messages without using RPC package |
| `xdr_opaque_auth()` | describes RPC messages, externally |
| `xdr_pmap()` | describes parameters for portmap procedures, externally |
| `xdr_pmaplist()` | describes a list of port mappings, externally |
| `xdr_rejected_reply()` | generates RPC-style rejections without using RPC package |
| `xdr_replymsg()` | generates RPC-style replies without using RPC package |
| `xprt_register()` | registers RPC service transport with RPC package |
| `xprt_unregister()` | unregisters RPC service transport from RPC package |

**SEE ALSO**
*A/UX Network Applications Programming.*

## NAME

scandir – scan a directory

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/dir.h>

scandir(dirname, namelist, select, compar)
char *dirname;
struct direct *(*namelist[]);
int (*select)();
int (*compar)();

alphasort(d1, d2)
struct direct **d1, **d2;
```

## DESCRIPTION

scandir reads the directory *dirname* and builds an array of pointers to directory entries using malloc(3). It returns the number of entries in the array and a pointer to the array through *namelist*.

The *select* parameter is a pointer to a user supplied subroutine which is called by scandir to select which entries are to be included in the array. The select routine is passed a pointer to a directory entry and should return a non-zero value if the directory entry is to be included in the array. If *select* is null, then all the directory entries will be included.

The *compar* parameter is a pointer to a user supplied subroutine which is passed to qsort(3) to sort the completed array. If this pointer is null, the array is not sorted. alphasort is a routine which can be used for the *compar* parameter to sort the array alphabetically.

The memory allocated for the array can be deallocated with free (see malloc(3)) by freeing each pointer in the array and the array itself.

## RETURN VALUE

Returns –1 if the directory cannot be opened for reading or if cannot allocate enough memory to hold all the data structures.

## SEE ALSO

directory(3), malloc(3C), malloc(3X), qsort(3C), dir(4).

## NAME

scanf, fscanf, sscanf – convert formatted input

## SYNOPSIS

```
#include <stdio.h>

int scanf(format [, pointer]...)
char *format;

int fscanf(stream, format [, pointer]...)
FILE *stream;
char *format;

int 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 at *s*. Each function reads characters, interprets them according to *format*, and stores the results in the location specified by the *pointer* arguments. Each function 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. White-space characters (blanks and tabs) which, except in two cases described below, cause input to be read up to the next nonwhite-space character.
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 suppression character *, an optional numerical maximum field width, an optional l (ell) or h indicating the size of the receiving variable, and a conversion code.

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 has been indicated by *. The suppression of assignment provides a way of describing an input field which is to be skipped. An input field is defined as a string of nonwhite-space characters; it extends to the next inappropriate character or until the field width, if specified, is

exhausted. For all descriptors except "[" and "c", white space leading an input field is ignored.

The conversion code indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument should be given. The following conversion codes 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.

`u` An unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.

`o` An octal integer is expected; the corresponding argument should be an integer pointer.

`x` A hexadecimal integer is expected; the corresponding argument should be an integer pointer.

`e,f,g` 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 an `e`, followed by an optional +, –, or space followed by an integer.

`s` A character string is expected; the corresponding argument should be a character pointer to an array of characters large enough to accept the string and a terminating `\0`, which will be added automatically. The input field is terminated by a white-space character.

`c` A character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next nonspace character, use `%1s`. If a field width is given, the corresponding argument should refer to a character array; the indicated number of characters is read.

`[` String data and the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters (the *scanset*) and a right bracket; the input field is the maximal sequence of input characters consisting entirely of

characters in the *scanset*. The caret, (^), when it appears as the first character in the *scanset*, serves as a complement operator and redefines the *scanset* as the set of all characters *not* contained in the remainder of the *scanset* string. There are some conventions used in the construction of the *scanset*. A range of characters may be represented by the construct *first-last*; thus, `[0123456789]` may be expressed `[0-9]`. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The dash will also stand for itself whenever it is the first or the last character in the *scanset*. To include the right square bracket as an element of the *scanset*, it must appear as the first character (possibly preceded by a circumflex) of the *scanset*; otherwise it will be interpreted syntactically as the closing bracket. The corresponding argument must point to a character array large enough to hold the data field and the terminating `\0`, which will be added automatically. At least one character must match for this conversion to be considered successful.

The conversion characters `d`, `u`, `o`, and `x` may be preceded by `l` or `h` to indicate that a pointer to `long` or `short`, rather than `int`, is in the argument list. Similarly, the conversion characters `e`, `f`, and `g` may be preceded by `l` to indicate that a pointer to `double`, rather than `float`, is in the argument list.

The `l` or `h` modifier is ignored for other conversion characters. `scanf` conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

`scanf` returns the number of successfully matched and assigned input items; this number can be zero when an early conflict between an input character and the control string occurs. If the input ends before the first conflict or conversion, EOF is returned.

## EXAMPLES

The call:

```
int i; n; float x; char name[50];
n =scanf ("%d%f%s", &i, &x, name);
```

with the input line

```
25 54.32E-1 thompson
```

will assign the value 3 to *n*, the value 25 to *i*, and the value 5.432 to *x*; *name* will contain thompson\0.

The call

```
int i; float x; char name[50];
(void) scanf ("%2d%f%*d %[0-9]", &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 (see getc(3S)) will return a.

**RETURN VALUE**

These functions return EOF on end of input and a short count for missing or illegal data items.

**NOTE**

Trailing white space is left unread unless matched in the control string.

**BUGS**

The success of literal matches and suppressed assignments is not directly determinable.

**SEE ALSO**

getc(3S), printf(3S), strtod(3C), strtol(3C).

## NAME

set42sig – set 4.2 BSD signal interface

## SYNOPSIS

```
int set42sig()
```

## DESCRIPTION

set42sig changes the signal interface to one closely resembling BSD 4.2 systems. This call is similar to the setcompat system call. Unlike setcompat(2), set42sig arranges for the current compatibility flags to be logically OR'ed with the new flags. set42sig is functionally equivalent to the following C code fragment:

```
#include <compat.h>

return (setcompat(getcompat() | COMPAT_BSDSIGNALS |
        COMPAT_BSDTTY | COMPAT_BSDSYSCALLS));
```

For the process calling it, it enables reliable signal delivery, the job control tty signals, and restarting of system calls when an interrupt is received.

If the COMPAT_SVID flag is set before calling set42sig, both BSD 4.2 and System V modes are set and 4.2 BSD mode will have precedence. COMPAT_SVID can be set in two ways, by calling setcompat(2) and by compiling the program with the -ZS flag option (see cc(1).

All aspects of 4.2 signals are inherited across fork system calls. 4.2 job control group membership is inherited across exec system calls. When exec is invoked, the inherited 4.2 signals are lost and the signal-handling mechanism returns to System V style. See setcompat(2) for more information.

## ERRORS

| | |
|---|---|
| [EINVAL] | The process has already arranged to catch signals. Normally set42sig is called prior to any other signal activity. |

## SEE ALSO

cc(1), setcompat(2), sigvec(2), signal(3), termio(7).

## NAME

setbuf, setvbuf – assign buffering to a stream

## SYNOPSIS

```
#include <stdio.h>

void setbuf(stream, buf)
FILE *stream;
char *buf;

int setvbuf(stream, buf, type, size)
FILE *stream;
char *buf;
int type, size;
```

## DESCRIPTION

setbuf may be used after a stream has been opened but before it is read or written. It causes the array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is the NULL pointer input/output will be completely unbuffered.

A constant BUFSIZ, defined in the <stdio.h> header file, tells how big an array is needed:

```
char buf[BUFSIZ];
```

setvbuf may be used after a stream has been opened but before it is read or written. *type* determines how *stream* will be buffered. Legal values for *type* (defined in stdio.h) are:

| | |
|---|---|
| _IOFBF | causes input/output to be fully buffered. |
| _IOLBF | causes output to be line buffered; the buffer will be flushed when a newline is written, the buffer is full, or input is requested. |
| _IONBF | causes input/output to be completely unbuffered. |

If *buf* is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. *size* specifies the size of the buffer to be used. The constant BUFSIZ in <stdio.h> is suggested as a good buffer size. If input/output is unbuffered, *buf* and *size* are ignored.

By default, output to a terminal is line buffered and all other input/output is fully buffered.

## RETURN VALUE

If an illegal value for *type* or *size* is provided, setvbuf returns a nonzero value. Otherwise, the value returned will be zero.

**SEE ALSO**

fopen(3S), getc(3S), intro(3), malloc(3C), putc(3S).

**NOTE**

A common source of error is allocating buffer space as an "automatic" variable in a code block, and then failing to close the stream in the same block.

setbuf allows assignment of a new I/O buffer after the stream has been read (written), and if unflushed data remains in the original buffer. This could lead to a loss of data error.

**NAME**

setjmp, longjmp – non-local goto

**SYNOPSIS**

```
#include <setjmp.h>

int setjmp(env)
jmp_buf env;

void longjmp(env, val)
jmp_buf env;
int val;
```

**DESCRIPTION**

These functions are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

`setjmp` saves its stack environment in *env* for later use by `longjmp`. The environment type `jmp_buf` is defined in the `<setjmp.h>` header file.

**RETURN VALUE**

When `setjmp` has been called by the calling process, returns 0.

`longjmp` restores the environment saved by the last call of `setjmp` with the corresponding *env* argument. After `longjmp` is completed, program execution continues as if the corresponding call of `setjmp` (which must not itself have returned in the interim) had just returned the value *val*. `longjmp` cannot cause `setjmp` to return the value 0. If `longjmp` is invoked with a second argument of 0, `setjmp` will return 1. All accessible data have values as of the time `longjmp` was called.

**SEE ALSO**

`signal`(3).

**WARNING**

`longjmp` fails if it is called when *env* was never primed by a call to `setjmp` or when the last such call is in a function which has since returned.

## NAME

setuid, setgid – set user and group IDs

## SYNOPSIS

```
int  setuid(uid)
int  uid;

int  setgid(gid)
int  gid;
```

## DESCRIPTION

setuid (setgid) is used to set the real user (group) ID and effective user (group) ID of the calling process.

If the effective user ID of the calling process is superuser, the real user (group) ID and effective user (group) ID are set to *uid* (*gid*).

If the effective user ID of the calling process is not superuser, but its real user (group) ID is equal to *uid* (*gid*), the effective user (group) ID is set to *uid* (*gid*).

If the effective user ID of the calling process is not superuser, but the saved set-user (group) ID from exec(2) is equal to *uid* (*gid*), the effective user (group) ID is set to uid(*gid*).

## 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

setuid (setgid) will fail if one of the following is true:

| | |
|---|---|
| [EPERM] | the real user (group) ID of the calling process is not equal to *uid* (*gid*) and its effective user ID is not superuser. |
| [EINVAL] | The *uid* (*gid*) is out of range. |

## SEE ALSO

getuid(2), intro(2), setregid(2), setreuid(2).

## NAME

sign, isign, dsign – Fortran transfer-of-sign intrinsic function

## SYNOPSIS

```
integer i, j, k
real r1, r2, r3
double precision dp1, dp2, dp3

k=isign(i, j)
k=sign(i, j)

r3=sign(r1, r2)

dp3=dsign(dp1, dp2)
dp3=sign(dp1, dp2)
```

## DESCRIPTION

isign returns the magnitude of its first argument with the sign of its second argument. sign and dsign are its real and double-precision counterparts, respectively. The generic version is sign, which devolves to the appropriate type depending on its arguments.

## NAME

signal – specify what to do upon receipt of a signal

## SYNOPSIS

```
#include <signal.h>

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

## DESCRIPTION

signal allows the calling process to choose one of three ways in which it is possible to handle the receipt of a specific signal. *sig* specifies the signal and *func* specifies the choice.

*sig* can be assigned any one of the following except SIGKILL:

| | | |
|---|---|---|
| SIGHUP | 1 | hangup |
| SIGINT | 2 | interrupt |
| SIGQUIT | 3* | quit |
| SIGILL | 4* | illegal instruction |
| SIGTRAP | 5* | trace trap |
| SIGIOT | 6* | IOT instruction |
| SIGEMT | 7* | EMT instruction |
| SIGFPE | 8* | floating point exception |
| SIGKILL | 9 | kill (cannot be caught, blocked, or 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 |
| SIGUSR1 | 16 | user defined signal 1 |
| SIGUSR2 | 17 | user defined signal 2 |
| SIGCLD | 18• | child status has changed |
| SIGPWR | 19 | power-fail restart |
| SIGTSTP | 20† | stop signal generated from keyboard |
| SIGTTIN | 21† | background read attempted from control terminal |
| SIGTTOU | 22† | background write attempted to control terminal |
| SIGSTOP | 23† | stop (cannot be caught, blocked, or ignored) |
| SIGXCPU | 24 | cpu time limit exceeded |
| SIGXFSZ | 25 | file size limit exceeded |
| SIGVTALRM | 26 | virtual time alarm (see setitimer(2)) |
| SIGPROF | 27 | profiling timer alarm (see setitimer(2)) |
| SIGWINCH | 28• | window size change |
| SIGCONT | 29• | continue after stop (cannot be blocked) |

| | | |
|---|---|---|
| SIGURG | 30• | urgent condition present on socket |
| SIGIO | 31• | I/O is possible on a descriptor (see fcntl(2)) |

The starred signals in the above list cause a core image if not caught or ignored (see below).

Signals marked with • are discarded if the action is SIG_DFL; signals marked with † cause the process to stop if the process is part of 4.2 job control.

*func* is assigned one of three values: SIG_DFL, SIG_IGN, or a *function-address*. The actions prescribed by these values are as follows:

SIG_DFL – terminate process upon receipt of a signal

Upon receipt of the signal *sig*, the receiving process is to be terminated with the following consequences:

All of the receiving process's open file descriptors will be closed.

If the parent process of the receiving process is executing a wait, it will be notified of the termination of the receiving process and the terminating signal's number will be made available to the parent process; see wait(2).

If the parent process of the receiving process is not executing a wait, the receiving process will be transformed into a zombie process (see exit(2) for definition of zombie process).

The parent process ID of each of the receiving process's existing child processes and zombie processes will be set to 1. This means the initialization process (see intro(2)) inherits each of these processes.

Each attached shared memory segment is detached and the value of shm_nattach in the data structure associated with its shared memory identifier is decremented by 1.

For each semaphore for which the receiving process has set a semadj value (see semop(2)), that semadj value is added to the semval of the specified semaphore.

If the process has a process, text, or data lock, an unlock is performed (see plock(2)).

An accounting record will be written on the accounting file if the system's accounting routine is enabled; see acct(2).

If the receiving process's process ID, tty group ID, and process group ID are equal, the signal SIGHUP will be sent to all of the processes that have a process group ID equal to the process group ID of the receiving process.

A "core image" will be made in the current working directory of the receiving process if *sig* is one for which an asterisk appears in the above list *and* the following conditions are met:

The effective user ID and the real user ID of the receiving process are equal.

An ordinary file named core exists and is writable or can be created. If the file must be created, it will have the following properties:

a mode of 0666 modified by the file creation mask (see umask(2))

a file owner ID that is the same as the effective user ID of the receiving process

a file group ID that is the same as the effective group ID of the receiving process

SIG_IGN – ignore signal
The signal *sig* is to be ignored.

*Note:* The signal SIGKILL cannot be ignored.

*function-address* – catch signal
Upon receipt of the signal *sig*, the receiving process is to execute the signal-catching function pointed to by *func*. The signal number *sig* will be passed as the only argument to the signal-catching function. Additional arguments are passed to the signal-catching function for hardware-generated signals. Before entering the signal-catching function, the value of *func* for the caught signal will be set to SIG_DFL unless the signal is SIGILL, SIGTRAP, or SIGPWR.

Upon return from the signal-catching function, the receiving process will resume execution at the point it was interrupted.

When a signal that is to be caught occurs during a read, a write, an open, or an ioctl system call on a slow

device (like a terminal; but not a file), during a pause system call, or during a wait system call that does not return immediately due to the existence of a previously stopped or zombie process, the signal-catching function will be executed and then the interrupted system call may return a –1 to the calling process with errno set to EINTR. This behavior is the default for 5.2 systems and it may be modified by the setcompat(2) system call.

*Note:* The signal SIGKILL cannot be caught.

A call to signal cancels a pending signal *sig* except for a pending SIGKILL signal.

**RETURN VALUE**

Upon successful completion, signal returns the previous value of *func* for the specified signal *sig*. Otherwise, a value of –1 is returned and errno is set to indicate the error.

**ERRORS**

signal will fail if:

[EINVAL] *sig* is an illegal signal number, including SIGKILL.

**WARNING**

Two other signals that behave differently than the signals described above exist in this release of the system; they are:

| | | |
|---|---|---|
| SIGCLD | 18 | death of a child (reset when caught) |
| SIGPWR | 19 | power fail (not reset when caught) |

There is no guarantee that, in future releases of the UNIX system, these signals will continue to behave as described below; they are included only for compatibility with other versions of the UNIX system. Their use in new programs is strongly discouraged.

For these signals, *func* is assigned one of three values: SIG_DFL, SIG_IGN, or a *function-address*. The actions prescribed by these values of are as follows:

SIG_DFL - ignore signal
The signal is to be ignored.

SIG_IGN - ignore signal
The signal is to be ignored. Also, if *sig* is SIGCLD, the calling process's child processes will not create zombie processes when they terminate; see exit(2).

*function-address* - catch signal

If the signal is SIGPWR, the action to be taken is the same as that described above for *func* equal to *function-address*. The same is true if the signal is SIGCLD except, that while the process is executing the signal-catching function, any received SIGCLD signals will be queued and the signal-catching function will be continually reentered until the queue is empty.

The SIGCLD affects two other system calls (wait(2), and exit(2)) in the following ways:

wait — If the *func* value of SIGCLD is set to SIG_IGN and a wait is executed, the wait will block until all of the calling process's child processes terminate; it will then return a value of –1 with errno set to ECHILD.

exit — If in the exiting process's parent process the *func* value of SIGCLD is set to SIG_IGN, the exiting process will not create a zombie process.

When processing a pipeline, the shell makes the last process in the pipeline the parent of the proceeding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should take care not to set SIGCLD to be caught.

SEE ALSO

kill(1), kill(2), pause(2), ptrace(2), setcompat(2), sigvec(2), wait(2), set42sig(3), setjmp(3C).

BUGS

If a repeated signal arrives before the last one can be reset, there is no chance to catch it. However, see the setcompat flag COMPAT_BSDSIGNALS.

The type specification of the routine and its *func* argument are problematical.

The symbols sighnd and sigtrap are globally defined symbols used by signal and are reserved words.

## NAME

signal – specify Fortran action on receipt of a system signal

## SYNOPSIS

```
integer i
external integer intfnc
call signal(i, intfnc)
```

## DESCRIPTION

signal allows a process to specify a function to be invoked upon receipt of a specific signal. The first argument specifies a fault or exception; the second argument specifies the function to be invoked.

## SEE ALSO

kill(2), signal(3).

## NAME

sin, dsin, csin – Fortran sine intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2
complex cx1, cx2

r2=sin(r1)

dp2=dsin(dp1)
dp2=sin(dp1)

cx2=csin(cx1)
cx2=sin(cx1)
```

## DESCRIPTION

sin returns the real sine of its real argument. dsin returns the double-precision sine of its double-precision argument. csin returns the complex sine of its complex argument. The generic sin function becomes dsin or csin as required by argument type.

## SEE ALSO

trig(3M).

## NAME

sinh, dsinh – Fortran hyperbolic sine intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2

r2=sinh(r1)

dp2=dsinh(dp1)
dp2=sinh(dp1)
```

## DESCRIPTION

sinh returns the real hyperbolic sine of its real argument. dsinh returns the double-precision hyperbolic sine of its double-precision argument. The generic form sinh may be used to return a double-precision value given a double-precision argument.

## SEE ALSO

sinh(3M).

## 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

sinh, cosh, and tanh return, respectively, the hyberbolic sine, cosine, and tangent of their argument.

## RETURN VALUE

sinh and cosh return HUGE (and sinh may return –HUGE for negative *x*) when the correct value would overflow and set errno to ERANGE.

These error-handling procedures may be changed with the function matherr(3M).

## SEE ALSO

matherr(3M).

## NAME

sleep – suspend execution for interval

## SYNOPSIS

```
unsigned sleep(seconds)
unsigned seconds;
```

## DESCRIPTION

sleep suspends the current process from execution for the number of *seconds* specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) scheduled wakeups occur at fixed 1-second intervals, (on the second, according to an internal clock) and (2) any caught signal will terminate sleep following execution of the signal catching routine. The suspension time may be longer than requested by an arbitrary amount, due to the scheduling of other activity in the system. The value returned by sleep is the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested sleep time or in case there is premature arousal due to another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling sleep. If the sleep time exceeds the time before the alarm signal, the process sleeps only until the alarm signal would have occurred and the caller's alarm catch routine is executed just before the sleep routine returns. If the sleep time is less than the time before the calling program's alarm, the prior alarm time is reset to go off at the same time it would have without the intervening sleep.

## SEE ALSO

alarm(2), pause(2), signal(3).

**NAME**

slots – ROM library functions

**SYNTAX**

cc [*flags*]*files* -lslots [*libraries*]

**DESCRIPTION**

The routines in the slots library provide access to board slot ROM from either user or kernel processes. Calls to library routines do not require knowledge of either the board ROM configuration or the ROM addressing requirements.

**USER FUNCTIONS**

slot_PRAM_init(*slot, data*)
Read the PRAM init structure for *slot* into the buffer pointed to by *data.*

slot_board_flags(*slot*)
Read and return the board flags for *slot.*

slot_board_id(*slot*)
Read and return the board ID number for *slot.*

slot_board_name(*slot, data, size*)
Read up to *size* bytes of the board name string for *slot* into the buffer pointed to by *data.*

slot_board_type(*slot, data*)
Read and return the unsigned 64 bit or 8 byte board type for *slot* into the buffer pointed to by *data.*

slot_ether_addr(*slot, data*)
For *slot* read 6 bytes of ethernet address into the buffer pointed to by *data.*

slot_primary_init(*slot, data*)
For *slot* read the primary init structure into the buffer pointed to by *data.*

slot_part_num(*slot, data, size*)
For *slot* get *size* bytes of the part number string into the buffer pointed to by *data.*

slot_rev_level(*slot, data, size*)
For *slot* get *size* bytes of the revision level of the ROM into the buffer pointed to by *data.*

slot_serial_number(*slot, data, size*)
For *slot* get *size* bytes of serial number string into the buffer pointed to by *data.*

`slot_vendor_id(`*slot, data, size*`)`
For *slot* read *size* bytes of vendor ID string into the buffer pointed to by *data.*

UTILITY FUNCTIONS

`slot_board_vendor_info(`*kind, slot, data, size*`)`
For *slot* get *size* bytes of the vendor information string of type *kind* into the buffer pointed to by *data.*

`slot_byte(`*address*`)`
Return the byte located at *address.*

`slot_data(`*slot, kind, request, data,* `size)`
For *slot*lot, read *size* BITS of data for resource of type *kind* from the resource list item of type *request* and put it into the location pointed to by *data.*

`slot_directory(`*slot, data, size*`)`
For *slot* read the resource directory into the buffer of *size* entries pointed to by *data.*

`slot_long(`*address, data*`)`
Return 32 bits of data from *address* offset by *data.*

`slot_resource(`*address, kind,* `request,` *data, size*`)`
For ROM starting at base *address* read *size* bytes of the *request* resource item from the *kind* resource into the buffer pointed to by *data.*

`slot_resource_list(`*address, kind, data, size*`)`
For ROM starting at base *address* read *size* entries of resource list of *kind* into the buffer pointed to by *data.*

`slot_structure(`*address, from, data, size*`)`
From ROM starting at *address* plus the offset in parameter *from* read *size* bytes of data into the buffer pointed to by *data.*

`slot_word(`*address*`)`
Return 16 bits of data located at *address.*

LOW LEVEL FUNCTIONS

`slot_seg_violation()`
This routine is passed to `slot_catch` to handle bus errors.

`slot_catch(`*kind, routine*`)`
Setup *routine* to handle interrupts of type *kind.*

`slot_ignore(`*kind*`)`
Return the system to default handling of interrupts of type *kind.*

`slot_address` (*slot*)
Returns a computed ROM base address for *slot*.

`slot_bytelane` (*address, bytelane*)
Return the ROM bytelane byte into *bytelane* for ROM starting at *address*.

`slot_calc_pointer` (*current, offset*)
Return a ROM pointer *offset* bytes from *current*.

`slot_rom_data` (*address, width, data*)
Starting with *address* fill the buffer pointed to by *data* with *width* bytes of data.

`slot_check_crc` (*top, fhp, bytelane*)
Check the CRC for the ROM with base address *top* using the format header information pointed to by *fhp* and the byte lane information in *bytelane*.

`slot_header` (*address, format_hdrp*)
Read the ROM format header into the buffer pointed to by *format_hdrp* for the ROM starting at base address *address*.

**SEE ALSO**
*Writing A/UX Device Drivers*

**BUGS**
The slots library is only accessible to processes with superuser privileges due to the required `phys` call to access board ROM.

## NAME

sputl, sgetl – access long integer data in a machine independent fashion

## SYNOPSIS

```
void  sputl(value,  buffer)
long  value;
char  *buffer;

long  sgetl(buffer)
char  *buffer;
```

## DESCRIPTION

sputl takes the 4 bytes of the long integer *value* and places them in memory, starting at the address pointed to by *buffer*. The ordering of the bytes is the same across all machines.

sgetl retrieves the 4 bytes in memory, starting at the address pointed to by *buffer*, and returns the long integer value in the byte ordering of the host machine.

Use of sputl and sgetl provide a machine independent way of storing long numeric data in a file in binary form without conversion to characters.

A program that uses these functions must be loaded with the object file access routine library libld.a.

## SEE ALSO

ar(4).

## NAME

sqrt, dsqrt, csqrt – Fortran square root intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2
complex cx1, cx2

r2=sqrt(r1)

dp2=dsqrt(dp1)
dp2=sqrt(dp1)

cx2=csqrt(cx1)
cx2=sqrt(cx1)
```

## DESCRIPTION

sqrt returns the real square root of its real argument. dsqrt returns the double-precision square root of its double-precision argument. csqrt returns the complex square root of its complex argument. sqrt, the generic form, will become dsqrt or csqrt as required by its argument type.

## SEE ALSO

exp(3M).

## NAME

ssignal, gsignal – software signals

## SYNOPSIS

```
#include <signal.h>

int (*ssignal(sig, action))()
int sig, (*action)();

int gsignal(sig)
int sig;
```

## DESCRIPTION

ssignal and gsignal implement a software facility similar to signal(3). This facility is used by the Standard C Library to enable users to indicate the disposition of error conditions; it is also made available to users for their own purposes.

Software signals made available to users are associated with integers in the inclusive range 1 through 15. A call to ssignal associates a procedure, *action*, with the software signal, *sig*; the software signal, *sig*, is raised by a call to gsignal. Raising a software signal causes the action established for that signal to be taken.

The first argument to ssignal is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a user-defined *action* function or one of the manifest constants SIG_DFL (default) or SIG_IGN (ignore). ssignal returns the action previously established for that signal type; if no *action* has been established or the signal number (*sig*) is illegal, ssignal returns SIG_DFL.

gsignal raises the signal identified by its argument, *sig*:

If an *action* function has been established for *sig*, then that *action* is reset to SIG_DFL and the *action* function is entered with argument *sig*. gsignal returns the value returned to it by the *action* function.

If the *action* for *sig* is SIG_IGN, gsignal returns the value 1 and takes no other action.

If the *action* for *sig* is SIG_DFL, gsignal returns the value 0 and takes no other action.

If *sig* has an illegal value or no *action* was ever specified for *sig*, gsignal returns the value 0 and takes no other action.

**SEE ALSO**

sigvec(2), signal(3).

**NOTES**

There are some additional signals with numbers outside the range 1 through 15 which are used by the Standard C Library to indicate error conditions. Thus, some signal numbers outside the range 1 through 15 are legal, although their use may interfere with the operation of the Standard C Library.

**NAME**

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok – string operations

**SYNOPSIS**

```
#include <string.h>

char *strcat(s1,s2)
char *s1, *s2;

char *strncat(s1,s2, n)
char *s1, *s2;
int n;

int strcmp(s1,s2)
char *s1, *s2;

int strncmp(s1,s2, n)
char *s1, *s2;
int n;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;
int n;

int strlen(s)
char *s;

char *strchr(s, c)
char *s;
int c;

char *strrchr(s, c)
char *s;
int c;

char *strpbrk(s1,s2)
char *s1, *s2;

int strspn(s1,s2)
char *s1, *s2;

int strcspn(s1,s2)
char *s1, *s2;

char *strtok(s1,s2)
char *s1, *s2;
```

## DESCRIPTION

The arguments *s1*, *s2*, and *s* point to strings (arrays of characters terminated by a null character). The functions `strcat`, `strncat`, `strcpy`, and `strncpy` all alter *s1*. These functions do not check for overflow of the array pointed to by *s1*.

`strcat` appends a copy of string *s2* to the end of string *s1*. `strncat` appends at most *n* characters. Each function returns a pointer to the null-terminated result.

`strcmp` performs a lexicographical comparison of its arguments and returns an integer less than, equal to, or greater than 0, when *s1* is less than, equal to, or greater than *s2*, respectively. `strncmp` makes the same comparison but looks at a maximum of *n* characters.

`strcpy` copies string *s2* to string *s1*, stopping after the null character has been copied. `strncpy` copies exactly *n* characters, truncating *s2* or adding null characters to *s1* if necessary. The result is not null-terminated if the length of *s2* is *n* or more. Each function returns *s1*.

`strlen` returns the number of characters in *s*, not including the terminating null character.

`strchr` (`strrchr`) returns a pointer to the first (last) occurrence of character *c* in string *s*, or a NULL pointer if *c* does not occur in the string. The null character terminating a string is considered to be part of the string.

`strpbrk` returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or a NULL pointer if no character from *s2* exists in *s1*.

`strspn` (`strcspn`) returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

`strtok` considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token, and writes a null character into *s1* immediately following the returned token. The function keeps track of its position in the string between separate calls, so that on subsequent calls (which must be made with a NULL pointer as the first argument) it works through the string *s1* immediately following that token. This can be continued until no tokens remain. The separator string *s2* may

be different from call to call. When no token remains in *s1*, a NULL pointer is returned.

NOTE

For user convenience, all these functions are declared in the optional <string.h> header file.

BUGS

strcmp use native character comparison. Thus the sign of the value returned when one of the characters has its high-order bit set is implementation-dependent.

All string movement is performed character by character starting at the left. Thus overlapping moves toward the left will work as expected, but overlapping moves to the right may yield surprises.

## NAME

strtod – convert string to double-precision number

## SYNOPSIS

```
double strtod(str, ptr)
char *str, **ptr;
```

## DESCRIPTION

strtod returns as a double-precision floating-point number the value represented by the character string pointed to by *str*. The string is scanned up to the first unrecognized character.

strtod recognizes an optional string of "white-space" characters (as defined by *isspace* in ctype(3C)), then an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer.

If the value of *ptr* is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no number can be formed, **ptr* is set to *str*, and zero is returned.

## SEE ALSO

atof(3C), ctype(3C), scanf(3S), strtol(3C).

## DIAGNOSTICS

If the correct value would cause overflow, ±HUGE is returned (according to the sign of the value), and errno is set to ERANGE.

If the correct value would cause underflow, zero is returned and errno is set to ERANGE.

## NAME

strtol, atol, atoi – convert string to integer

## SYNOPSIS

```
long strtol(str, ptr, base)
char *str, **ptr;
int base;

long atol(str)
char *str;

int atoi(str)
char *str;
```

## DESCRIPTION

`strtol` returns as a long integer the value represented by the character string pointed to by *str*. The string is scanned up to the first character inconsistent with the base. Leading white-space characters (blanks and tabs) are ignored.

If the value of *ptr* is not `(char **)NULL`, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no integer can be formed, zero is returned.

If *base* is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored; a leading `0x` or `0X` is ignored if *base* is 16.

If *base* is zero, the string itself determines the base. After an optional leading sign, a leading zero indicates octal conversion and a leading 0x or 0X indicates hexadecimal conversion; otherwise, decimal conversion is used.

Truncation from `long` to `int` can take place upon assignment or by an explicit cast.

`atol(str)` is equivalent to:

```
strtol(str, (char **)NULL, 10)
```

`atoi(str)` is equivalent to:

```
(int)strtol(str, (char **)NULL, 10)
```

## SEE ALSO

ctype(3C), scanf(3S), strtod(3C).

## BUGS

Overflow conditions are ignored.

## NAME

swab – swap bytes

## SYNOPSIS

```
void swab(from, to, nbytes)
char *from, *to;
int nbytes;
```

## DESCRIPTION

swab copies *nbytes* bytes referenced by *from* to the array referenced by *to*, exchanging adjacent even and odd bytes. It is useful for carrying binary data between PDP-11s and other machines. *nbytes* should be even and non-negative. If *nbytes* is odd and positive, swab uses *nbytes*–1 instead. If *nbytes* is negative, swab does nothing.

## NAME

system – issue a shell command from Fortran

## SYNOPSIS

```
character *N c
call system(c)
```

## DESCRIPTION

system causes its character argument to be given to sh(1) as input, as if the string had been typed at a terminal. The current process waits until the shell has completed.

## SEE ALSO

sh(1), exec(2), system(3S).

## NAME

system – issue a shell command

## SYNOPSIS

```
#include <stdio.h>

int system(string)
char *string;
```

## DESCRIPTION

system causes *string* to be given to sh(1) 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.

## RETURN VALUE

system forks to create a child process that in turn performs exec(2) on /bin/sh in order to execute *string*. If the fork or exec fails, system returns a negative value and sets errno.

## FILES

/bin/sh

## SEE ALSO

sh(1), exec(2).

## NAME

tan, dtan – Fortran tangent intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2
r2=tan(r1)
dp2=dtan(dp1)
dp2=ftan(dp1)
```

## DESCRIPTION

tan returns the real tangent of its real argument. dtan returns the double-precision tangent of its double-precision argument. The generic tan function becomes dtan as required with a double-precision argument.

## SEE ALSO

trig(3M).

## NAME

tanh, dtanh – Fortran hyperbolic tangent intrinsic function

## SYNOPSIS

```
real r1, r2
double precision dp1, dp2

r2=tanh(r1)

dp2=dtanh(dp1)
dp2=tanh(dp1)
```

## DESCRIPTION

tanh returns the real hyperbolic tangent of its real argument. dtanh returns the double-precision hyperbolic tangent of its double precision argument. The generic form tanh may be used to return a double-precision value given a double-precision argument.

## SEE ALSO

sinh(3M).

## NAME

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

## SYNOPSIS

```
char PC;
char *BC;
char *UP;
short ospeed;

int tgetent(bp, name)
char *bp, *name

int tgetnum(id
char *id;

int tgetflag(id)
char *id;

char *tgetstr(id, area)
char *id, **area;

char *tgoto(cm, destcol, destline)
char *cm;
int destcol;
int destline;

int tputs(cp, affcnt, outc)
char *cp;
int affcnt;
int(*outc)();
```

## DESCRIPTION

These functions extract and use capabilities from the terminal capability data base termcap(4). Note that these are low-level routines.

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 successful. It looks in the environment for a TERMCAP variable. If a variable is found whose value does not begin with a slash and the terminal type *name* is the same as the environment string TERM, the TERMCAP string is used instead of reading the termcap file. If the value does begin with a slash, the string is used as a pathname rather than /etc/termcap. This can speed up entry into programs that call tgetent. Bt can

also help debug new terminal descriptions or be used 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 gets the string value of capability *id*, placing it in the buffer at *area*, advancing the *area* pointer. It decodes the abbreviations for this field described in termcap(4), except for cursor addressing and padding information.

tgoto returns a cursor addressing string decoded from *cm* to go to column *destcol* in line *destline*. It uses the external variables UP (from the up capability) and BC (if bc is given rather than bs) if necessary to avoid placing \n, ^D or ^@ in the returned string. (Programs that call tgoto should be sure to turn off the XTABS bit(s), since tgoto may now output a tab. Note that programs using termcap should in general turn off XTABS anyway since some terminals use CONTROL-I for other functions, such as nondestructive space.) If a % sequence is given which is not understood, then 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 that is called with each character in turn. The external variable ospeed should contain the output speed of the terminal as encoded by stty (1). The external variable PC should contain a pad character to be used (from the pc capability) if a null (^@) is inappropriate.

**FILES**

/lib/libtermcap.a
/etc/termcap

**SEE ALSO**

ex(1), termcap(4).

## NAME

tmpfile – create a temporary file

## SYNOPSIS

```
#include <stdio.h>
FILE *tmpfile()
```

## DESCRIPTION

tmpfile creates a temporary file using a name generated by tmpnam(3S), and returns a corresponding FILE pointer. The file is automatically deleted when the process using it terminates. The file is opened for update ("w+"). tmpfile calls fopen and so returns any error code passed to it from

## RETURN VALUE

If the temporary file cannot be opened, an error message is printed using perror(3C), and a NULL pointer is returned. fopen.

## SEE ALSO

creat(2), unlink(2), fopen(3S), mktemp(3C), perror(3C), tmpnam(3S).

## NAME

tmpnam, tempnam – create a name for a temporary file

## SYNOPSIS

```
#include <stdio.h>

char *tmpnam(s)
char *s;

char *tempnam(dir, pfx)
char *dir, *pfx;
```

## DESCRIPTION

These functions generate filenames that can safely be used for a temporary file.

tmpnam always generates a filename using the pathname defined as p_tmpdir in the <stdio.h> header file. If *s* is NULL, *tmpnam* leaves its result in an internal static area and returns a pointer to that area. The next call to tmpnam will destroy the contents of the area. If *s* is not NULL, it is assumed to be the address of an array of at least l_tmpnam bytes, where l_tmpnam is a constant defined in <stdio.h>; tmpnam places its result in that array and returns *s*.

tempnam allows the user to control the choice of a directory. The argument *dir* points to the pathname of the directory in which the file is to be created. If *dir* is NULL or points to a string which is not a pathname for an appropriate directory, the pathname defined as p_tmpdir in the <stdio.h> header file is used. If that pathname is not accessible, /tmp will be used as a last resort. This entire sequence can be upstaged by providing an environment variable TMPDIR in the user's environment, whose value is a pathname for the desired temporary-file directory.

Many applications prefer that names of temporary files contain favorite initial letter sequences. Use the *pfx* argument for this. This argument may be NULL or point to a string of up to 5 characters to be used as the first few characters of the name of the temporary file.

tempnam uses malloc(3C) to get space for the constructed filename and returns a pointer to this area. Thus, any pointer value returned from tempnam may serve as an argument to *free* (see malloc(3C)). If tempnam cannot return the expected result for any reason (i.e., malloc failed or attempts to find an appropriate directory were unsuccessful), a NULL pointer will be returned.

## NOTES

These functions generate a different filename each time they are called.

Files created using these functions and either `fopen`(3S) or `creat`(2) are temporary only in the sense that they reside in a directory intended for temporary use and their names are unique. It is the user's responsibility to use `unlink`(2) to remove the file when its use is ended.

## SEE ALSO

`creat`(2), `unlink`(2), `fopen`(3S), `malloc`(3C), `mktemp`(3C), `tmpfile`(3S).

## BUGS

If called more than 17,576 times in a single process, `tmpnam` and `tempnam` will start recycling previously used names.

Between the time a filename is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using `tmpnam`, `tempnam`, or `mktemp`(3C) and the filenames are chosen carefully to avoid duplication by other means.

## NAME

sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

## 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 x, y;
```

## DESCRIPTION

sin, cos, and tan return, respectively, the sine, cosine, and tangent of their argument, which is in radians.

asin returns the arcsine of $x$, in the range $-\pi/2$ to $\pi/2$.

acos returns the arccosine of $x$, in the range 0 to $\pi$.

atan returns the arctangent of $x$, in the range $-\pi/2$ to $\pi/2$.

atan2 returns the arctangent of $y/x$, in the range $-\pi$ to $\pi$, using the signs of both arguments to determine the quadrant of the return value.

## RETURN VALUE

sin, cos, and tan lose accuracy when their argument is far from zero. For arguments sufficiently large, these functions return 0 when there would otherwise be a complete loss of significance. In this case a message indicating TLOSS error is printed on the standard error output. For less extreme arguments, a PLOSS error is generated but no message is printed. In both cases, errno is set to ERANGE.

If the magnitude of the argument of `asin` or `acos` is greater than one, or if both arguments of `atan2` are zero, zero is returned and `errno` is set to `EDOM`. In addition, a message indicating `DOMAIN` error is printed on the standard error output.

These error-handling procedures may be changed with the function `matherr`(3M).

**SEE ALSO**

`matherr`(3M).

## NAME

tsearch, tfind, tdelete, twalk – manage binary search trees

## SYNOPSIS

```
#include <search.h>

char *tsearch(key, rootp, compar)
char *key;
char **rootp;
int (*compar)();

char *tfind(key, rootp, compar);
char *key;
char **rootp;
int (*compar)();

char *tdelete(key, rootp, compar);
char *key;
char **rootp;
int (*compar)();

void twalk(root, action)
char *root;
void (*action)();
```

## DESCRIPTION

tsearch, tfind, tdelete, and twalk are routines for manipulating binary search trees. They are generalized from Knuth (6.2.2) Algorithms T and D. All comparisons are done with a user-supplied routine. This routine is called with two arguments, the pointers to the elements being compared. It returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

tsearch is used to build and access the tree. *key* is a pointer to a datum to be accessed or stored. If there is a datum in the tree equal to **key* (the value referenced by *key*), a pointer to this found datum is returned. Otherwise, **key* is inserted, and a pointer to it returned. Only pointers are copied, so the calling routine must store the data. *rootp* points to a variable that points to the root of the tree. A NULL value for the variable referenced by *rootp* denotes an empty tree; in this case, the variable will be set to point to the datum which will be at the root of the new tree.

Like `tsearch`, `tfind` will search for a datum in the tree, returning a pointer to it if found. However, if it is not found, `tfind` will return a NULL pointer. The arguments for `tfind` are the same as for `tsearch`.

`tdelete` deletes a node from a binary search tree. The arguments are the same as for `tsearch`. The variable pointed to by *rootp* will be changed if the deleted node was the root of the tree. `tdelete` returns a pointer to the parent of the deleted node, or a NULL pointer if the node is not found.

`twalk` traverses a binary search tree. *root* is the root of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) *action* is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument is the address of the node being visited. The second argument is a value from an enumeration data type

`typedef enum{`*preorder*`,`*postorder*`,`*endorder*`,`*leaf*`} VISIT;`

(defined in the `<search.h>` header file), depending on whether this is the first, second or third time that the node has been visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument is the level of the node in the tree, with the root being level zero.

The pointers to the key and the root of the tree should be of type pointer-to-element, and cast to type pointer-to-character. Similarly, although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

**EXAMPLE**

The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <search.h>
#include <stdio.h>

struct node {              /*pointers to these are
       char * string;        stored in the tree*/
       int length;
};
char string_space[10000]; /*space to store
                                        strings*/
```

```
struct node nodes[500];     /*nodes to store*/
struct node *root = NULL; /*this points to the
                                        root*/

main( )
{
     char *strptr = string_space;
     struct node *nodeptr = nodes;
     void print_node( ), twalk( );
     int i = 0, node_compare( );

     while(gets(strptr) != NULL && i++ < 500)   {
          /* set node */
          nodeptr->string = strptr;
          nodeptr->length = strlen(strptr);
          /* put node into the tree */
          (void) tsearch((char *)nodeptr, &root,
                 node_compare);
          /* adjust pointers, so we
              don't overwrite tree */
          strptr += nodeptr->length + 1;
          nodeptr++;
     }
     twalk(root, print_node);
}
/*
     This routine compares two nodes, based on an
     alphabetical ordering of the string field.
*/
int
node_compare(node1, node2)
struct node *node1, *node2;
{
     return strcmp(node1->string, node2->string);
}
/*
     This routine prints out a node, the
     first time twalk encounters it.
*/
void
print_node(node, order, level)
struct node **node;
VISIT order;
```

```
int  level;
{
     if  (order  ==  preorder  ||  order  ==  leaf)    {
          (void)printf("string  =  %20s,    length  =  %d\n",
               (*node)->string,  (*node)->length);
     }
}
```

**RETURN VALUE**

A NULL pointer is returned by `tsearch` if there is not enough space available to create a new node.

A NULL pointer is returned by `tsearch`, `tfind` and `tdelete` if *rootp* is NULL on entry.

If the datum is found, both `tsearch` and `tfind` return a pointer to it. If not, `tfind` returns NULL, and `tsearch` returns a pointer to the inserted item.

**SEE ALSO**

`bsearch`(3C), `hsearch`(3C), `lsearch`(3C).

**WARNINGS**

The *root* argument to `twalk` is one level of indirection less than the *rootp* arguments to `tsearch` and `tdelete`.

There are two nomenclatures used to refer to the order in which tree nodes are visited. `tsearch` uses preorder, postorder and endorder to respectively refer to visting a node before any of its children, after its left child and before its right, and after both its children. The alternate nomenclature uses preorder, inorder and postorder to refer to the same visits, which could result in some confusion over the meaning of postorder.

**BUGS**

If the calling function alters the pointer to the root, results are unpredictable.

## NAME

ttyname, isatty – find name of a terminal

## SYNOPSIS

```
char  *ttyname (fildes)
int  fildes;

int  isatty (fildes)
int  fildes;
```

## DESCRIPTION

ttyname returns a pointer to a string containing the null-terminated pathname of the terminal device associated with file descriptor *fildes*.

## RETURN VALUE

ttyname returns a NULL pointer if *fildes* does not describe a terminal device in directory /dev.

isatty returns 1 if *fildes* is associated with a terminal device; otherwise, it returns 0.

## FILES

/dev/*

## BUGS

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

**NAME**

ttyslot – find the slot in the utmp file of the current user

**SYNOPSIS**

```
int ttyslot()
```

**DESCRIPTION**

ttyslot returns the index of the current user's entry in the /etc/utmp file. This is accomplished by scanning the file /etc/inittab for the name of the terminal device associated with the standard input, the standard output, or the error output (0, 1, or 2).

**SEE ALSO**

getut(3C), ttyname(3C).

**FILES**

/etc/inittab
/etc/utmp

**RETURN VALUE**

A value of 0 is returned if an error is encountered while searching for the terminal name or if none of the above file descriptors is associated with a terminal device.

**NAME**

umount – unmount a file system

**SYNOPSIS**

```
int  umount(spec)
char  *spec;
```

**DESCRIPTION**

umount requests that a previously mounted file system contained on the block special device identified by *spec* be unmounted. *spec* is a pointer to a path name. After unmounting the file system, the directory upon which the file system was mounted reverts to its ordinary interpretation.

umount may be invoked only by the superuser.

**ERRORS**

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

| | |
|---|---|
| [EPERM] | The process's effective user ID is not superuser. |
| [ENXIO] | *spec* does not exist. |
| [ENOTBLK] | *spec* is not a block special device. |
| [EINVAL] | *spec* is not mounted. |
| [EBUSY] | A file on *spec* is busy. |
| [EFAULT] | *spec* points to an illegal address. |

**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.

**SEE ALSO**

fsmount(2), unmount(2), mount(3).

## NAME

ungetc – push character back into input stream

## SYNOPSIS

```
#include <stdio.h>

int ungetc(c, stream)
char c;
FILE *stream;
```

## DESCRIPTION

ungetc inserts the character *c* into the buffer associated with an input *stream*. That character, *c*, will be returned by the next getc call on that *stream*. ungetc returns *c* and leaves the file *stream* unchanged.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. In the case that *stream* is stdin, one character may be pushed back onto the buffer without a previous read statement.

If *c* equals EOF, ungetc does nothing to the buffer and returns EOF.

fseek(3S) erases all memory of inserted characters.

## RETURN VALUE

ungetc returns EOF if it can't insert the character.

## SEE ALSO

fseek(3S), getc(3S), setbuf(3S).

**NAME**

varargs – handle variable argument list

**SYNOPSIS**

```
#include <varargs.h>

va_alist

va_dcl

void va_start(pvar)
va_list pvar;

type va_arg(pvar, type)
va_list pvar;

void va_end(pvar)
va_list pvar;
```

**DESCRIPTION**

This set of macros allows portable procedures that accept variable argument lists to be written. Routines that have variable argument lists (such as printf(3S)) but do not use varargs are inherently nonportable, as different machines use different argument-passing conventions.

va_alist is used as the parameter list in a function header.

va_dcl is a declaration for va_alist. No semicolon should follow va_dcl.

va_list is a type defined for the variable used to traverse the list.

va_start is called to initialize *pvar* to the beginning of the list.

va_arg will return the next argument in the list referenced by *pvar*. *type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, as it cannot be determined at runtime.

va_end is used to clean up.

Multiple traversals, each bracketed by va_start ... va_end, are possible.

**EXAMPLE**

This example is a possible implementation of execl(2).

```
#include <varargs.h>
#define MAXARGS   100

/*execl is called by
```

```
   execl(file, arg1, arg2, ..., (char *)0);
*/
execl(va_alist)
va_dcl
{
    va_list ap;
    char *file;
     char *args[MAXARGS];
     int argno = 0;

     va_start(ap);
     file = va_arg(ap, char *);
     while ((args[argno++] = va_arg(ap, char *)) != (char *)0)
        ;
     va_end(ap);
       return execv(file, args);
}
```

**SEE ALSO**

exec(2), printf(3S).

**BUGS**

It is up to the calling routine to specify how many arguments there are, since it is not always possible to determine this from the stack frame. For example, `execl` is passed a zero pointer to signal the end of the list. `printf` can tell how many arguments are there by the format.

It is non-portable to specify a second argument of `char`, `short`, or `float` to `va_arg`, since arguments seen by the called function are not `char`, `short`, or `float`. C converts `char` and `short` arguments to `int` and converts `float` arguments to `double` before passing them to a function.

## NAME

vprintf, vfprintf, vsprintf – print formatted output of a varargs argument list

## SYNOPSIS

```
#include <stdio.h>
#include <varargs.h>

int vprintf(format, ap)
char *format;
va_list ap;

int vfprintf(stream, format, ap)
FILE *stream;
char *format;
va_list ap;

int vsprintf(s, format, ap)
char *s, *format;
va_list ap;
```

## DESCRIPTION

vprintf, vfprintf, and vsprintf are the same as printf, fprintf, and sprintf respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by varargs(5).

## EXAMPLE

The following demonstrates how vfprintf could be used to write an error routine.

```
#include <stdio.h>
#include <varargs.h>
        .
        .
        .
/*
  *    error should be called like
  *         error(function_name, format, arg1, arg2...);
  */
/*VARARGS0*/
void
error(va_alist)
/* Note that the function_name and format arguments
  * cannot be separately declared because of the
  */definition of varargs.
va_dcl
```

```
{
    va_list args;
    char *fmt;

    va_start(args);
    /* print out name of function causing error */
    (void)fprintf(stderr, "ERROR in %s: ",
                                va_arg(args, char *));
    fmt = va_arg(args, char *);
    /* print out remainder of message */
    (void)vfprintf(fmt, args);
    va_end(args);
    (void)abort( );
}
```

**SEE ALSO**

varargs(5).

## NAME

xdr – library routines for external data representation

## DESCRIPTION

These routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Data for remote procedure calls are transmitted using these routines.

## FUNCTIONS

| | |
|---|---|
| `xdr_array()` | translate arrays to/from external representation |
| `xdr_bool()` | translate Booleans to/from external representation |
| `xdr_bytes()` | translate counted byte strings to/from external representation |
| `xdr_destroy()` | destroy XDR stream and free associated memory |
| `xdr_double()` | translate double precision to/from external representation |
| `xdr_enum()` | translate enumerations to/from external representation |
| `xdr_float()` | translate floating point to/from external representation |
| `xdr_getpos()` | return current position in XDR stream |
| `xdr_inline()` | invoke the in-line routines associated with XDR stream |
| `xdr_int()` | translate integers to/from external representation |
| `xdr_long()` | translate long integers to/from external representation |
| `xdr_opaque()` | translate fixed-size opaque data to/from external representation |
| `xdr_reference()` | chase pointers within structures |
| `xdr_setpos()` | change current position in XDR stream |
| `xdr_short()` | translate short integers to/from external representation |
| `xdr_string()` | translate null-terminated strings to/from external representation |
| `xdr_u_int()` | translate unsigned integers to/from external representation |
| `xdr_u_long()` | translate unsigned long integers to/from external representation |

| | |
|---|---|
| `xdr_u_short()` | translate unsigned short integers to/from external representation |
| `xdr_union()` | translate discriminated unions to/from external representation |
| `xdr_void()` | always return one (1) |
| `xdr_wrapstring()` | package RPC routine for XDR routine, or vice-versa |
| `xdrmem_create()` | initialize an XDR stream |
| `xdrrec_create()` | initialize an XDR stream with record boundaries |
| `xdrrec_endofrecord()` | mark XDR record stream with an end-of-record |
| `xdrrec_eof()` | mark XDR record stream with an end-of-file |
| `xdrrec_skiprecord()` | skip remaining record in XDR record stream |
| `xdrstdio_create()` | initialize an XDR stream as standard I/O FILE stream |

**SEE ALSO**

*A/UX Network Applications Programming.*

**NAME**

ypclnt, yp_bind, yp_unbind, yp_get_default_domain, 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`(4) and `ypserv`(1M) 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 ERRORS 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

the client process can't bind a server for the proper domain, or

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 *n*th call to `yp_next()` (to get the *n*th +

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 `yp_first()` function is called on a particular map, and then the `yp_next()` function is repeatedly called on the same map at the same server until the call fails with a reason of `YPERR_NOMORE`, every entry in the data base will be seen exactly once. Further, if the same sequence of operations is performed on the same map at the same server, the entries will be seen in the same order.

Under conditions of heavy server load or server failure, it is possible for the domain to become unbound, then bound once again (perhaps to a different server) while a client is running. This can cause a break in one of the enumeration rules; specific entries may be seen twice by the client, or not at all. This approach protects the client from error messages that would otherwise be returned in the midst of the enumeration. The next paragraph describes a better solution to enumerating all entries in a map.

`yp_all` provides a way to transfer an entire map from server to client in a single request using TCP (rather than UDP as with other functions in this package). The entire transaction take place as a single RPC request and response. You can use `yp_all` just like any other YP procedure, identify the map in the normal manner, and supply the name of a function which will be called to process each key-value pair within the map. You return from the call to `yp_all` only when the transaction is completed (successfully or unsuccessfully), or your "`foreach`" function decides that it doesn't want to see any more key-value pairs.

The third parameter to `yp_all` is

```
struct ypall_callback *incallback {
int (*foreach)();
char *data;
};
```

The function `foreach` is called

```
foreach(instatus, inkey, inkeylen, inval, invallen, indata);
int instatus;
char *inkey;
```

```
int inkeylen;
char *inval;
int invalllen;
char *indata;
```

The *instatus* parameter will hold one of the return status values defined in <rpcsvc/yp_prot.h>; either YP_TRUE or an error code. (See ypprot_err, below, for a function which converts a YP protocol error code to a ypclnt layer error code.)

The key and value parameters are somewhat different than defined in the synopsis section above. First, the memory pointed to by the *inkey* and *inval* parameters is private to the yp_all function, and is overwritten with the arrival of each new key-value pair. It is the responsibility of the foreach function to do something useful with the contents of that memory, but it does not own the memory itself. Key and value objects presented to the foreach function look exactly as they do in the server's map; if they were not newline-terminated or null-terminated in the map, they won't be here either.

The *indata* parameter is the contents of the *incallback->data* element passed to yp_all. The *data* element of the callback structure may be used to share state information between the foreach function and the mainline code. Its use is optional, and no part of the YP client package inspects its contents; cast it to something useful, or ignore it as you see fit.

The foreach function is a Boolean. It should return zero to indicate that it wants to be called again for further received key-value pairs, or non-zero to stop the flow of key-value pairs. If foreach returns a non-zero value, it is not called again; the functional value of yp_all is then 0.

yp_order returns the order number for a map.

yp_master returns the machine name of the master YP server for a map.

yperr_string returns a pointer to an error message string that is null-terminated but contains no period or newline.

ypprot_err takes a YP protocol error code as input, and returns a ypclnt layer error code, which may be used in turn as an input to yperr_string.

ERRORS

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 */
```

**FILES**

/usr/include/rpcsvc/ypclnt.h
/usr/include/rpcsvc/yp_prot.h

**SEE ALSO**

ypserv(1M), ypfiles(4).

# Table of Contents

## Section 5: Miscellaneous Facilities


intro ..... introduction to miscellaneous facilities
ae ..... 3Com 10 Mb/s Ethernet interface
arp ..... Address Resolution Protocol
ascii ..... map of ASCII character set
environ ..... user environment
eqnchar ..... special character definitions for eqn and neqn
fcntl ..... file control options
font ..... description files for device-independent troff
greek ..... graphics for the extended TTY-37 type-box
inet ..... Internet protocol family
ip ..... Internet Protocol
lo ..... software loopback network interface
man ..... macros for formatting entries in this manual
math ..... math functions and constants
mm ..... macro package for formatting documents
mptx ..... the macro package for formatting a permuted index
ms ..... text formatting macros
mv ..... a troff macro package for typesetting viewgraphs and slides
nterm ..... terminal driving tables for nroff
prof ..... profile within a function
regexp ..... regular expression compile and match routines
stat ..... data returned by stat system call
tcp ..... Internet Transmission Control Protocol
term ..... conventional names for terminals
troff ..... description of output language
types ..... primitive system data types
udp ..... Internet User Datagram Protocol
values ..... machine-dependent values

## NAME

intro – introduction to miscellaneous facilities

## SYNOPSIS

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

## DESCRIPTION

This section describes miscellaneous facilities (such as macro packages, character set tables, etc.) and networking facilities (such as network protocols) available in the system.

Macro packages, character set tables and hardware support for network interfaces are found among the standard Section 5 entries. Entries describing a protocol family are marked "5F", while entries describing protocol use are marked "5P".

### NETWORKING FACILITIES

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(2N) 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(2N). 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 nonstandard facilities or extensions to a mechanism. For example, a protocol supporting the SOCK_STREAM abstraction may allow more than one byte of out-of-band data to be transmitted per out-of-band message.

A network interface is similar to a device interface. Network interfaces comprise the lowest layer of the networking subsystem,

interacting with the actual transport hardware. An interface may support one or more protocol families, and/or address formats.

## PROTOCOLS

The system currently supports only the DARPA Internet protocols fully. Raw socket interfaces are provided to IP protocol layer of the DARPA Internet, to the IMP link layer (1822), and to Xerox PUP-1 layer operating on top of 3Mb/s Ethernet interfaces. 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:

```
#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*/
```

*Note:* Only `AF_INET` is appropriate for this implementation.

## 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 superuser.

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

```
struct rtentry {
        u_long      rt_hash;
        struct      sockaddr rt_dst;
        struct      sockaddr rt_gateway;
        short       rt_flags;
        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)*/
```

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, each network interface autoconfigured installs a routing table entry when it wishes to have packets sent through it. Normally the interface specifies the route through it is 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 may be requested to address the packet to an entity different from the eventual recipient (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 nonzero), the resources associated with it will not be reclaimed until further references to it are released.

The routing code returns `EEXIST` if requested to duplicate an existing entry, `ESRCH` if requested to delete a nonexistent 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. This value is used to select among multiple routes to the same destination. When multiple routes to the same destination exist, the least used route is selected.

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(5), do not.

At boot time each interface which has underlying hardware support makes itself known to the system during the autoconfiguration process. Once the interface has acquired its address it is expected to install a routing table entry so that messages may be routed through it. Most interfaces require some part of their address specified with an SIOCSIFADDR ioctl before they will allow traffic to flow through them. On interfaces where the network-link layer address mapping is static, only the network number is taken from the ioctl; the remainder is found in a hardware specific manner. On interfaces which provide dynamic network-link layer address mapping facilities (e.g. 10Mb/s Ethernets), the entire address specified in the ioctl is used.

The following ioctl calls may be used to manipulate network interfaces. Unless specified otherwise, the request takes an ifrequest structure as its parameter. This structure has the form:

```
#define ifr_addr    ifr_ifru.ifru_addr    /* address */
#define ifr_dstaddr ifr_ifru.ifru_dstaddr /* other end of
                                             p-to-p link */
#define ifr_flags   ifr_ifru.ifru_flags   /* flags */

struct ifreq {
        char    ifr_name[16];     /* name of interface
                                     (e.g. "ec0") */
        union {
                struct    sockaddr ifru_addr;
                struct    sockaddr ifru_dstaddr;
                short     ifru_flags;
        } ifr_ifru;
};
```

SIOCSIFADDR — Set interface address. Following the address assignment, the "initialization" routine for the interface is called.

SIOCGIFADDR — Get interface address.

SIOCSIFDSTADDR — Set point to point address for interface.

SIOCGIFDSTADDR — Get point to point address for interface.

SIOCSIFFLAGS — Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are

notified.

SIOCGIFFLAGS — Get interface flags.

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.

```
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */
#define ifc_buf ifc_ifcu.ifcu_buf  /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req  /* array of structures
struct  ifconf {

        int     ifc_len;         /* size of associated
                                        buffer */
        union {
                caddr_t ifcu_buf;
                struct  ifreq *ifcu_req;
        } ifc_ifcu;
                                          returned */
};
```

**SEE ALSO**

routed(1M), socket(2N), ioctl(2).

## NAME

ae – 3Com 10 Mb/s Ethernet interface

## DESCRIPTION

The ae interface provides host access to an industry standard 10 Mb/s Ethernet.

The host's Internet address is specified at boot time with an SIOCSIFADDR ioctl. The hosts's Ethernet address is read from ROM on the Ethernet board using etheraddr(1M). The ae interface employs the address resolution protocol described in arp(5P) to dynamically map between Internet and Ethernet addresses on the local network.

## DIAGNOSTICS

ae%d: init failed. The NIC chip on the Ethernet board would not initalize.

ae%d transmitter frozen – resetting. A packet transmission failed to complete within a predetermined timeout period.

ae%d spurious interrupt. An interrupt was received but no operation was active.

ae%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

etheraddr(1M), inet(5F), intro(5), arp(5P).

## FILES

/etc/boot.d/ae6
/etc/master.d/ae6
/etc/startup.d/ae6

## NAME

arp – Address Resolution Protocol

## DESCRIPTION

arp is a protocol used to dynamically map between DARPA Internet and 10Mb/s Ethernet addresses on a local area network. It is used by all the 10Mb/s Ethernet interface drivers and is not directly accessible to users.

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 messages are transmitted. arp itself is not Internet or Ethernet specific; this implementation, however, is. 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.

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) and will, optionally, periodically probe a network looking for impostors.

## 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.

## NAME

ascii – map of ASCII character set

## SYNOPSIS

```
cat /usr/pub/ascii
```

## DESCRIPTION

ascii is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. It contains:

```
000 nul |001 soh |002 stx |003 etx |004 eot |005 enq |006 ack |007 bel
010 bs  |011 ht  |012 nl  |013 vt  |014 np  |015 cr  |016 so  |017 si
020 dle |021 dc1 |022 dc2 |023 dc3 |024 dc4 |025 nak |026 syn |027 etb
030 can |031 em  |032 sub |033 esc |034 fs  |035 gs  |036 rs  |037 us
040 sp  |041 !   |042 "   |043 #   |044 $   |045 %   |046 &   |047
050 (   |051 )   |052 *   |053 +   |054 ,   |055 -   |056 .   |057 /
060 0   |061 1   |062 2   |063 3   |064 4   |065 5   |066 6   |067 7
070 8   |071 9   |072 :   |073 ;   |074 <   |075 =   |076 >   |077 ?
100 @   |101 A   |102 B   |103 C   |104 D   |105 E   |106 F   |107 G
110 H   |111 I   |112 J   |113 K   |114 L   |115 M   |116 N   |117 O
120 P   |121 Q   |122 R   |123 S   |124 T   |125 U   |126 V   |127 W
130 X   |131 Y   |132 Z   |133 [   |134 \   |135 ]   |136 ^   |137 _
140     |141 a   |142 b   |143 c   |144 d   |145 e   |146 f   |147 g
150 h   |151 i   |152 j   |153 k   |154 l   |155 m   |156 n   |157 o
160 p   |161 q   |162 r   |163 s   |164 t   |165 u   |166 v   |167 w
170 x   |171 y   |172 z   |173 {   |174 |   |175 }   |176 ~   |177 del
```

```
 00 nul | 01 soh | 02 stx | 03 etx | 04 eot | 05 enq | 06 ack | 07 bel
 08 bs  | 09 ht  | 0a nl  | 0b vt  | 0c np  | 0d cr  | 0e so  | 0f si
 10 dle | 11 dc1 | 12 dc2 | 13 dc3 | 14 dc4 | 15 nak | 16 syn | 17 etb
 18 can | 19 em  | 1a sub | 1b esc | 1c fs  | 1d gs  | 1e rs  | 1f us
 20 sp  | 21 !   | 22 "   | 23 #   | 24 $   | 25 %   | 26 &   | 27
 28 (   | 29 )   | 2a *   | 2b +   | 2c ,   | 2d -   | 2e .   | 2f /
 30 0   | 31 1   | 32 2   | 33 3   | 34 4   | 35 5   | 36 6   | 37 7
 38 8   | 39 9   | 3a :   | 3b ;   | 3c <   | 3d =   | 3e >   | 3f ?
 40 @   | 41 A   | 42 B   | 43 C   | 44 D   | 45 E   | 46 F   | 47 G
 48 H   | 49 I   | 4a J   | 4b K   | 4c L   | 4d M   | 4e N   | 4f O
 50 P   | 51 Q   | 52 R   | 53 S   | 54 T   | 55 U   | 56 V   | 57 W
 58 X   | 59 Y   | 5a Z   | 5b [   | 5c \   | 5d ]   | 5e ^   | 5f _
 60     | 61 a   | 62 b   | 63 c   | 64 d   | 65 e   | 66 f   | 67 g
 68 h   | 69 i   | 6a j   | 6b k   | 6c l   | 6d m   | 6e n   | 6f o
 70 p   | 71 q   | 72 r   | 73 s   | 74 t   | 75 u   | 76 v   | 77 w
 78 x   | 79 y   | 7a z   | 7b {   | 7c |   | 7d }   | 7e ~   | 7f del
```

**FILES**

/usr/pub/ascii

## NAME

environ – user environment

## SYNOPSIS

```
extern char **environ;
```

## DESCRIPTION

An array of strings called the **environment** is made available by exec(2) when a process begins. By convention these strings have the form "*name=value*". The following names are used by various commands:

PATH — The sequence of directory prefixes that sh, time, nice(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by ":". login(1) sets: PATH=:/bin:/usr/bin.

HOME — A user's login directory, set by login(1) from the password file passwd(4).

TERM — The kind of terminal for which output is to be prepared. This information is used by commands, such as nroff, more, or vi, which may exploit special terminal capabilities. See /etc/termcap or (termcap(4)) for a list of terminal types.

SHELL — The file name of the user's login shell.

TERMCAP — The string describing the terminal in TERM, or the name of the termcap file, see termcap(4).

EXINIT — A startup list of commands read by ex(1), edit(1), and vi(1).

LOGNAME — The login name of the user.

TZ — Time zone information. The format is xxx*n*zzz where xxx is standard local time zone abbreviation, *n* is the difference is hours from GMT, and zzz is the abbreviation for the daylight-saving local time zone, if any; for example, EST5EDT.

Further names may be placed in the environment by the export command and "*name=value*" arguments in sh(1), or by the setenv command if you use csh(1). Arguments may also be placed in the environment at the point of an exec(2). It is unwise to conflict with certain sh(1) variables that are frequently exported by ".profile" files: MAIL,PS1,PS2,

**SEE ALSO**

csh(1), ex(1), ksh(1), login(1), sh(1), exec(2), system(3S), termcap(4), tty(7).

## NAME

eqnchar – special character definitions for eqn and neqn

## SYNOPSIS

eqn /usr/pub/eqnchar [*options*] [–] *files* | troff [*options*]

eqn /usr/pub/cateqnchar [*options*] [–] *files* | troff [*options*]

neqn /usr/pub/eqnchar [*options*] [–] *files* | troff [*options*]

eqn -Taps /usr/pub/apseqnchar [*options*] [–] *files* | troff [*options*]

## DESCRIPTION

/usr/pub/eqnchar contains troff(1) and nroff(1) character definitions for constructing characters that are not ordinarily available on a phototypesetter or printer. These definitions are primarily intended for use with eqn(1) and neqn(1).

For a complete list of input and output characters contained in /usr/pub/eqnchar, see the "eqn Reference" in *A/UX Text Processing Tools.*

/usr/pub/apseqnchar is a version of eqnchar tailored for the Autologic APS-5 phototypesetter. If you use apseqnchar, output will not look optimal on other phototypesetters. cateqnchar is more "device independent," and should look reasonable on any device supported by troff(1). You may link /usr/pub/eqnchar to /usr/pub/cateqnchar or to /usr/pub/apseqnchar. By default, /usr/pub/eqnchar is linked to /usr/pub/apseqnchar.

## FILES

/usr/pub/eqnchar
/usr/pub/apseqnchar
/usr/pub/cateqnchar

## SEE ALSO

eqn(1), neqn(1), troff(1).
"eqn Reference" in *A/UX Text Processing Tools.*

**NAME**

fcntl – file control options

**SYNOPSIS**

```
#include <fcntl.h>
```

**DESCRIPTION**

The fcntl(2) function provides for control over open files. The include file describes *requests* and *arguments* to fcntl and open(2).

```
/* Flag values accessible to open(2) and fcntl(2) */
/*  (The first three can only be set by open) */

#define O_RDONLY        0
#define O_WRONLY        1
#define O_RDWR  2
#define O_NDELAY        04      /* Non-blocking I/O */
#define O_APPEND        010     /* append (writes
                                   guaranteed at the end) */

/* Flag values accessible only to open(2) */
#define O_CREAT 00400   /* open with file create
                                (uses third open arg) */
#define O_TRUNC 01000   /* open with truncation */
#define O_EXCL  02000   /* exclusive open */

/* fcntl(2) requests */
#define F_DUPFD 0       /* Duplicate fildes */
#define F_GETFD 1       /* Get fildes flags */
#define F_SETFD 2       /* Set fildes flags */
#define F_GETFL 3       /* Get file flags */
#define F_SETFL 4       /* Set file flags */
#define F_GETLK 5       /* Get blocking file locks */
#define F_SETLK 6       /* Set or clear file locks
                                and fail on busy */
#define F_SETLKW 7      /* Set or clear file locks
                                and wait on busy */
#define F_GETOWN 8      /* Get owner */
#define F_SETOWN 9      /* Set owner */

/* file segment locking control structure */
struct flock {
short           l_type;
short           l_whence;
```

```
long            l_start;
long            l_len;  /* if 0 then until EOF */
int             l_pid;  /* returned with F_GETLK */

/* file segment locking types */
#define F_RDLCK 01      /* Read lock */
#define F_WRLCK 02      /* Write lock */
#define F_UNLCK 03      /* Remove locks */
```

**SEE ALSO**

fcntl(2), open(2).

## NAME

font – description files for device-independent troff

## SYNOPSIS

troff –T *tty-type* ...

## DESCRIPTION

For each phototypesetter that troff(1) supports and that is available on your system, there is a directory containing files describing the device and its fonts. This directory is named /usr/lib/font/dev*tty-type* where *tty-type* is the name of the phototypesetter. Currently the supported devices are aps for the Autologic APS–5 and i10 for the Imagen Imprint-10 laser printer.

For a particular phototypesetter, *tty-type*, the ASCII file *DESC* in the directory /usr/lib/font/dev*tty-type* describes its characteristics. A binary version of the file (described below) is found in /usr/lib/font/dev*tty-type*/DESC.out. Each line of this ASCII file starts with a word that identifies the characteristic, which is followed by appropriate specifiers. Blank lines and lines beginning with the # character are ignored.

The legal lines for DESC are:

| | |
|---|---|
| res *num* | resolution of device in basic increments per inch |
| hor *num* | smallest unit of horizontal motion |
| vert *num* | smallest unit of vertical motion |
| unitwidth *num* | pointsize in which widths are specified |
| sizescale *num* | scaling for fractional pointsizes |
| paperwidth *num* | width of paper in basic increments |
| paperlength *num* | length of paper in basic increments |
| biggestfont *num* | maximum size of a font |
| sizes *num num* . . . 0 | list of pointsizes available on typesetter |
| fonts *num name* ... | number of initial fonts followed by the names of the fonts. For example:<br>fonts 4 R I B S |

| | |
|---|---|
| `charset` | this always comes last in the file and is on a line by itself. Following it is the list of special character names for this device. Names are separated by a space or a newline. The list can be as long as necessary. Names not in this list are not allowed in the font description files. |

`res` is the basic resolution of the device in increments per inch. `hor` and `vert` describe the relationships between motions in the horizontal and vertical directions. If the device is capable of moving in single basic increments in both directions, both `hor` and `vert` would have values of 1. If the vertical motions only take place in multiples of two basic units while the horizontal motions take place in the basic increments, then `hor` would be 1, while `vert` would be 2. `unitwidth` is the pointsize in which all width tables in the font description files are given. `troff` automatically scales the widths from the `unitwidth` size to the pointsize it is working with. `sizescale` is not currently used and is 1. `paperwidth` is the width of the paper in basic increments. The APS-5 is 6120 increments wide. `paperlength` is the length of a sheet of paper in the basic increments. `biggestfont` is the maximum number of characters on a font.

For each font supported by the phototypesetter, there is also an ASCII file with the same name as the font (e.g., `R`, `I`, `CW`). The format for a font description file is:

| | |
|---|---|
| `name` ***name*** | name of the font, such as `R` or `CW` |
| `internalname` ***name*** | internal name of font |
| `special` | sets flag indicating that the font is special |
| `ligatures` ***name*** ... `0` | Sets flag indicating font has ligatures. The list of ligatures follows and is terminated by a zero. Accepted ligatures are: `ff fi fl ffi ffl`. |
| spacewidth ***num*** | specifies width of space if something other than default (1/3 of em) is desired. |

`charset` The charset must come at the end. Each line following the word `charset` describes one character in the font. Each line has one of two formats:
*name width kerning code*
*name* "

where *name* is either a single ASCII character or a special character name from the list found in `DESC`. The width is in basic increments. The kerning information is 1 if the character descends below the line, 2 if it rises above the letter "a," and 3 if it both rises and descends. The kerning information for special characters is not used and so may be 0. The code is the number sent to the typesetter to produce the character. The second format is used to indicate that the character has more than one name. The double quote indicates that this name has the same values as the preceding line. The kerning and code fields are not used if the width field is a double quote character. The total number of different characters in this list should not be greater than the value of `biggest-font` in the `DESC` file (see above).

`troff` and its postprocessors read this information from binary files produced from the ASCII files by a program distributed with `troff` called `makedev`. For those with a need to know, a description of the format of these files follows:

The file `DESC.out` starts with the *dev* structure, defined by `dev.h`:

```
/*
dev.h: characteristics of a typesetter
*/

struct dev {
short  filesize;    /* number of bytes in file, */
                    /* excluding dev part */
short  res;         /* basic resolution in goobies
                       per inch */
short  hor;         /* goobies horizontally */
short  vert;
short  unitwidth;   /* size at which widths
                       are given*/
```

```
short  nfonts;      /* number fonts physically
                       available */
short  nsizes;      /* number of pointsizes */
short  sizescale;   /* scaling for fractional
                       point sizes */
short  paperwidth;  /* max line length in units */
short  paperlength; /* max paper length in units */
short  nchtab;      /* number of funny names
                       in chtab */
short  lchname;     /* length of chname table */
short  biggestfont; /* max # of chars in a font */
short  spare2;      /* in case of expansion */
};
```

*filesize* is just the size of everything in `DESC.out` excluding the *dev* structure. `nfonts` is the number of different font positions available. `nsizes` is the number of different point sizes supported by this typesetter. `nchtab` is the number of special character names. `lchname` is the total number of characters, including nulls, needed to list all the special character names. At the end of the structure are two spares for later expansions.

Immediately following the *dev* structure are a number of tables. First is the *sizes* table, which contains `nsizes` + 1 shorts(a null at the end), describing the pointsizes of text available on this device. The second table is the `funny_char_index_table`. It contains indexes into the the table which follows it, the `funny_char_strings`. The indexes point to the beginning of each special character name which is stored in the `funny_char_strings` table. The `funny_char_strings` table is `lchname` characters long, while the `funny_char_index_table` is `nchtab` `shorts` long.

Following the *dev* structure will occur `nfonts` {*font*}*.out* {*font*}*.out* which are used to initialize the font positions. These {*font*}*.out* files, which also exist as separate files, begin with a *font* structure and then are followed by four character arrays:

```
struct Font {       /* characteristics of a font */
char   nwfont;      /* number of width entries */
char   specfont;    /* 1 == special font */
```

```
char   ligfont;       /* 1 == ligatures exist
                         on this font */
char   namefont[10]; /* name of this font,
                         e.g., R */
char   intname[10];   /* internal name of font,
                         in ASCII */
};
```

The *font* structure tells how many defined characters there are in the font, whether the font is a "special" font and if it contains ligatures. It also has the ASCII name of the font, which should match the name of the file it appears in, and the internal name of the font on the typesetting device (*intname*). The internal name is independent of the font position and name that `troff` knows about. For example, you might say "`mount R in position 4`", but when asking the typesetter to actually produce a character from the R font, the postprocessor which instructs the typesetter would use *intname*.

The first three character arrays are specific for the font and run in parallel. The first array, *widths*, contains the width of each character relative to *unitwidth*. *unitwidth* is defined in `DESC`. The second array, *kerning*, contains kerning information. If a character rises above the letter "a," 02 is set. If it descends below the line, 01 is set. The third array, *codes*, contains the code that is sent to the typesetter to produce the character.

The fourth array is defined by the device description in `DESC`. It is the `font_index_table`. This table contains indices into the *width*, *kerning*, and *code* tables for each character. The order that characters appear in these three tables is arbitrary and changes from one font to the next. In order for `troff` to be able to translate from ASCII and the special character names to these arbitrary tables, the `font_index_table` is created with an order which is constant for each device. The number of entries in this table is 96 plus the number of special character names for this device. The value 96 is 128 - 32, the number of printable characters in the ASCII alphabet. To determine whether a normal ASCII character exists, `troff` takes the ASCII value of the character, subtracts 32, and looks in the `font_index_table`. If it finds a 0, the character is not

defined in this font. If it finds anything else, that is the index into *widths*, *kerning*, and *codes* that describe that character.

To look up a special character name, for example `\(pl`, the mathematical plus sign, and determine whether it appears in a particular font or not, the following procedure is followed. A *counter* is set to 0 and an index to a special character name is picked out of the *counter'th* position in the `funny_char_index_table`. A string comparision is performed between *funny_char_strings* [*funny_char_index_table* [*counter*]] and the special character name, in our example `pl`, and if it matches, then `troff` refers to this character as ( 96 + *counter*). When it wants to determine whether a specific font supports this character, it looks in `font_index_table[(96+counter)]`, (see below), to see whether there is a 0, meaning the character does not appear in this font, or number, which is the index into the *widths*, *kerning*, and *codes* tables.

Notice that since a value of 0 in the *font_index_table* indicates that a character does not exist, the 0th element of the *width*, *kerning*, and *codes* arrays are not used. For this reason the 0th element of the *width* array can be used for a special purpose, defining the width of a space for a font. Normally a space is defined by `troff` to be 1/3 of the width of the \(em character, but if the 0th element of the *width* array is nonzero, then that value is used for the width of a space.

**SEE ALSO**

`troff`(1).

**FILES**

`/usr/lib/font/dev`*tty_type*`/DESC.out`
`/usr/lib/font/dev`*tty-type*`/`{*font*}`.out`

## NAME

greek – graphics for the extended TTY-37 type-box

## SYNOPSIS

cat /usr/pub/greek [ | greek –T*terminal* ]

## DESCRIPTION

greek gives the mapping from ASCII to the "shift-out" graphics in effect between SO and SI on TELETYPE Model 37 terminals equipped with a 128-character type-box. These are the default greek characters produced by nroff. The filters of greek(1) attempt to print them on various other terminals. The file contains:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| alpha | α | A | beta | β | B | gamma | γ | \ |
| GAMMA | Γ | G | delta | δ | D | DELTA | Δ | W |
| epsilon | ε | S | zeta | ζ | Q | eta | η | N |
| THETA | Θ | T | theta | θ | O | lambda | λ | L |
| LAMBDA | Λ | E | mu | μ | M | nu | ν | @ |
| xi | ξ | X | pi | π | J | PI | Π | P |
| rho | ρ | K | sigma | σ | Y | SIGMA | Σ | R |
| tau | τ | I | phi | ϕ | U | PHI | Φ | F |
| psi | ψ | V | PSI | Ψ | H | omega | ω | C |
| OMEGA | Ω | Z | nabla | ∇ | [ | not | ¬ | _ |
| partial | ∂ | ] | integral | ∫ | ^ | | | |

## FILES

/usr/pub/greek

## SEE ALSO

300(1), 4014(1), 450(1), greek(1), nroff(1), tc(1).
"Other Text Processing Tools" in *AIUX Text Processing Tools.*

## 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 address INADDR_ANY to effect "wildcard" matching on incoming messages.

## 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 not directly accessible.

## SEE ALSO

tcp(5P), udp(5P), ip(5P).

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, 0);
```

## DESCRIPTION

IP is the transport layer protocol used by the Internet protocol family. It may be accessed through a "raw socket" when developing new protocols, or special purpose applications. IP sockets are connectionless, and are normally used with the `sendto` and `recvfrom` calls, though the `connect`(2N) call may also be used to fix the destination for future packets (in which case the `read`(2) or `recv`(2N) and `write`(2) or `send`(2N) system calls may be used).

Outgoing packets automatically have an IP header prefixed to them (based on the destination address and the protocol number the socket is created with). Likewise, incoming packets have their IP header stripped before being sent to the user.

## ERRORS

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`(2N), `recv`(2N), `intro`(5), `inet`(5F).

BUGS

One should be able to send and receive `ip` options.

The protocol should be settable after socket creation.

## 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. By default, the loopback interface is accessible at address 127.0.0.1 (nonstandard); this address may be changed with the SIOCSIFADDR ioctl.

## 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(5), inet(5F).

## BUGS

It should handle all address and protocol families. An approved network address should be reserved for this interface.

## NAME

man – macros for formatting entries in this manual

## SYNOPSIS

nroff –man *files*

troff –man [–rs1] *files*

## DESCRIPTION

These nroff(1)/ troff(1) macros are used to lay out the format of the entries of this manual. The default page size is 8.5″×11″, with a 6.5″×10″ text area; the –rs1 flag option reduces these dimensions to 6″×9″ and 4.75″×8.375″, respectively; this option (which is *not* effective in nroff(1)) also reduces the default type size from 10-point to 9-point, and the vertical line spacing from 12-point to 10-point. The –rV2 flag option may be used to set certain parameters to values appropriate for certain Versatec printers: it sets the line length to 82 characters, the page length to 84 lines, and it inhibits underlining.

Any *text* argument below may be one to six "words". Double quotes (" ") may be used to include blanks in a "word". If *text* is empty, the special treatment is applied to the next line that contains text to be printed. For example, .I may be used to italicize a whole line, or .SM followed by .B to make small bold text. By default, hyphenation is turned off for nroff(1), but remains on for troff(1).

Type font and size are reset to default values before each paragraph and after processing font- and size-setting macros, e.g., .I, .RB, .SM. Tab stops are neither used nor set by any macro except .DT and .TH.

Default units for indents *in* are ens. When *in* is omitted, the previous indent is used. This remembered indent is set to its default value (7.2 ens in troff(1), 5 ens in nroffthis corresponds to 0.5″ in the default page size) by .TH, .P, and .RS, and restored by .RE.

| | |
|---|---|
| .TH *t s c n* | Set the title and entry heading; *t* is the title, *s* is the section number, *c* is extra commentary, e.g., "local," *n* is new manual name. Invokes .DT (see below). |
| .SH *text* | Place subhead *text*, e.g., SYNOPSIS, here. |
| .SS *text* | Place sub-subhead *text*, e.g., "Options", here. |
| .B *text* | Make *text* bold. |

| | |
|---|---|
| .I *text* | Make *text* italic. |
| .SM *text* | Make *text* 1 point smaller than default point size. |
| .RI *a b* | Concatenate roman *a* with italic *b*, and alternate these two fonts for up to six arguments. Similar macros alternate between any two of roman, italic, and bold: .IR .R B .BR .IB .BI |
| .P | Begin a paragraph with normal font, point size, and indent. .PP is a synonym for .P. |
| .HP *in* | Begin paragraph with hanging indent. |
| .TP *in* | Begin indented paragraph with hanging tag. The next line that contains text to be printed is taken as the tag. If the tag does not fit, it is printed on a separate line. |
| .IP *t in* | Same as .TP *in* with tag *t*; often used to get an indented paragraph without a tag. |
| .RS *in* | Increase relative indent (initially zero). Indent all output an extra *in* units from the current left margin. |
| .RE *k* | Return to the *k*th relative indent level (initially, *k*=1; *k*=0 is equivalent to *k*=1); if *k* is omitted, return to the most recent lower indent level. |
| .PM *m* | Produces proprietary markings; see mm(1). |
| .DT | Restore default tab settings (every 7.2 ens in troff(1), 5 ens in nroff(1)). |
| .PD *v* | Set the interparagraph distance to *v* vertical spaces. If *v* is omitted, set the interparagraph distance to the default value (0.4v in troff(1), 1v in nroff(1)). |

The following *strings* are defined:

| | |
|---|---|
| *R | ® in troff(1), (Reg.) in nroff. |
| *S | Change to default type size. |
| *(Tm | Trademark indicator. |

The following *number registers* are given default values by .TH:

| | |
|---|---|
| IN | Left margin indent relative to subheads (default is 7.2 ens in troff(1), 5 ens in nroff(1)). |
| LL | Line length including IN. |
| PD | Current interparagraph distance. |

## EXAMPLES

The man macros are provided to process manual pages already on-line at a given location and to enable users to make their own manual pages. The preceding section demonstrated the usage of the macros themselves; the following section provides examples of command lines typically used to process the completed files.

`man` macros are designed to run with either `nroff` or `troff`. The first command line will process a file using only macros and `nroff` requests:

```
nroff -Tlp -man file | lp
```

The file is piped to the local line printer, `lp`.

The next command line will process a file containing tables as well as macros and `nroff` requests:

```
tbl | nroff -Tlp -man file | col | lp
```

Notice that before it is sent to the line printer, the output is first filtered through `col`, to process the reverse line feeds used by `tbl`.

The final example is a command line that processes an unusual manual page, one using `pic`. If the manual pages created with `man` are intended for an on-line facility, components requiring `troff`, such as `pic` (or `grap`) should be avoided since the average installation of terminals will not be able to process typeset documents.

`pic file | tbl | troff -Taps -man |` *typesetter*

`grap` precedes `pic` because it is a preprocessor to `pic`; the reverse order, of course, will not format correctly. The file contains one or more tables, requiring `tbl`, but `col` is no longer necessary because typeset documents do not use reverse line feeds with which to make tables. The `-T` flag option for specifying the output device (terminal type) takes the argument `aps` here, readying the document for processing on the APS-5 phototypesetter.

## CAVEATS

Special macros, strings, and number registers exist, internal to `man`, in addition to those mentioned above. Except for names predefined by `troff (1)` and number registers `d`, `m`, and `y`, all such internal names are of the form *XA*, where *X* is one of `)`, `]`, and `}`, and *A* stands for any alphanumeric character.

The programs that prepare the table of contents and the permuted index for this manual assume the NAME section of each entry

consists of a single line of input that has the following format:

*name*[, *name*, *name* ...] \- explanatory text

increases The macro package the interword spaces (to eliminate ambiguity) in the SYNOPSIS section of each entry.

The macro package itself uses only the roman font (so that one can replace, for example, the bold font by the constant-width font (CW). Of course, if the input text of an entry contains requests for other fonts (e.g., `.I`, `.RB`, `\fI`), the corresponding fonts must be mounted.

**FILES**

```
/usr/lib/tmac/tmac.an
/usr/lib/macros/cmp.n.[dt].an
/usr/lib/macros/ucmp.n.an
```

**SEE ALSO**

eqn(1), man(1), tbl(1), tc(1), troff(1).
"Other Text Processing Tools" in *A/UX Text Processing Tools*.

**BUGS**

If the argument to `.TH` contains *any* blanks and is *not* enclosed by double quotes (`""`), there will be strange irregular dots on the output.

## NAME

math – math functions and constants

## SYNOPSIS

```
#include <math.h>
```

## DESCRIPTION

This file contains declarations of all the functions in the Math Library (described in Section 3M), as well as various functions in the C Library (Section 3C) that return floating-point values.

It defines the structure and constants used by the matherr(3M) error-handling mechanisms, including the following constant used as an error-return value:

| | |
|---|---|
| HUGE | The maximum value of a single-precision floating-point number. |

The following mathematical constants are defined for user convenience:

| | |
|---|---|
| M_E | The base of natural logarithms ($e$). |
| M_LOG2E | The base-2 logarithm of $e$. |
| M_LOG10E | The base-10 logarithm of $e$. |
| M_LN2 | The natural logarithm of 2. |
| M_LN10 | The natural logarithm of 10. |
| M_PI | The ratio of the circumference of a circle to its diameter. (There are also several fractions of its reciprocal and its square root.) |
| M_SQRT2 | The positive square root of 2. |
| M_SQRT1_2 | The positive square root of 1/2. |

For the definitions of various machine-dependent "constants," see the description of the <values.h> header file.

## FILES

/usr/include/math.h

## SEE ALSO

intro(3), matherr(3M), values(5).

## NAME

mm – macro package for formatting documents

## SYNOPSIS

mm [*options*] [*files*]

nroff –mm [*options*] [*files*]

nroff –cm [*options*] [*files*]

mmt [*options*] [*files*]

troff –mm [*options*] [*files*]

## DESCRIPTION

This package provides a formatting capability for a very wide variety of documents. The manner in which you type and edit a document is essentially independent of whether the document is to be eventually formatted at a terminal or is to be phototypeset.

Full details are provided in *A/UX Text Processing Tools.*

## FILES

| | |
|---|---|
| /usr/lib/tmac/tmac.m | pointer to the noncompacted version of the package |
| /usr/lib/macros/mm[nt] | noncompacted version of the package |

## SEE ALSO

mm(1), mmt(1), nroff(1), troff(1).
"mm Reference" in *A/UX Text Processing Tools.*

## NAME

mptx – the macro package for formatting a permuted index

## SYNOPSIS

nroff –mptx [*options*] [*files*]

troff –mptx [*options*] [*files*]

## DESCRIPTION

This package provides a definition for the .xx macro used for formatting a permuted index as produced by ptx(1). This package does not provide any other formatting capabilities such as headers and footers. If these or other capabilities are required, the mptx macro package may be used in conjuction with the mm macro package. In this case, the –mptx flag option must be invoked *after* the –mm call. For example:

```
nroff –mm –mptx file
```

or

```
mm –mptx file
```

## FILES

| | |
|---|---|
| /usr/lib/tmac/tmac.ptx | pointer to the macro package |
| /usr/lib/macros/ptx | macro package |

## SEE ALSO

mm(1), nroff(1), ptx(1), troff(1), mm(5).
"Other Text Processing Tools" in *A/UX Text Processing Tools*.

## NAME

ms – text formatting macros

## SYNOPSIS

`nroff -ms [ options ] file ...`

`troff -ms [ options ] file ...`

## DESCRIPTION

This package of `nroff` and `troff` macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through `col`(1). All external `ms` macros are defined below. Many `nroff` and `troff` requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

| | |
|---|---|
| `.bp` | begin new page |
| `.br` | break output line |
| `.sp` *n* | insert *n* spacing lines |
| `.ce` *n* | center next *n* lines |
| `.ls` *n* | line spacing: *n*=1 single, *n*=2 double space |
| `.na` | no alignment of right margin |

Font and point size changes with `\f` and `\s` are also allowed; for example, "`\fIword\fR`" will italicize *word*. Output of the `tbl`, `eqn`, and `refer`(1) preprocessors for equations, tables, and references is acceptable as input.

Full details are provided in *A/UX Text Processing Tools*.

## FILES

`/usr/lib/tmac/tmac.x`
`/usr/lib/ms/x.???`

## SEE ALSO

`eqn`(1), `refer`(1), `tbl`(1), `troff`(1).
"`ms` Reference" in *A/UX Text Processing Tools*.

## REQUESTS

| MACRO NAME | INITIAL VALUE | BREAK? RESET? | EXPLANATION |
|---|---|---|---|
| `.AB` *x* | – | y | begin abstract; if *x*=no don't label abstract |
| `.AE` | – | y | end abstract |
| `.AI` | – | y | author's institution |
| `.AU` | – | y | author's name |

| | | | |
|---|---|---|---|
| .B *x* | – | n | embolden *x*; if no *x*, switch to boldface |
| .B1 | – | y | begin text to be enclosed in a box |
| .B2 | – | y | end boxed text and print it |
| .BT | date | n | bottom title, printed at foot of page |
| .BX *x* | – | n | print word *x* in a box |
| .CM | if t | n | cut mark between pages |
| .CT | – | y,y | chapter title: page number moved to CF (TM only) |
| .DA *x* | if n | n | force date *x* at bottom of page; today if no *x* |
| .DE | – | y | end display (unfilled text) of any kind |
| .DS *x y* | I | y | begin display with keep; *x*=I,L,C,B; *y*=indent |
| .ID *y* | 8n,.5i | y | indented display with no keep; *y*=indent |
| .LD | – | y | left display with no keep |
| .CD | – | y | centered display with no keep |
| .BD | – | y | block display; center entire block |
| .EF *x* | – | n | even page footer *x* (3 part as for .tl) |
| .EH *x* | – | n | even page header *x* (3 part as for .tl) |
| .EN | – | y | end displayed equation produced by eqn |
| .EQ *x y* | – | y | break out equation; *x*=L,I,C; *y*=equation number |
| .FE | – | n | end footnote to be placed at bottom of page |
| .FP | – | n | numbered footnote paragraph; may be redefined |
| .FS *x* | – | n | start footnote; *x* is optional footnote label |
| .HD | undef | n | optional page header below header margin |
| .I *x* | – | n | italicize *x*; if no *x*, switch to italics |
| .IP *x y* | – | y,y | indented paragraph, with hanging tag *x*; *y*=indent |
| .IX *x y* | – | y | index words *x y* and so on (up to 5 levels) |
| .KE | – | n | end keep of any kind |
| .KF | – | n | begin floating keep; text fills remainder of page |
| .KS | – | y | begin keep; unit kept together on a single page |
| .LG | – | n | larger; increase point size by 2 |
| .LP | – | y,y | left (block) paragraph. |
| .MC *x* | – | y,y | multiple columns; *x*=column width |
| .ND *x* | if t | n | no date in page footer; *x* is date on cover |
| .NH *x y* | – | y,y | numbered header; *x*=level, *x*=0 resets, *x*=S sets to *y* |
| .NL | 10p | n | set point size back to normal |
| .OF *x* | – | n | odd page footer *x* (3 part as for .tl) |
| .OH *x* | – | n | odd page header *x* (3 part as for .tl) |
| .P1 | if TM | n | print header on 1st page |
| .PP | – | y,y | paragraph with first line indented |
| .PT | - % - | n | page title, printed at head of page |
| .PX *x* | – | y | print index (table of contents); *x*=no suppresses title |
| .QP | – | y,y | quote paragraph (indented and shorter) |
| .R | on | n | return to Roman font |
| .RE | 5n | y,y | retreat: end level of relative indentation |

| | | | |
|---|---|---|---|
| .RP *x* | – | n | released paper format; *x*=no stops title on 1st page |
| .RS | 5n | y,y | right shift: start level of relative indentation |
| .SH | – | y,y | section header, in boldface |
| .SM | – | n | smaller; decrease point size by 2 |
| .TA | 8n,5n | n | set tabs to 8n 16n ... (nroff) 5n 10n ... (troff) |
| .TC *x* | – | y | print table of contents at end; *x*=no suppresses title |
| .TE | – | y | end of table processed by tbl |
| .TH | – | y | end multi-page header of table |
| .TL | – | y | title in boldface and two points larger |
| .TM | off | n | thesis mode |
| .TS *x* | – | y,y | begin table; if *x*=H table has multi-page header |
| .UL *x* | – | n | underline *x*, (troff) |
| .UX *x* | – | n | UNIX; trademark message first time; *x* appended |
| .XA *x y* | – | y | another index entry; *x*=page or no for none; *y*=indent |
| .XE | – | y | end index entry (or series of .IX entries) |
| .XP | – | y,y | paragraph with first line exdented, others indented |
| .XS *x y* | – | y | begin index entry; *x*=page or no for none; *y*=indent |
| .1C | on | y,y | one column format, on a new page |
| .2C | – | y,y | begin two column format |
| .]- | – | n | beginning of refer reference |
| .[0 | – | n | end of unclassifiable type of reference |
| .[N | – | n | N= 1:journal-article, 2:book, 3:book-article, 4:report |

## REGISTERS

Formatting distances can be controlled in ms by means of built-in number registers. For example, this sets the line length to 6.5 inches:

```
.nr  LL  6.5i
```

Here is a table of number registers and their default values:

| | | | |
|---|---|---|---|
| PS | point size | paragraph | 10 |
| VS | vertical spacing | paragraph | 12 |
| LL | line length | paragraph | 6i |
| LT | title length | next page | same as LL |
| FL | footnote length | next .FS | 5.5i |
| PD | paragraph distance | paragraph | 1v (if n), .3v (if t) |
| DD | display distance | displays | 1v (if n), .5v (if t) |
| PI | paragraph indent | paragraph | 5n |
| QI | quote indent | next .QP | 5n |
| FI | footnote indent | next .FS | 2n |
| PO | page offset | next page | 0 (if n), ~1i (if t) |
| HM | header margin | next page | 1i |
| FM | footer margin | next page | 1i |

| | | | |
|---|---|---|---|
| FF | footnote format | next .FS | 0 (1, 2, 3 available) |

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting FF to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an .IP-like footnote paragraph.

Here is a list of string registers available in ms; they may be used anywhere in the text:

| NAME | STRING'S FUNCTION |
|---|---|
| *Q | quote (" in nroff, `` in troff) |
| *U | unquote (" in nroff, '' in troff) |
| *- | dash (-- in nroff, — in troff) |
| *(MO | month (month of the year) |
| *(DY | day (current date) |
| ** | automatically numbered footnote |
| *' | acute accent (before letter) |
| *` | grave accent (before letter) |
| *^ | circumflex (before letter) |
| *, | cedilla (before letter) |
| *: | umlaut (before letter) |
| *~ | tilde (before letter) |

## BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

## NAME

mv – a troff macro package for typesetting viewgraphs and slides

## SYNOPSIS

mvt [–a] [*options*] [*files*]

troff [–a] [–rX1] –mv [*options*] [*files*]

## DESCRIPTION

This package makes it easy to typeset viewgraphs and projection slides in a variety of sizes. A few macros (briefly described below) accomplish most of the formatting tasks needed in making transparencies. All of the facilities of troff(1), eqn(1), tbl(1), pic(1), and grap(1) are available for more difficult tasks.

The output can be previewed on most terminals, and, in particular, on the TEKTRONIX 4014. For this device, specify the –rX1 option (this option is automatically specified by the mvt command when that command is invoked with the –D4014 option). To preview output on other terminals, specify the –a option.

The available macros are:

.VS [*n*] [*i*] [*d*] Foil-start macro; foil size is to be 7″×7″; *n* is the foil number, *i* is the foil identification, *d* is the date; the foil-start macro resets all parameters (indent, point size, etc.) to initial default values, except for the values of *i* and *d* arguments inherited from a previous foil-start macro; it also invokes the .A macro (see below).

The naming convention for this and the following eight macros is that the first character of the name (V or S) distinguishes between viewgraphs and slides, respectively, while the second character indicates whether the foil is square (S), small wide (w), small high (h), big wide (W), or big high (H). Slides are "skinnier" than the corresponding viewgraphs: the ratio of the longer dimension to the shorter one is larger for slides than for viewgraphs. As a result, slide foils can be used for viewgraphs, but not vice versa; on the other hand, viewgraphs can accommodate a bit more text.

| | | |
|---|---|---|
| .Vw | [*n*] [*i*] [*d*] | Same as .VS, except that foil size is 7″ wide × 5″ high. |
| .Vh | [*n*] [*i*] [*d*] | Same as .VS, except that foil size is 5″×7″. |
| .VW | [*n*] [*i*] [*d*] | Same as .VS, except that foil size is 7″×5.4″. |
| .VH | [*n*] [*i*] [*d*] | Same as .VS, except that foil size is 7″×9″. |
| .Sw | [*n*] [*i*] [*d*] | Same as .VS, except that foil size is 7″×5″. |
| .Sh | [*n*] [*i*] [*d*] | Same as .VS, except that foil size is 5″×7″. |
| .SW | [*n*] [*i*] [*d*] | Same as .VS, except that foil size is 7″×5.4″. |
| .SH | [*n*] [*i*] [*d*] | Same as .VS, except that foil size is 7″×9″. |
| .A | [*x*] | Place text that follows at the first indentation level (left margin); the presence of *x* suppresses the ½ line spacing from the preceding text. |
| .B | [*m* [*s*] ] | Place text that follows at the second indentation level; text is preceded by a mark; *m* is the mark (default is a large bullet); *s* is the increment or decrement to the point size of the mark with respect to the *prevailing* point size (default is 0); if *s* is 100, it causes the point size of the mark to be the same as that of the *default* mark. |
| .C | [*m* [*s*] ] | Same as .B, but for the third indentation level; default mark is a dash. |
| .D | [*m* [*s*] ] | Same as .B, but for the fourth indentation level; default mark is a small bullet. |
| .T | *string* | *string* is printed as an oversize, centered title. |
| .I | [*in*] [*a* [*x*] ] | Change the current text indent (does not affect titles); *in* is the indent (in inches unless dimensioned, default is 0); if *in* is signed, it is an increment or decrement; the presence of *a* invokes the .A macro (see below) and passes *x* (if any) to it. |
| .S | [*p*] [*l*] | Set the point size and line length; *p* is the point size (default is "previous''); if *p* is 100, the point size reverts to the *initial* default for the current foil-start macro; if *p* is signed, it is an increment or decrement (default is 18 for .VS, .VH, and .SH, and 14 for the other foil-start macros); *l* is the line length (in inches unless dimensioned; default is 4.2″ for .Vh, 3.8″ for .Sh, 5″ for .SH, and 6″ for the other foil-start macros). |

`.DF` *n f* [*n f*...] Define font positions; may not appear within a foil's input text (i.e., it may only appear after all the input text for a foil, but before the next foil-start macro); *n* is the position of font *f*; up to four "*n f*" pairs may be specified; the first font named becomes the *prevailing* font; the initial setting is (`H` is a synonym for `G`):

```
DF 1 H 2 I 3 B 4 S
```

`.DV` [*a*] [*b*] [*c*] [*d*] Alter the vertical spacing between indentation levels; *a* is the spacing for `.A`, *b* is for `.B`, *c* is for `.C`, and *d* is for `.D`; all nonnull arguments must be dimensioned; null arguments leave the corresponding spacing unaffected; initial setting is:

```
DV 5v 5v 5v 0v
```

`.U` *str1* [*str2*] Underline *str1* and concatenate *str2* (if any) to it.

The last four macros in the above list do not cause a break; the `.I` macro causes a break only if it is invoked with more than one argument; all the other macros cause a break.

The macro package also recognizes the following uppercase synonyms for the corresponding lowercase `troff` requests:

```
AD BR CE FI HY NA NF NH NX SO SP
TA TI
```

The `Tm` string produces the trademark symbol.

The input tilde (`~`) character is translated into a blank on output.

See the user's manual cited below for further details.

**FILES**

```
/usr/lib/tmac/tmac.v
/usr/lib/macros/vmca
```

**SEE ALSO**

`eqn`(1), `mmt`(1), `tbl`(1), `troff`(1).
"Other Text Processing Tools" in *A/UX Text Processing Tools*.

**NAME**

nterm – terminal driving tables for nroff

**DESCRIPTION**

nroff(1) uses driving tables to customize its output for various types of output devices, such as printing terminals, special word processing terminals (such as Diablo, Qume, or NEC Spinwriter mechanisms), or special output filter programs. These driving tables are written as ASCII files, and are installed in /usr/lib/nterm/tab.name, where name is the name for that terminal type as given in term(5).

The first line of a driving table should contain the name of the terminal: simply a string with no embedded white space. "white space" means any combination of spaces, tabs and newlines. The next part of the driver table is structured as follows:

bset [*integer*] (not supported in all versions of nroff)
breset [*integer*] (not supported in all versions of nroff)
Hor [*integer*]
Vert [*integer*]
Newline [*integer*]
Char [*integer*]
Em [*integer*]
Halfline [*integer*]
Adj [*integer*]
twinit [*character-string*]
twrest [*character-string*]
twnl [*character-string*]
hlr [*character-string*]
hlf [*character-string*]
flr [*character-string*]
bdon [*character-string*]
bdoff [*character-string*]
iton [*character-string*]
itoff [*character-string*]
ploton [*character-string*]
plotoff [*character-string*]
up [*character-string*]
down [*character-string*]
right [*character-string*]
left [*character-string*]

The meanings of these fields are as follows:

| | |
|---|---|
| `bset` | bits to set in the `c_oflag` field of the `termio` structure before output. |
| `breset` | bits to reset in the `c_oflag` field of the `termio` structure before output. |
| `Hor` | horizontal resolution in units of 1/240 of an inch. |
| `Vert` | vertical resolution in units of 1/240 of an inch. |
| `Newline` | space moved by a newline (linefeed) character in units of 1/240 of an inch. |
| `Char` | quantum of character sizes, in units of 1/240 of an inch. (i.e., a character is a multiple of `Char` units wide) |
| `Em` | size of an em in units of 1/240 of an inch. |
| `Halfline` | space moved by a half-linefeed (or half-reverse-linefeed) character in units in 1/240 of an inch. |
| `Adj` | quantum of white space, in 1/240 of an inch. (i.e., white spaces are a multiple of Adj units wide)<br>*Note:* if this is less than the size of the space character, `nroff` will output fractional spaces using plot mode. Also, if the `-e` switch to `nroff` is used, `Adj` is set equal to `Hor` by `nroff`. |
| `twinit` | sequence of characters used to initialize the terminal in a mode suitable for `nroff`. |
| `twrest` | sequence of characters used to restore the terminal to normal mode. |
| `twnl` | sequence of characters used to move down one line. |
| `hlr` | sequence of characters used to move up one-half line. |
| `hlf` | sequence of characters used to move down one-half line. |
| `flr` | sequence of characters used to move up one line. |

`bdon` sequence of characters used to turn on hardware boldface mode, if any.

`bdoff` sequence of characters used to turn off hardware boldface mode, if any.

`iton` sequence of characters used to turn on hardware italics mode, if any.

`itoff` sequence of characters used to turn off hardware italics mode, if any.

`ploton` sequence of characters used to turn on hardware plot mode (for Diablo type mechanisms), if any.

`plotoff` sequence of characters used to turn off hardware plot mode (for Diablo type mechanisms), if any.

`up` sequence of characters used to move up one resolution unit (`Vert`) in plot mode, if any.

`down` sequence of characters used to move down one resolution unit (`Vert`) in plot mode, if any.

`right` sequence of characters used to move right one resolution unit (`Hor`) in plot mode, if any.

`left` sequence of characters used to move left one resolution unit (`Hor`) in plot mode, if any.

This part of the driving table is fixed format, and you cannot change the order of entries. You should put entries on separate lines, and these lines should contain exactly two fields (no comments allowed) separated by white space. For example,

```
Cbset   0
breset  0
Hor     24
```

and so on.

Follow this first part of the driving table with a line containing the word "`charset`," and then specify a table of special characters that you want to include. That is, specify all the non-ASCII characters that `nroff`(1) knows by two character names, such as -. If `nroff` does not find the word "`charset`" where it expects to, it will abort with an error message.

Each definition in the part after "charset" occupies one line, and has the following format:

*chname width output*

where "*chname*" is the (two letter) name of the special character, "*width*" is its width in ems, and "*output*" is the string of characters and escape sequences to send to the terminal to produce the special character.

If any field in the "charset" part of the driving table does not pertain to the output device, you may give that particular sequence as a null string, or leave out the entry. Special characters that do not have a definition in this file are ignored on output by nroff(1).

You may put the "charset" definitions in any order, so it is possible to speed up nroff by putting the most used characters first. For example,

```
charset
em 1 -
hy 1 -
\- 1 -
bu 1 +
```

and so on.

The best way to create a terminal table for a new device is to take an existing terminal table and edit it to suit your needs. Once you create such a file, put it in the directory /usr/lib/nterm, and give it the name tab.*xyz* where *xyz* is the name of the terminal and the name that you pass nroff via the -T flag option (for example, nroff -Txyz).

**FILES**

/usr/lib/nterm/tab.name

**SEE ALSO**

nroff(1).

## NAME

prof – profile within a function

## SYNOPSIS

```
#define MARK
#include <prof.h>

void MARK (name)
```

## DESCRIPTION

MARK will introduce a mark called *name* that will be treated the same as a function entry point. Execution of the mark will add to a counter for that mark, and program-counter time spent will be accounted to the immediately preceding mark or to the function if there are no preceding marks within the active function.

*name* may be any combination of up to six letters, numbers or underscores. Each *name* in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol MARK must be defined before the header file <prof.h> is included. This may be defined by a preprocessor directive as in the synopsis, or by a command line argument, i.e:

```
cc -p -DMARK foo.c
```

If MARK is not defined, the MARK (*name*) statements may be left in the source files containing them and will be ignored.

## EXAMPLE

In this example, marks can be used to determine how much time is spent in each loop. Unless this example is compiled with MARK defined on the command line, the marks are ignored.

```
#include <prof.h>

foo( )
{
        int  i,  j;

        .
        .
        .
        MARK(loop1);
        for  (i  =  0;  i  <  2000;  i++)  {
              ...
        }
        MARK(loop2);
```

```
        for (j = 0; j < 2000; j++) {
                ...
        }
}
```

**SEE ALSO**

prof(1), profil(2), monitor(3C).

## NAME

regexp – regular expression compile and match routines

## SYNOPSIS

```
#define  INIT <declarations>
#define  GETC() <getc code>
#define  PEEKC() <peekc code>
#define  UNGETC(c) <ungetc code>
#define  RETURN (pointer) <return code>
#define  ERROR (val) <error code>

#include  <regexp.h>

char  *compile (instring,  expbuf,  endbuf,  eof)
char  *instring,  *expbuf,  *endbuf;
int eof;

int  step  (string,  exbuf)
char  *string,  *exbuf;

extern  char  *loc1,  *loc2,  *locs;

extern  int  circf,  sed,  nbra;
```

## DESCRIPTION

This page describes general-purpose regular expression matching routines in the form of ed(1), defined in /usr/include/regexp.h. Programs such as ed(1), sed(1), grep(1), bs(1), expr(1), etc., which perform regular expression matching use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is unpleasantly complex. Programs that include this file must have the following five macros declared before the "#include <regexp.h>" statement. These macros are used by the compile routine.

| | |
|---|---|
| GETC() | Return the value of the next character in the regular expression pattern. Successive calls to GETC() should return successive characters of the regular expression. |
| PEEKC() | Return the next character in the regular expression. Successive calls to PEEKC() should return the same character (which should also be the next character returned by GETC()). |

UNGETC(*c*) Cause the argument *c* to be returned by the next call to GETC() (and PEEKC()). No more that one character of pushback is ever needed and this character is guaranteed to be the last character read by GETC(). The value of the macro UNGETC(*c*) is always ignored.

RETURN(*pointer*) This macro is used on normal exit of the compile routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.

ERROR(*val*) This is the abnormal return from the compile routine. The argument *val* is an error number (see table below for meanings). This call should never return.

| ERROR | MEANING |
|---|---|
| 11 | Range endpoint too large. |
| 16 | Bad number. |
| 25 | "\digit" out of range. |
| 36 | Illegal or missing delimiter. |
| 41 | No remembered search string. |
| 42 | \( \) imbalance. |
| 43 | Too many \(. |
| 44 | More than 2 numbers given in \{ \}. |
| 45 | } expected after \. |
| 46 | First number exceeds second in \{ \}. |
| 49 | [ ] imbalance. |
| 50 | Regular expression overflow. |

The syntax of the compile routine is as follows:

compile(*instring*, *expbuf*, *endbuf*, *eof*)

The first parameter *instring* is never used explicitly by the compile routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((char *) 0) for this parameter.

The next parameter *expbuf* is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (*endbuf–expbuf*) bytes, a call to `ERROR(50)` is made.

The parameter *eof* is the character which marks the end of the regular expression. For example, in `ed`(1), this character is usually a `/`.

Each program that includes this file must have a `#define` statement for `INIT`. This definition will be placed right after the declaration for the function `compile` and the opening curly brace ({). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point the beginning of the regular expression so that this register variable can be used in the declarations for `GETC()`, `PEEKC()` and `UNGETC()`. Otherwise it can be used to declare external variables that might be used by `GETC()`, `PEEKC()` and `UNGETC()`. See the example below of the declarations taken from `grep`(1).

There are other functions in this file which perform actual regular expression matching, one of which is the function `step`. The call to `step` is as follows:

`step(`*string*`,` *expbuf*`)`

The first parameter to `step` is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter *expbuf* is the compiled regular expression which was obtained by a call of the function `compile`.

The function `step` returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to `step`. The variable set in `step` is `loc1`. This is a pointer to the first character that matched the regular expression. The variable `loc2`, which is set by the function `advance`, points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, `loc1` will point to the first character of *string* and `loc2` will point to the null at the end of *string*.

`step` uses the external variable `circf` which is set by `compile` if the regular expression begins with `^`. If this is set then `step` will try to match the regular expression to the beginning of

the string only. If more than one regular expression is to be compiled before the first is executed the value of `circf` should be saved for each compiled expression and `circf` should be set to that saved value before each call to `step`.

The function `advance` is called from `step` with the same arguments as `step`. The purpose of `step` is to step through the *string* argument and call `advance` until `advance` returns non-zero indicating a match or until the end of *string* is reached. If one wants to constrain *string* to the beginning of the line in all cases, `step` need not be called; simply call `advance`.

When `advance` encounters a * or \{ \} sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, `advance` will back up along the string until it finds a match or reaches the point in the string that initially matched the * or \{ \}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer `locs` is equal to the point in the string at sometime during the backing up process, `advance` will break out of the loop that backs up and will return zero. This is used by `ed`(1) and `sed`(1) for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like `s/y*//g` do not loop forever.

The additional external variables `sed` and `nbra` are used for special purposes.

## EXAMPLES

The following is an example of how the regular expression macros and calls look from `grep`(1):

```
#define INIT        register char *sp=instring;
#define GETC()      (*sp++)
#define PEEKC()     (*sp)
#define UNGETC(c)   (--sp)
#define RETURN(c)   return;
#define ERROR(c)    regerr()

#include <regexp.h>
...
    (void) compile(*argv,expbuf,&expbuf[ESIZE],'\0');
...
    if (step(linebuf,expbuf))
    succeed();
```

**FILES**

`/usr/include/regexp.h`

**SEE ALSO**

bs(1), ed(1), expr(1), grep(1), sed(1).

**BUGS**

The handling of `circf` is kludgy.

The actual code is probably easier to understand than this manual page.

## NAME

stat – data returned by stat system call

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
```

## DESCRIPTION

The system calls stat and fstat return data whose structure is defined by this include file. The encoding of the field st_mode is defined in this file also.

```
/*
 * Structure of the result of stat
 */

struct  stat
{
        dev_t   st_dev;
        ino_t   st_ino;
        ushort  st_mode;
        short   st_nlink;
        short   st_uid;
        short   st_gid;
        dev_t   st_rdev;
        off_t   st_size;
        time_t  st_atime;
        int     st_spare1;
        time_t  st_mtime;
        int     st_spare2;
        time_t  st_ctime;
        int     st_spare3;
        long    st_blksize;
        long    st_blocks;
        long    st_spare4[2];
};

#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_IFIFO 0010000    /* FIFO */
```

```
#define S_IFLNK 0120000    /* symbolic link */
#define S_IFSOC 0140000    /* socket */
#define S_ISUID 04000      /* set user ID on execution */
#define S_ISGID 02000      /* set group ID on execution */
#define S_ISVTX 01000      /* save swapped text even
                              after use */
#define S_IREAD 00400      /* read permission, owner */
#define S_IWRIT 00200      /* write permission, owner */
#define S_IEXEC 00100      /* execute/search permission,
                              owner */
```

**FILES**

/usr/include/sys/types.h
/usr/include/sys/stat.h

**SEE ALSO**

stat(2), types(5).

## 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(2N) system call must be used after binding the socket with the bind(2N) system call. Only passive sockets may use the accept(2N) call to accept incoming connections. Only active sockets may use the connect(2N) 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.

## ERRORS

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; |
| [EADDRNOTAVAIL] | when an attempt is made to create a socket with a network address for which no network interface exists. |

**SEE ALSO**

intro(5), inet(5F).

**BUGS**

It should be possible to send and receive TCP options. The system always tries to negotiates the maximum TCP segment size to be 1024 bytes. This can result in poor performance if an intervening network performs excessive fragmentation.

NAME

term – conventional names for terminals

DESCRIPTION

These names are used by certain commands (e.g., nroff(1), mm(1), man(1), tabs(1)) and are maintained as part of the shell environment (see sh(1), profile(4), and environ(5)) in the variable $TERM:

| | |
|---|---|
| 1520 | Datamedia 1520 |
| 1620 | Diablo 1620 and others using the HyType II printer |
| 1620–12 | same, in 12-pitch mode |
| 2621 | Hewlett-Packard HP2621 series |
| 2631 | Hewlett-Packard 2631 line printer |
| 2631–c | Hewlett-Packard 2631 line printer - compressed mode |
| 2631–e | Hewlett-Packard 2631 line printer - expanded mode |
| 2640 | Hewlett-Packard HP2640 series |
| 2645 | Hewlett-Packard HP264n series (other than the 2640 series) |
| 300 | DASI/DTC/GSI 300 and others using the HyType I printer |
| 300–12 | same, in 12-pitch mode |
| 300s | DASI/DTC/GSI 300s |
| 382 | DTC 382 |
| 300s–12 | same, in 12-pitch mode |
| 3045 | Datamedia 3045 |
| 33 | TELETYPE Terminal Model 33 KSR |
| 37 | TELETYPE Terminal Model 37 KSR |
| 40–2 | TELETYPE Terminal Model 40/2 |
| 40–4 | TELETYPE Terminal Model 40/4 |
| 4540 | TELETYPE Terminal Model 4540 |
| 3270 | IBM Model 3270 |
| 4000a | Trendata 4000a |
| 4014 | Tektronix 4014 |
| 43 | TELETYPE Model 43 KSR |
| 450 | DASI 450 (same as Diablo 1620) |
| 450–12 | same, in 12-pitch mode |
| 735 | Texas Instruments TI735 and TI725 |
| 745 | Texas Instruments TI745 |
| dumb | generic name for terminals that lack reverse linefeed and other special escape sequences |
| sync | generic name for synchronous TELETYPE 4540-compatible terminals |
| hp | Hewlett-Packard (same as 2645) |
| lp | generic name for a line printer |
| tn1200 | General Electric TermiNet 1200 |

term(5)

tn300 General Electric TermiNet 300

Up to 8 characters, chosen from [–a–z0–9], make up a basic terminal name. Terminal submodels and operational modes are distinguished by suffixes beginning with a –. Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name.

Commands whose behavior depends on the type of terminal should accept arguments of the form –T*term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable `$TERM`, which, in turn, should contain `term`.

See `/etc/termcap` on your system for a complete list.

SEE ALSO

mm(1), `nroff`(1), `sh`(1), `stty`(1), `tabs`(1), `tplot`(1G), `profile`(4), `environ`(5).

BUGS

This is a small candle trying to illuminate a large, dark problem. Programs that ought to adhere to this nomenclature do so somewhat fitfully.

## NAME

troff – description of output language

## DESCRIPTION

The device-independent troff outputs a pure ASCII description of a typeset document. The description specifies the typesetting device, the fonts, and the point sizes of characters to be used as well as the position of each character on the page. A list of all the legal commands follows. Most numbers are denoted as *n* and are ASCII strings. Strings inside of brackets ([]) are optional. troff may produce them, but they are not required for the specification of the language. The character \n has the standard meaning of "newline" character. Between commands, white space has no meaning. White space characters are spaces and newlines.

| | |
|---|---|
| s*n* | The point size of the characters to be generated. |
| f*n* | The font mounted in the specified position is to be used. The number ranges from 0 to the highest font presently mounted. 0 is a special position, invoked by troff, but not directly accessible to the troff user. Normally fonts are mounted starting at position 1. |
| c*x* | Generate the character *x* at the current location on the page; *x* is a single ASCII character. |
| C*xyz* | Generate the special character *xyz*. The name of the character is delimited by white space. The name will be one of the special characters legal for the typesetting device as specified by the device specification found in the file DESC. This file resides in a directory specific for the typesetting device. (See font(5) and /usr/lib/font/dev*.) |
| H*n* | Change the horizonal position on the page to the number specified. The number is in basic units of motions as specified by *DESC*. This is an absolute "goto". |

| | |
|---|---|
| h*n* | Add the number specified to the current horizontal position. This is a relative "goto". |
| V*n* | Change the vertical position on the page to the number specified (down is positive). |
| v*n* | Add the number specified to the current vertical position. |
| *nnx* | This is a two-digit number followed by an ASCII character. The meaning is a combination of h*n* followed by c*x*. The two digits *nn* are added to the current horizontal position and then the ASCII character, *x*, is produced. This is the most common form of character specification. |
| n*b a* | This command indicates that the end of a line has been reached. No action is required, though by convention the horizontal position is set to 0. troff will specify a resetting of the *x*,*y* coordinates on the page before requesting that more characters be printed. The first number, *b*, is the amount of space before the line and the second number, *a*, the amount of space after the line. The second number is delimited by white space. |
| w | A w appears between words of the input document. No action is required. It is included so that one device can be emulated more easily on another device. |
| p*n* | Begin a new page. The new page number is included in this command. The vertical position on the page should be set to 0. |
| # .... \n | A line beginning with a pound sign is a comment. |
| Dl *x y*\n | Draw a line from the current location to *x*,*y*. |

`Dc` *d*`\n` Draw a circle of diameter *d* with the leftmost edge being at the current location (x, y). The current location after drawing the circle will be x+*d*,y, the rightmost edge of the circle.

`De` *dx dy*`\n` Draw an ellipse with the specified axes. *dx* is the axis in the x direction and *dy* is the axis in the y direction. The leftmost edge of the ellipse will be at the current location. After drawing the ellipse the current location will be x+*dx*,y.

`Da` *x y u v* Draw a counterclockwise arc from the current location to *x*+*u*,*y*+*v* using a circle of whose center is *x*,*y* from the current location. The current location after drawing the arc will be at its end.

`D~` *x y x y*...`\n` Draw a spline curve (wiggly line) between each of the *x*,*y* coordinate pairs starting at the current location. The final location will be the final *x*,*y* pair of the list.

`x i[nit]\n` Initialize the typesetting device. The actions required are dependent on the device. An `init` command will always occur before any output generation is attempted.

`x T` *device*`\n` The name of the typesetter is *device*. This is the same as the argument to the `-T` option. The information about the typesetter will be found in the directory `/usr/lib/font/dev`*{device}* .

`x r[es]` *n h v*`\n` The resolution of the typesetting device in increments per inch is *n*. Motion in the horizontal direction can take place in units of *h* basic increments. Motion in the vertical direction can take place in units of *v* basic increments. For example, the APS-5 typesetter has a basic resolution of 723 increments per inch and can move in either direction in 723rds of an inch. Its specification is:

`x res 723 1 1`

| | |
|---|---|
| `x p[ause]\n` | Pause. Cause the current page to finish but do not relinquish the typesetter. |
| `x s[top]\n` | Stop. Cause the current page to finish and then relinquish the typesetter. Perform any shutdown and bookkeeping procedures required. |
| `x t[railer]\n` | Generate a trailer. On some devices no operation is performed. |
| `x f[ont]` ***n name*** `\n` | Load the font ***name*** into position ***n***. |
| `x H[eight]` ***n*** `\n` | Set the character height to ***n*** points. This causes the letters to be elongated or shortened. It does not affect the width of a letter. |
| `x S[lant]` ***n*** `\n` | Set the slant to ***n*** degrees. Only some typesetters can do this and not all angles are supported. |

**SEE ALSO**

`troff`(1).

"`nroff/troff` Reference" and "Introduction to `troff` and `mm`" in *A/UX Text Processing Tools*.

## NAME

types – primitive system data types

## SYNOPSIS

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

## DESCRIPTION

The data types defined in the include file are used in A/UX System code; some data of these types are accessible to user code:

```
typedef    struct { int r[1]; } *physadr;
typedef    long daddr_t;
typedef    char *caddr_t;
typedef    unsigned int uint;
typedef    unsigned short ushort;
typedef    ushort ino_t;
typedef    short cnt_t;
typedef    long time_t;
typedef    int label_t[10];
typedef    shor tdev_t;
typedef    long off_t;
typedef    long paddr_t;
typedef    long key_t;
```

The form daddr_t is used for disk addresses except in an inode on disk, see fs(4). 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(4).

## 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`(2N) call may also be used to fix the destination for future packets (in which case the `recv`(2N) or `send`(2N) 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.

## ERRORS

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.

**SEE ALSO**

send(2N), recv(2N), intro(5), inet(5F).

## NAME

values – machine-dependent values

## SYNOPSIS

```
#include <values.h>
```

## DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

BITS (*type*)
: The number of bits in a specified type (e.g., int).

HIBITS
: The value of a short integer with only the high-order bit set (in most implementations, 0x8000).

HIBITL
: The value of a long integer with only the high-order bit set (in most implementations, 0x80000000).

HIBITI
: The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).

MAXSHORT
: The maximum value of a signed short integer (in most implementations, 0x7FFF ≡ 32767).

MAXLONG
: The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF ≡ 2147483647).

MAXINT
: The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).

MAXFLOAT, LN_MAXFLOAT
: The maximum value of a single-precision floating-point number, and its natural logarithm.

MAXDOUBLE, LN_MAXDOUBLE
: The maximum value of a double-precision floating-point number, and its natural logarithm.

MINFLOAT, LN_MINFLOAT
: The minimum positive value of a single-precision floating-

point number, and its natural logarithm.

MINDOUBLE, LN_MINDOUBLE
The minimum positive value of a double-precision floating-point number, and its natural logarithm.

FSIGNIF
The number of significant bits in the mantissa of a single-precision floating-point number.

DSIGNIF
The number of significant bits in the mantissa of a double-precision floating-point number.

**FILES**

/usr/include/values.h

**SEE ALSO**

intro(3), math(5).

# Appendix A
# Permuted Index


| | | |
|---|---|---|
| ae: 3Com | 10 Mb/s Ethernet interface. . | ae(5) |
| functions of DASI 300 and 300s/ | 300, 300s: handle special . . | 300(1) |
| handle special functions of DASI | 300 and 300s terminals. /300s: | 300(1) |
| DASI 300 and 300s/ 300, | 300s: handle special functions of | 300(1) |
| special functions of DASI 300 and | 300s terminals. /300s: handle . | 300(1) |
| l3tol, ltol3: convert between | 3-byte integers and long/ . . | l3tol(3C) |
| ae: | 3Com 10 Mb/s Ethernet interface. | ae(5) |
| diagnostic. dual: | 3Com Ethernet interface . . . | dual(1M) |
| comparison. diff3: | 3-way differential file . . . . | diff3(1) |
| 4014 terminal. | 4014: paginator for the Tektronix | 4014(1) |
| 4014: paginator for the Tektronix | 4014 terminal. . . . . | 4014(1) |
| set42sig: set | 4.2 BSD signal interface. . . | set42sig(3) |
| the DASI 450 terminal. | 450: handle special functions of | 450(1) |
| special functions of the DASI | 450 terminal. 450: handle . . | 450(1) |
| f77: Fortran | 77 compiler. . . . . | f77(1) |
| integer and base-64 ASCII/ | a64l, l64a: convert between long | a64l(3C) |
| | abort: generate an IOT fault. . | abort(3C) |
| | abort: terminate Fortran program. | abort(3F) |
| Fortran absolute value. | abs, iabs, dabs, cabs, zabs: . . | abs(3F) |
| value. | abs: return integer absolute . . | abs(3C) |
| abs: return integer | absolute value. . . . . | abs(3C) |
| iabs, dabs, cabs, zabs: Fortran | absolute value. abs, . . . . | abs(3F) |
| fabs: floor, ceiling, remainder, | absolute value functions. /fmod, | floor(3M) |
| accept: | accept a connection on a socket. | accept(2N) |
| socket. | accept: accept a connection on a | accept(2N) |
| | accept: allow LP requests. . . | accept(1M) |
| a file. touch: update | access and modification times of | touch(1) |
| utime: set file | access and modification times. | utime(2) |
| of a file. | access: determine accessibility | access(2) |
| getgroups: get group | access list. . . . . | getgroups(2) |
| initgroups: initialize group | access list. . . . . | initgroups(3) |
| setgroups: set group | access list. . . . . | setgroups(2) |
| machine/ sputl, sgetl: | access long integer data in a . | sputl(3X) |
| phys: allow a process to | access physical addresses. . . | phys(2) |
| ldfcn: common object file | access routines. . . . . | ldfcn(3X) |
| copy file systems for optimal | access time. dcopy: . . . . | dcopy(1M) |
| /setutent, endutent, utmpname: | access utmp file entry. . . . | getut(3C) |
| access: determine | accessibility of a file. . . . . | access(2) |
| acct: enable or disable process | accounting. . . . . | acct(2) |
| acctcon1, acctcon2: connect-time | accounting. . . . . | acctcon(1M) |
| acctprc1, acctprc2: process | accounting. . . . . | acctprc(1M) |
| turnacct: shell procedures for | accounting. /shutacct, startup, | acctsh(1M) |
| /accton, acctwtmp: overview of | accounting and miscellaneous/ | acct(1M) |
| of accounting and miscellaneous | accounting commands. /overview | acct(1M) |
| diskusg: generate disk | accounting data by user ID. . | diskusg(1M) |

acct: per-process accounting file format. . . . acct(4)
acctcom: search and print process accounting file(s). . . . . acctcom(1M)
acctmerg: merge or add total accounting files. . . . . acctmerg(1M)
mclock: return Fortran time accounting. . . . . mclock(3F)
command summary from per-process accounting records. acctcms: . acctcms(1M)
wtmpfix: manipulate connect accounting records. fwtmp, . fwtmp(1M)
runacct: run daily accounting. . . . . runacct(1M)
accounting. acct: enable or disable process acct(2)
format. acct: per-process accounting file acct(4)
per-process accounting records. acctcms: command summary from acctcms(1M)
accounting file(s). acctcom: search and print process acctcom(1M)
accounting. acctcon1, acctcon2: connect-time acctcon(1M)
accounting. acctcon1, acctcon2: connect-time . . . acctcon(1M)
acctwtmp: overview of accounting/ acctdisk, acctdusg, accton, . . acct(1M)
overview of accounting/ acctdisk, acctdusg, accton, acctwtmp: . acct(1M)
accounting files. acctmerg: merge or add total . acctmerg(1M)
accounting/ acctdisk, acctdusg, accton, acctwtmp: overview of acct(1M)
accounting. acctprc1, acctprc2: process . . acctprc(1M)
acctprc1, acctprc2: process accounting. . acctprc(1M)
and/ acctdisk, acctdusg, accton, acctwtmp: overview of accounting acct(1M)
functions. sin, cos, tan, asin, acos, atan, atan2: trigonometric trig(3M)
intrinsic function. acos, dacos: Fortran arccosine acos(3F)
rdumpfs: file system dump across the network. . . . . rdumpfs(1M)
restore a file system dump across the network. rrestore: . rrestore(1M)
signal. signal: specify Fortran action on receipt of a system . signal(3F)
killall: kill all active processes. . . . . killall(1M)
sag: system activity graph. . . . . sag(1G)
sadc, sa1, sa2: system activity report package. . . . sadc(1M)
sar: system activity reporter. . . . . sar(1)
print current SCCS file editing activity. sact: . . . . . sact(1)
report process data and system activity. timex: time a command; timex(1)
a random, hopefully interesting, adage. fortune: print . . . . fortune(6)
adb: debugger. . . . . adb(1)
acctmerg: merge or add total accounting files. . . acctmerg(1M)
putenv: change or add value to environment. . . putenv(3C)
bibliographic database. addbib: create or extend . . . addbib(1)
get file/ setmntent, getmntent, addmntent, endmntent, hasmntopt: getmntent(3)
setptabent,/ getptabent, addptabent, endptabent, . . . getptabent(3)
read an interface's Ethernet address.. etheraddr: . . . . etheraddr(1M)
/inet_lnaof, inet_netof: Internet address manipulation routines. inet(3N)
arp: Address Resolution Protocol. . arp(5P)
a process to access physical addresses. phys: allow . . . phys(2)
allow synchronization of the/ adjtime: correct the time to . . adjtime(2)
files. admin: create and administer SCCS admin(1)
admin: create and administer SCCS files. . . . admin(1)
escher: autorecovery administration. . . . . escher(1M)
swap: swap administrative interface. . . . swap(1M)
file format. afm: Adobe PostScript font metrics afm(7)
flock: apply or remove an advisory lock on an open file. . flock(2)
interface. ae: 3Com 10 Mb/s Ethernet . ae(5)
yes: be repetitively affirmative. . . . . yes(1)

metrics file format. afm: Adobe PostScript font . afm(7)
part of complex argument. aimag, dimag: Fortran imaginary aimag(3F)
intrinsic function. aint, dint: Fortran integer part . aint(3F)
alarm: set a process's alarm clock. . . . . alarm(2)
clock. alarm: set a process's alarm . alarm(2)
sendmail. aliases: aliases file for . . . . aliases(4)
locate a program file including aliases and paths. which: . . which(1)
aliases: aliases file for sendmail. . . . aliases(4)
the data base for the mail aliases file. /rebuild . . . . newaliases(1M)
aliens: alien invaders attack the earth. aliens(6)
earth. aliens: alien invaders attack the aliens(6)
sbrk: change data segment space allocation. brk, . . . . . brk(2)
calloc, cfree: main memory allocator. /free, realloc, . . . malloc(3C)
mallinfo: fast main memory allocator. /calloc, mallopt, . . malloc(3X)
physical addresses. phys: allow a process to access . . phys(2)
accept: allow LP requests. . . . . accept(1M)
adjtime: correct the time to allow synchronization of the/ . adjtime(2)
logarithm intrinsic/ log, alog, dlog, clog: Fortran natural log(3F)
logarithm intrinsic/ log10, alog10, dlog10: Fortran common log10(3F)
information for bad block/ altblk: alternate block . . . . altblk(4)
bad block handling. altblk: alternate block information for altblk(4)
Fortran maximum-value/ max, max0, amax0, max1, amax1, dmax1: max(3F)
max, max0, amax0, max1, amax1, dmax1: Fortran/ . . . max(3F)
Fortran minimum-value/ min, min0, amin0, min1, amin1, dmin1: . min(3F)
min, min0, amin0, min1, amin1, dmin1: Fortran/ . . . min(3F)
intrinsic functions. mod, amod, dmod: Fortran remaindering mod(3F)
of a document. style: analyze surface characteristics style(1)
rshift: Fortran bitwise boolean/ and, or, xor, not, lshift, . . . bool(3F)
sigstack: set and/or get signal stack context. sigstack(2)
whereis: locate source, binary, and/or manual for program. . whereis(1)
sort: sort and/or merge files. . . . . sort(1)
terminal. worms: animate worms on a display . worms(6)
rain: animated raindrops display. . rain(6)
Fortran nearest integer/ anint, dnint, nint, idnint: . . . round(3F)
bcd: convert to antique media. . . . . bcd(6)
editor output. a.out: common assembler and link a.out(4)
files. aouthdr.h: a.out header for common object aouthdr(4)
common object files. aouthdr.h: a.out header for . . aouthdr(4)
execute a Macintosh binary application. launch: . . . . launch(1)
arguments. apply: apply a command to a set of . apply(1)
of arguments. apply: apply a command to a set apply(1)
on an open file. flock: apply or remove an advisory lock flock(2)
keyword lookup. apropos: locate commands by apropos(1)
Postprocessor for the Autologic APS-5 phototypesetter. daps: . daps(1)
maintainer for portable/ ar: archive and library . . . . ar(1)
ar: common archive file format. ar(4)
number: convert Arabic numerals to English. . number(6)
language. bc: arbitrary-precision arithmetic . bc(1)
acos, dacos: Fortran arccosine intrinsic function. . acos(3F)
for portable archives. ar: archive and library maintainer ar(1)
cpio: format of cpio archive. . . . . cpio(4)

ar: common archive file format. . . . . ar(4)
archive header of a member of an archive file. ldahread: read the ldahread(3X)
archive file. ldahread: read the archive header of a member of an ldahread(3X)
tp: manipulate tape archive. . . . . tp(1)
tar: tape file archiver. . . . . tar(1)
library maintainer for portable archives. ar: archive and . . ar(1)
cpio: copy file archives in and out. . . . . cpio(1)
asin, dasin: Fortran arcsine intrinsic function. . . asin(3F)
atan2, datan2: Fortran arctangent intrinsic function. . atan2(3F)
atan, datan: Fortran arctangent intrinsic function. . atan(3F)
Fortran imaginary part of complex argument. aimag, dimag: . . aimag(3F)
return Fortran command-line argument. getarg: . . . . . getarg(3F)
varargs: handle variable argument list. . . . . varargs(3X)
formatted output of a varargs argument list. /vsprintf: print . vprintf(3S)
command. xargs: construct argument list(s) and execute . xargs(1)
getopt: get option letter from argument vector. . . . . getopt(3C)
apply a command to a set of arguments. apply: . . . . . apply(1)
expr: evaluate arguments as an expression. . expr(1)
echo: echo arguments. . . . . echo(1)
iargc: return command line arguments. . . . . iargc(3F)
bc: arbitrary-precision arithmetic language. . . . . bc(1)
number facts. arithmetic: provide drill in . . arithmetic(6)
arp: Address Resolution Protocol. arp(5P)
ftp: ARPANET file transfer program. ftp(1N)
biff: be notified if mail arrives and who it is from. . . biff(1)
as: common assembler. . . . as(1)
asa: interpret ASA carriage control characters. asa(1)
control characters. asa: interpret ASA carriage . asa(1)
ascii: map of ASCII character set. . . . . ascii(5)
convert date and time to ASCII. /tzset, tzsetwall: . . . ctime(3)
set. ascii: map of ASCII character ascii(5)
between long integer and base-64 ASCII string. /l64a: convert . a64l(3C)
number. atof: convert ASCII string to floating-point . atof(3C)
ctime, localtime, gmtime, asctime, tzset, tzsetwall:/ . . ctime(3)
trigonometric/ sin, cos, tan, asin, acos, atan, atan2: . . . trig(3M)
intrinsic function. asin, dasin: Fortran arcsine . . asin(3F)
help: ask for help in using SCCS. . help(1)
a.out: common assembler and link editor output. a.out(4)
as: common assembler. . . . . as(1)
assert: verify program assertion. assert(3X)
assert: verify program assertion. . . . . assert(3X)
setbuf, setvbuf: assign buffering to a stream. . setbuf(3S)
device nodes. pname: associate named partitions with pname(1M)
nfssvc, async_daemon: NFS daemons. nfssvc(2)
later time. at, batch: execute commands at a at(1)
sin, cos, tan, asin, acos, atan, atan2: trigonometric/ . . trig(3M)
intrinsic function. atan, datan: Fortran arctangent atan(3F)
intrinsic function. atan2, datan2: Fortran arctangent atan2(3F)
sin, cos, tan, asin, acos, atan, atan2: trigonometric functions. trig(3M)
floating-point number. atof: convert ASCII string to . atof(3C)
strtol, atol, atoi: convert string to integer. . strtol(3C)

integer. strtol, atol, atoi: convert string to . . strtol(3C)
signals and wait for/ sigpause: atomically release blocked . . sigpause(2)
aliens: alien invaders attack the earth. . . . . aliens(6)
up-to-date kernel. autoconfig: build a new . . . autoconfig(1M)
daps: Postprocessor for the Autologic APS-5 phototypesetter. daps(1)
autorobots: escape from the automatic robots. . . . . autorobots(6)
escher: autorecovery administration. . escher(1M)
system repair. autorecovery: standalone file . autorecovery(8)
automatic robots. autorobots: escape from the . autorobots(6)
environment. launch: launch an A/UX kernel from the standalone launch(8)
dumps out information from A/UX kernels. module_dump: module_dump(1M)
chgnod: change current A/UX system nodename. . . chgnod(1M)
/set up for and clean up after A/UX Toolbox programs. . . toolboxd(1M)
lav: print load average statistics. . . . . lav(1)
processing language. awk: pattern scanning and . . awk(1)
back: the game of backgammon. back(6)
back: the game of backgammon. . . . . back(6)
finc: fast incremental backup. . . . . finc(1M)
frec: recover files from a backup tape. . . . . frec(1M)
alternate block information for bad block handling. altblk: . . altblk(4)
badblk: set or update bad block information. . . . badblk(1M)
information. badblk: set or update bad block badblk(1M)
banner: make posters. . . . . banner(1)
banner7: print large banner on printer. . . . . banner7(1)
printer. banner7: print large banner on banner7(1)
newaliases: rebuild the data base for the mail aliases file. . newaliases(1M)
hosts: host name data base. . . . . hosts(4)
networks: network name data base. . . . . networks(4N)
ttytype: data base of terminal types by port. ttytype(4)
remote host phone number data base. phones: . . . . . phones(4)
protocols: protocol name data base. . . . . protocols(4N)
servers: Inet server data base. . . . . servers(4)
services: service name data base. . . . . services(4N)
delete, firstkey, nextkey: data base subroutines. /fetch, store, dbm(3X)
termcap: terminal capability data base. . . . . termcap(4)
terminal capability data base. terminfo: . . . . . terminfo(4)
ypcat: print values in a YP data base. . . . . ypcat(1)
convert between long integer and base-64 ASCII string. /l64a: . a64l(3C)
portions of pathnames. basename, dirname: deliver . basename(1)
later time. at, batch: execute commands at a at(1)
arithmetic language. bc: arbitrary-precision . . . bc(1)
bcd: convert to antique media. bcd(6)
initialization shell/ brc, bcheckrc, rc, powerfail: system brc(1M)
string operations. bcopy, bcmp, bzero, ffs: bit and byte . bstring(3)
byte string operations. bcopy, bcmp, bzero, ffs: bit and bstring(3)
bcopy: interactive block copy. bcopy(1M)
bdiff: diff large files. . . . . bdiff(1)
cb: C program beautifier. . . . . cb(1)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)
bfs: big file scanner. . . . . bfs(1)
addbib: create or extend bibliographic database. . . . addbib(1)

roffbib: run off bibliographic database. . . . roffbib(1)
sortbib: sort bibliographic database. . . . sortbib(1)
a/ /build inverted index for a bibliography, find references in lookbib(1)
find references in a bibliography. /a bibliography, lookbib(1)
and who it is from. biff: be notified if mail arrives biff(1)
comsat: biff(1) server. . . . . comsat(1M)
bfs: big file scanner. . . . . bfs(1)
program. whereis: locate source, binary, and/or manual for . . whereis(1)
launch: execute a Macintosh binary application. . . . . launch(1)
cpset: install object files in binary directories. . . . . cpset(1M)
mail. /uudecode: encode/decode a binary file for transmission via uuencode(1C)
strings in an object, or other binary file. /find the printable . strings(1)
fread, fwrite: binary input/output. . . . . fread(3S)
bsearch: binary search a sorted table. . bsearch(3C)
tfind, tdelete, twalk: manage binary search trees. tsearch, . tsearch(3C)
bind: bind a name to a socket. . . . bind(2N)
bind: bind a name to a socket. bind(2N)
ypbind: yellow pages server and binder processes. ypserv, . . ypserv(1M)
nfsd, biod: NFS daemons. . . . . nfsd(1M)
bcopy, bcmp, bzero, ffs: bit and byte string operations. bstring(3)
reset: set or reset the teletype bits to a sensible state. tset, . tset(1)
xor, not, lshift, rshift: Fortran bitwise boolean functions. /or, bool(3F)
bj: the game of black jack. . . bj(6)
bj: the game of black jack. . . . . bj(6)
bcopy: interactive block copy. . . . . bcopy(1M)
sum: print checksum and block count of a file. . . . . sum(1)
block information for bad block handling. /alternate . . altblk(4)
badblk: set or update bad block information. . . . . badblk(1M)
handling. altblk: alternate block information for bad block altblk(4)
sigblock: block signals. . . . . sigblock(2)
blt, blt512: block transfer data. . . . . blt(3C)
bzb: format of Block Zero Blocks. . . . . bzb(4)
sigpause: atomically release blocked signals and wait for/ . sigpause(2)
bzb: format of Block Zero Blocks. . . . . bzb(4)
df: report number of free disk blocks. . . . . df(1)
blt, blt512: block transfer data. blt(3C)
blt, blt512: block transfer data. . . blt(3C)
all users over a network running B-NET software. rwall: write to rwall(1M)
lshift, rshift: Fortran bitwise boolean functions. /or, xor, not, bool(3F)
boot: startup procedures. . . boot(8)
command programming/ sh, rsh: Bourne shell, standard/restricted sh(1)
system initialization shell/ brc, bcheckrc, rc, powerfail: . brc(1M)
space allocation. brk, sbrk: change data segment brk(2)
modest-sized programs. bs: a compiler/interpreter for . bs(1)
set42sig: set 4.2 BSD signal interface. . . . . set42sig(3)
facilities. sigvec: optional BSD-compatible software signal sigvec(2)
"optimal" cursor/ curses5.0: BSD-style screen functions with curses5.0(3X)
table. bsearch: binary search a sorted bsearch(3C)
setbuf, setvbuf: assign buffering to a stream. . . . . setbuf(3S)
autoconfig: build a new up-to-date kernel. autoconfig(1M)
database. ypinit: build and install yellow pages ypinit(1M)

bibliography,/ lookbib, indxbib: build inverted index for a . . lookbib(1)
mknod: build special file. . . . . mknod(1M)
values between host and network byte order. /ntohs: convert . . byteorder(3N)
bcopy, bcmp, bzero, ffs: bit and byte string operations. . . . bstring(3)
swab: swap bytes. . . . . swab(3C)
bzb: format of Block Zero Blocks. bzb(4)
operations. bcopy, bcmp, bzero, ffs: bit and byte string . bstring(3)
cc: C compiler. . . . . cc(1)
cflow: generate C flowgraph. . . . . cflow(1)
cpp: the C language preprocessor. . . cpp(1)
cb: C program beautifier. . . . . cb(1)
lint: a C program checker. . . . . lint(1)
cxref: generate C program cross-reference. . cxref(1)
ctags: maintain a tags file for a C program. . . . . ctags(1)
ctrace: C program debugger. . . . . ctrace(1)
indent: indent and format C program source. . . . . indent(1)
xstr: extract strings from C programs to implement shared/ xstr(1)
with C-like syntax. csh: C shell, a command interpreter csh(1)
error message file by massaging C source. mkstr: create an . . mkstr(1)
value. abs, iabs, dabs, cabs, zabs: Fortran absolute . abs(3F)
ncstats: display kernel name cache statistics. . . . . ncstats(1M)
cal: print calendar. . . . . cal(1)
dc: desk calculator. . . . . dc(1)
cal: print calendar. . . . . cal(1)
calendar: reminder service. . . calendar(1)
cu: call another system. . . . . cu(1C)
data returned by stat system call. stat: . . . . . stat(5)
malloc, free, realloc, calloc, cfree: main memory/ . malloc(3C)
main/ malloc, free, realloc, calloc, mallopt, mallinfo: fast . malloc(3X)
intro: introduction to system calls and error numbers. . . . intro(2)
routines for remote procedure calls. rpc: library . . . . . rpc(3N)
line printer. lp, cancel: send/cancel requests to a lp(1)
termcap: terminal capability data base. . . . . termcap(4)
terminfo: terminal capability data base. . . . . terminfo(4)
cribbage: the card game cribbage. . . . . cribbage(6)
asa: interpret ASA carriage control characters. . . asa(1)
cat: concatenate and print files. cat(1)
and uncompress files, and cat them. /ccat: compress . . compact(1)
cb: C program beautifier. . . cb(1)
cc: C compiler. . . . . cc(1)
files, and/ compact, uncompact, ccat: compress and uncompress compact(1)
function. cos, dcos, ccos: Fortran cosine intrinsic . cos(3F)
of an SCCS delta. cdc: change the delta commentary cdc(1)
remainder, absolute value/ floor, ceil, fmod, fabs: floor, ceiling, floor(3M)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute/ . floor(3M)
intrinsic function. exp, dexp, cexp: Fortran exponential . . exp(3F)
cflow: generate C flowgraph. . cflow(1)
malloc, free, realloc, calloc, cfree: main memory allocator. malloc(3C)
tuning. kconfig: change a kernel's parameters for kconfig(1M)
nodename. chgnod: change current A/UX system . chgnod(1M)
allocation. brk, sbrk: change data segment space . . brk(2)

chsh: change default login shell. . . chsh(1)
chfn: change finger entry. . . . . chfn(1)
pages. yppasswd: change login password in yellow yppasswd(1)
passwd: change login password. . . . passwd(1)
chmod: change mode. . . . . chmod(1)
chmod: change mode of file. . . . . chmod(2)
environment. putenv: change or add value to . . . putenv(3C)
chown, fchown: change owner and group of a file. chown(2)
chown, chgrp: change owner or group. . . . chown(1)
nice: change priority of a process. . nice(2)
chroot: change root directory. . . . . chroot(2)
command. chroot: change root directory for a . . chroot(1M)
SCCS delta. cdc: change the delta commentary of an cdc(1)
newform: change the format of a text file. newform(1)
rename: change the name of a file. . . rename(2)
color. scr_color: program to change the terminal's screen . scr_color(1)
delta: make a delta (change) to an SCCS file. . . delta(1)
chdir: change working directory. . . chdir(2)
yppush: force propagation of a changed YP map. . . . . yppush(1M)
pipe: create an interprocess channel. . . . . pipe(2)
/sngl, dble, cmplx, dcmplx, ichar, char: explicit Fortran type/ . . ftype(3F)
ungetc: push character back into input stream. ungetc(3S)
neqn. eqnchar: special character definitions for eqn and eqnchar(5)
freq: report on character frequencies in a file. freq(1)
cuserid: get character login name of the user. cuserid(3S)
iwmap: format of iwprep(1) character map description files. iwmap(4)
getc, getchar, fgetc, getw: get character or word from a stream. getc(3S)
putc, putchar, fputc, putw: put character or word on a stream. putc(3S)
ascii: map of ASCII character set. . . . . ascii(5)
style: analyze surface characteristics of a document. style(1)
interpret ASA carriage control characters. asa: . . . . . asa(1)
_tolower, toascii: translate characters. /tolower, _toupper, conv(3C)
iscntrl, isascii: classify characters. /isprint, isgraph, . ctype(3C)
given/ sumdir: sum and count characters in the files in the . sumdir(1)
tr: translate characters. . . . . tr(1)
lastlogin, monacct, nulladm,/ chargefee, ckpacct, dodisk, . . acctsh(1M)
robots. chase: try to escape the killer . chase(6)
chdir: change working directory. chdir(2)
fsck: file system consistency check and interactive repair. . fsck(1M)
the mm macros. checkmm: check documents formatted with checkmm(1)
checknr: check nroff/troff files. . . . . checknr(1)
text for otroff. cw, checkcw: prepare constant-width cw(1)
lint: a C program checker. . . . . lint(1)
pwck, grpck: password/group file checkers. . . . . pwck(1M)
copy file systems with label checking. volcopy, labelit: . . volcopy(1M)
formatted with the mm macros. checkmm: check documents . checkmm(1)
checknr: check nroff/troff files. checknr(1)
file. sum: print checksum and block count of a sum(1)
chfn: change finger entry. . . chfn(1)
system nodename. chgnod: change current A/UX chgnod(1M)
chown, chgrp: change owner or group. chown(1)

times: get process and child process times. . . . . times(2)
terminate. wait: wait for child process to stop or . . . wait(2)
terminate. wait3: wait for child process to stop or . . . wait3(2N)
set the system time/real time chip. mactime: . . . . . mactime(1M)
chmod: change mode. . . . . chmod(1)
chmod: change mode of file. . chmod(2)
group. chown, chgrp: change owner or chown(1)
group of a file. chown, fchown: change owner and chown(2)
chroot: change root directory. . chroot(2)
a command. chroot: change root directory for chroot(1M)
chsh: change default login shell. chsh(1)
monacct, nulladm,/ chargefee, ckpacct, dodisk, lastlogin, . . acctsh(1M)
isgraph, iscntrl, isascii: classify characters. /isprint, . ctype(3C)
toolboxdaemon: set up for and clean up after A/UX Toolbox/ toolboxd(1M)
uuclean: uucp spool directory clean-up. . . . . uuclean(1M)
clear: clear terminal screen. . clear(1)
clri: clear inode. . . . . clri(1M)
clear: clear terminal screen. . . . . clear(1)
inquiries. ferror, feof, clearerr, fileno: stream status . ferror(3S)
ypclnt: yellow pages client interface. . . . . ypclnt(3N)
shell, a command interpreter with C-like syntax. csh: C . . . . csh(1)
synchronization of the system clock. /correct the time to allow adjtime(2)
alarm: set a process's alarm clock. . . . . alarm(2)
cron: clock daemon. . . . . cron(1M)
nonvolatile memory/time of day clock interface. nvram: . . . nvram(7)
clock: report CPU time used. . clock(3C)
intrinsic/ log, alog, dlog, clog: Fortran natural logarithm log(3F)
ldclose, ldaclose: close a common object file. . ldclose(3X)
close: close a file descriptor. . . . . close(2)
close: close a file descriptor. . close(2)
fclose, fflush: close or flush a stream. . . . fclose(3S)
/telldir, seekdir, rewinddir, closedir: directory operations. directory(3)
clri: clear inode. . . . . clri(1M)
format. cml: configuration master list . cml(4)
cmp: compare two files. . . . cmp(1)
/idint, real, float, sngl, dble, cmplx, dcmplx, ichar, char:/ . ftype(3F)
col: filter reverse linefeeds. . . col(1)
terminal previewing. colcrt: filter nroff output for . colcrt(1)
to change the terminal's screen color. scr_color: program . . scr_color(1)
file. colrm: remove columns from a colrm(1)
colrm: remove columns from a file. . . . . colrm(1)
comb: combine SCCS deltas. . comb(1)
comb: combine SCCS deltas. . . . comb(1)
common to two sorted files. comm: select or reject lines . comm(1)
nice: run a command at low priority. . . nice(1)
change root directory for a command. chroot: . . . . . chroot(1M)
env: set environment for command execution. . . . . env(1)
uux: UNIX-to-UNIX system command execution. . . . . uux(1C)
system: issue a shell command from Fortran. . . . system(3F)
nohup: run a command immune to hangups. nohup(1)
standalone environment. sash: a command interpreter for the . sash(8)

syntax. csh: C shell, a command interpreter with C-like csh(1)
whatis: describe what a command is. . . . . whatis(1)
iargc: return command line arguments. . . iargc(3F)
getopt: parse command options. . . . . getopt(1)
ksh: Korn shell, a command programming language. ksh(1)
/Bourne shell, standard/restricted command programming language. sh(1)
returning a stream to a remote command. /ruserok: routines for rcmd(3N)
system activity. timex: time a command; report process data and timex(1)
rexec: return stream to a remote command. . . . . rexec(3N)
accounting records. acctcms: command summary from per-process acctcms(1M)
system: issue a shell command. . . . . system(3S)
test: condition evaluation command. . . . . test(1)
time: time a command. . . . . time(1)
apply: apply a command to a set of arguments. apply(1)
argument list(s) and execute command. xargs: construct . xargs(1)
getarg: return Fortran command-line argument. . . getarg(3F)
and miscellaneous accounting commands. /overview of accounting acct(1M)
at, batch: execute commands at a later time. . . at(1)
apropos: locate commands by keyword lookup. apropos(1)
install: install commands. . . . . install(1M)
magic: file command's magic number file. magic(4)
cdc: change the delta commentary of an SCCS delta. cdc(1)
ar: common archive file format. . ar(4)
output. a.out: common assembler and link editor a.out(4)
as: common assembler. . . . . as(1)
log10, alog10, dlog10: Fortran common logarithm intrinsic/ . log10(3F)
routines. ldfcn: common object file access . . ldfcn(3X)
ldopen, ldaopen: open a common object file for reading. ldopen(3X)
/line number entries of a common object file function. . ldlread(3X)
ldclose, ldaclose: close a common object file. . . . . ldclose(3X)
read the file header of a common object file. ldfhread: ldfhread(3X)
number entries of a section of a common object file. /seek to line ldlseek(3X)
to the optional file header of a common object file. /seek . . ldohseek(3X)
entries of a section of a common object file. /relocation ldrseek(3X)
/section header of a common object file. . . . . ldshread(3X)
to an indexed/named section of a common object file. /seek . . ldsseek(3X)
of a symbol table entry of a common object file. /the index ldtbindex(3X)
indexed symbol table entry of a common object file. /read an . ldtbread(3X)
seek to the symbol table of a common object file. ldtbseek: ldtbseek(3X)
linenum: line number entries in a common object file. . . . . linenum(4)
nm: print name list of common object file. . . . . nm(1)
relocation information for a common object file. reloc: . . reloc(4)
scnhdr: section header for a common object file. . . . . scnhdr(4)
format. syms: common object file symbol table syms(4)
aouthdr.h: a.out header for common object files. . . . . aouthdr(4)
filehdr: file header for common object files. . . . . filehdr(4)
ld: link editor for common object files. . . . . ld(1)
size: print section sizes of common object files. . . . . size(1)
comm: select or reject lines common to two sorted files. . comm(1)
ipcs: report interprocess communication facilities status. ipcs(1)
ftok: standard interprocess communication package. . . ftok(3C)

talkd: remote user communication server. . . . talkd(1M)
socket: create an endpoint for communication. . . . . socket(2N)
the system. users: compact list of users who are on users(1)
compress and uncompress files,/ compact, uncompact, ccat: . . compact(1)
differential file and directory comparator. diff: . . . . . diff(1)
cmp: compare two files. . . . . cmp(1)
file. sccsdiff: compare two versions of an SCCS sccsdiff(1)
lge, lgt, lle, llt: string comparision intrinsic functions. lge(3F)
diff3: 3-way differential file comparison. . . . . diff3(1)
dircmp: directory comparison. . . . . dircmp(1)
getcompat: set or get process compatibility mode. setcompat, setcompat(2)
expression. regcmp, regex: compile and execute a regular regcmp(3X)
regexp: regular expression compile and match routines. . regexp(5)
regcmp: regular expression compile. . . . . regcmp(1)
rez: compile resources. . . . . rez(1)
term: format of compiled term file.. . . . . term(4)
cc: C compiler. . . . . cc(1)
f77: Fortran 77 compiler. . . . . f77(1)
tic: terminfo compiler. . . . . tic(1M)
tzic: time zone compiler. . . . . tzic(1M)
yacc: yet another compiler-compiler. . . . . yacc(1)
modest-sized programs. bs: a compiler/interpreter for . . . bs(1)
erf, erfc: error function and complementary error function. erf(3M)
dimag: Fortran imaginary part of complex argument. aimag, . . aimag(3F)
function. conjg, dconjg: Fortran complex conjugate intrinsic . conjg(3F)
pack, pcat, unpack: compress and expand files. . . pack(1)
and/ compact, uncompact, ccat: compress and uncompress files, compact(1)
table entry of a/ ldtbindex: compute the index of a symbol ldtbindex(3X)
comsat: biff(1) server. . . . comsat(1M)
cat: concatenate and print files. . . cat(1)
test: condition evaluation command. test(1)
resolver: resolver configuration file. . . . . resolver(4)
uvar: returns system-specific configuration information. . . uvar(2)
cml: configuration master list format. cml(4)
parameters. ifconfig: configure network interface . ifconfig(1M)
lpadmin: configure the LP spooling system. lpadmin(1M)
conjugate intrinsic function. conjg, dconjg: Fortran complex conjg(3F)
conjg, dconjg: Fortran complex conjugate intrinsic function. . conjg(3F)
fwtmp, wtmpfix: manipulate connect accounting records. . fwtmp(1M)
a socket. connect: initiate a connection on connect(2N)
tip: connect to a remote system. . tip(1C)
getpeername: get name of connected peer. . . . . getpeername(2N)
an out-going terminal line connection. dial: establish . . dial(3C)
accept: accept a connection on a socket. . . . accept(2N)
connect: initiate a connection on a socket. . . . connect(2N)
shut down part of a full-duplex connection. shutdown: . . . shutdown(2N)
listen: listen for connections on a socket. . . . listen(2N)
acctcon1, acctcon2: connect-time accounting. . . acctcon(1M)
repair. fsck: file system consistency check and interactive fsck(1M)
console: keyboard/screen driver. console(7)
math: math functions and constants. . . . . math(5)

cw, checkcw: prepare constant-width text for otroff. . cw(1)
mkfs1b: construct a file system. . . . mkfs1b(1M)
mkfs: construct a file system. . . . mkfs(1M)
execute command. xargs: construct argument list(s) and xargs(1)
remove nroff/troff, tbl, and eqn constructs. deroff: . . . . . deroff(1)
ls: list contents of directory. . . . . ls(1)
set and/or get signal stack context. sigstack: . . . . . sigstack(2)
csplit: context split. . . . . csplit(1)
asa: interpret ASA carriage control characters. . . . . asa(1)
ioctl: control device. . . . . ioctl(2)
fcntl: file control. . . . . fcntl(2)
init, telinit: process control initialization. . . . . init(1M)
msgctl: message control operations. . . . . msgctl(2)
semctl: semaphore control operations. . . . . semctl(2)
shmctl: shared memory control operations. . . . . shmctl(2)
fcntl: file control options. . . . . fcntl(5)
tcp: Internet Transmission Control Protocol. . . . . tcp(5P)
uucp status inquiry and job control. uustat: . . . . . uustat(1C)
vc: version control. . . . . vc(1)
tty: controlling terminal interface. tty(7)
conv: object file converter. . . conv(1)
term: conventional names for terminals. term(5)
char: explicit Fortran type conversion. /dcmplx, ichar, . ftype(3F)
units: conversion program. . . . . units(1)
dd: convert and copy a file. . . . dd(1)
English. number: convert Arabic numerals to . number(6)
floating-point number. atof: convert ASCII string to . . . atof(3C)
and long integers. l3tol, ltol3: convert between 3-byte integers l3tol(3C)
base-64 ASCII/ a64l, l64a: convert between long integer and a64l(3C)
/asctime, tzset, tzsetwall: convert date and time to ASCII. ctime(3)
string. ecvt, fcvt, gcvt: convert floating-point number to ecvt(3C)
scanf, fscanf, sscanf: convert formatted input. . . . scanf(3S)
double-precision number. strtod: convert string to . . . . . strtod(3C)
strtol, atol, atoi: convert string to integer. . . . strtol(3C)
bcd: convert to antique media. . . bcd(6)
to/ psdit: convert troff intermediate format psdit(1)
htonl, htons, ntohl, ntohs: convert values between host and/ byteorder(3N)
conv: object file converter. . . . . conv(1)
dd: convert and copy a file. . . . . dd(1)
bcopy: interactive block copy. . . . . bcopy(1M)
cpio: copy file archives in and out. . cpio(1)
access time. dcopy: copy file systems for optimal . dcopy(1M)
checking. volcopy, labelit: copy file systems with label . volcopy(1M)
cp: copy files. . . . . cp(1)
rcp: remote file copy. . . . . rcp(1C)
UNIX system to UNIX system copy. uucp, uulog, uuname: . uucp(1C)
public UNIX-to-UNIX system file copy. uuto, uupick: . . . . . uuto(1C)
core: format of core image file. core(4)
core: format of core image file. . . . . core(4)
mem, kmem: core memory. . . . . mem(7)
synchronization of the/ adjtime: correct the time to allow . . . adjtime(2)

intrinsic function. cos, dcos, ccos: Fortran cosine cos(3F)
atan2: trigonometric/ sin, cos, tan, asin, acos, atan, . . . trig(3M)
cosine intrinsic function. cosh, dcosh: Fortran hyperbolic cosh(3F)
sinh, cosh, tanh: hyperbolic functions. sinh(3M)
cos, dcos, ccos: Fortran cosine intrinsic function. . . cos(3F)
cosh, dcosh: Fortran hyperbolic cosine intrinsic function. . . cosh(3F)
the given/ sumdir: sum and count characters in the files in sumdir(1)
sum: print checksum and block count of a file. . . . . sum(1)
wc: word count. . . . . wc(1)
cp: copy files. . . . . cp(1)
cpio: format of cpio archive. . . . . cpio(4)
out. cpio: copy file archives in and cpio(1)
cpio: format of cpio archive. . cpio(4)
cpp: the C language preprocessor. cpp(1)
binary directories. cpset: install object files in . . cpset(1M)
clock: report CPU time used. . . . . clock(3C)
craps: the game of craps. . . . . craps(6)
craps: the game of craps. . . craps(6)
crashes. crash: what to do when the system crash(8)
crash: what to do when the system crashes. . . . . crash(8)
rewrite an existing one. creat: create a new file or . . creat(2)
file. tmpnam, tempnam: create a name for a temporary tmpnam(3S)
existing one. creat: create a new file or rewrite an . creat(2)
fork: create a new process. . . . . fork(2)
document. ndx: create a subject-page index for a ndx(1)
tmpfile: create a temporary file. . . . tmpfile(3S)
communication. socket: create an endpoint for . . . . socket(2N)
massaging C source. mkstr: create an error message file by mkstr(1)
pipe: create an interprocess channel. pipe(2)
admin: create and administer SCCS files. admin(1)
database. addbib: create or extend bibliographic addbib(1)
umask: set and get file creation mask. . . . . umask(2)
file. settc: set the type and creator of a Macintosh resource settc(1)
cribbage: the card game cribbage. . . . . cribbage(6)
cribbage: the card game cribbage. cribbage(6)
cron: clock daemon. . . . . cron(1M)
crontab: user crontab utility. . crontab(1)
crontab: user crontab utility. . . . . crontab(1)
cxref: generate C program cross-reference. . . . . cxref(1)
files. macref: produce cross-reference listing of macro macref(1)
optimization package. curses: CRT screen handling and . . curses(3X)
more: file perusal filter for CRT viewing. . . . . more(1)
crypt: encode/decode. . . . . crypt(1)
DES encryption. crypt, setkey, encrypt: generate crypt(3C)
interpreter with C-like syntax. csh: C shell, a command . . . csh(1)
function. sin, dsin, csin: Fortran sine intrinsic . . sin(3F)
csplit: context split. . . . . csplit(1)
intrinsic function. sqrt, dsqrt, csqrt: Fortran square root . . sqrt(3F)
terminal. ct: spawn getty to a remote . . ct(1C)
C program. ctags: maintain a tags file for a ctags(1)
terminal. ctermid: generate filename for ctermid(3S)

asctime, tzset, tzsetwall:/ ctime, localtime, gmtime, . . ctime(3)
ctrace: C program debugger. . ctrace(1)
cu: call another system. . . . cu(1C)
ttt, cubic: tic-tac-toe. . . . . ttt(6)
chgnod: change current A/UX system nodename. chgnod(1M)
/set or display name of current domain system. . . . domainname(1)
printenv: print the current environment. . . . printenv(1)
get/set unique identifier of current host. /sethostid: . . . gethostid(2N)
sethostname: get/set name of current host. gethostname, . . gethostname(2N)
set or print identifier of current host system. hostid: . hostid(1N)
hostname: set or print name of current host system. . . . . hostname(1N)
/setdomainname: get/set name of current network domain. . . . getdomainname(2N)
activity. sact: print current SCCS file editing . . sact(1)
sigsetmask: set current signal mask. . . . . sigsetmask(2)
uname: print name of current system. . . . . uname(1)
uname: get name of current system. . . . . uname(2)
whoami: print effective current user ID. . . . . whoami(1)
the slot in the utmp file of the current user. ttyslot: find . . ttyslot(3C)
getcwd: get pathname of current working directory. . . getcwd(3C)
pathname. getwd: get current working directory . . getwd(3)
optimization package. curses: CRT screen handling and curses(3X)
functions with "optimal"/ curses5.0: BSD-style screen . curses5.0(3X)
screen functions with "optimal" cursor motion. /BSD-style . . curses5.0(3X)
spline: interpolate smooth curve. . . . . spline(1G)
of the user. cuserid: get character login name cuserid(3S)
each line of a file. cut: cut out selected fields of . cut(1)
line of a file. cut: cut out selected fields of each . cut(1)
constant-width text for otroff. cw, checkcw: prepare . . . . cw(1)
cross-reference. cxref: generate C program . . cxref(1)
absolute value. abs, iabs, dabs, cabs, zabs: Fortran . . . abs(3F)
intrinsic function. acos, dacos: Fortran arccosine . . . acos(3F)
cron: clock daemon. . . . . cron(1M)
errdemon: error-logging daemon. . . . . errdemon(1M)
terminate the error-logging daemon. errstop: . . . . . errstop(1M)
inetd: Internet services daemon. . . . . inetd(1M)
routed: network routing daemon. . . . . routed(1M)
nfsd, biod: NFS daemons. . . . . nfsd(1M)
nfssvc, async_daemon: NFS daemons. . . . . nfssvc(2)
runacct: run daily accounting. . . . . runacct(1M)
postprocessor filter. daiw: Apple ImageWriter II troff daiw(1)
Autologic APS-5 phototypesetter. daps: Postprocessor for the . . daps(1)
Protocol server. ftpd: DARPA Internet File Transfer ftpd(1M)
mapper. portmap: DARPA port to RPC program number portmap(1M)
telnetd: DARPA TELNET protocol server. telnetd(1M)
Protocol server. tftpd: DARPA Trivial File Transfer . tftpd(1M)
/300s: handle special functions of DASI300 and 300s terminals. 300(1)
handle special functions of the DASI 450 terminal. 450: . . 450(1)
function. asin, dasin: Fortran arcsine intrinsic asin(3F)
time a command; report process data and system activity. timex: timex(1)
file. newaliases: rebuild the data base for the mail aliases , newaliases(1M)
hosts: host name data base. . . . . hosts(4)

networks: network name data base. . . . . networks(4N)
port. ttytype: data base of terminal types by ttytype(4)
phones: remote host phone number data base. . . . . phones(4)
protocols: protocol name data base. . . . . protocols(4N)
servers: Inet server data base. . . . . servers(4)
services: service name data base. . . . . services(4N)
store, delete, firstkey, nextkey: data base subroutines. /fetch, . dbm(3X)
termcap: terminal capability data base. . . . . termcap(4)
terminfo: terminal capability data base. . . . . terminfo(4)
ypcat: print values in a YP data base. . . . . ypcat(1)
blt, blt512: block transfer data. . . . . blt(3C)
diskusg: generate disk accounting data by user ID. . . . . diskusg(1M)
sputl, sgetl: access long integer data in a machine independent/ sputl(3X)
plock: lock process, text, or data in memory. . . . . plock(2)
prof: display profile data. . . . . prof(1)
library routines for external data representation. xdr: . . . xdr(3N)
call. stat: data returned by stat system . stat(5)
brk, sbrk: change data segment space allocation. brk(2)
types: primitive system data types. . . . . types(5)
create or extend bibliographic database. addbib: . . . . . addbib(1)
ypfiles: the yellowpages database and directory structure. ypfiles(4)
join: relational database operator. . . . . join(1)
roffbib: run off bibliographic database. . . . . roffbib(1)
sortbib: sort bibliographic database. . . . . sortbib(1)
tput: query terminfo database. . . . . tput(1)
build and install yellow pages database. ypinit: . . . . . ypinit(1M)
ypmake: rebuild yellow pages database. . . . . ypmake(1M)
udp: Internet User Datagram Protocol. . . . . udp(5P)
intrinsic function. atan, datan: Fortran arctangent . . atan(3F)
intrinsic function. atan2, datan2: Fortran arctangent . . atan2(3F)
settimeofday: get/set date and time. gettimeofday, . gettimeofday(2)
tzset, tzsetwall: convert date and time to ASCII. /asctime, ctime(3)
date: print and set the date. . . . . date(1)
date: print and set the date. . . date(1)
/ifix, idint, real, float, sngl, dble, cmplx, dcmplx, ichar, char:/ ftype(3F)
makedbm: make a yellow pages dbm file. . . . . makedbm(1M)
firstkey, nextkey: data base/ dbminit, fetch, store, delete, . dbm(3X)
dc: desk calculator. . . . . dc(1)
/real, float, sngl, dble, cmplx, dcmplx, ichar, char: explicit/ . ftype(3F)
intrinsic function. conjg, dconjg: Fortran complex conjugate conjg(3F)
optimal access time. dcopy: copy file systems for . dcopy(1M)
intrinsic function. cos, dcos, ccos: Fortran cosine . . cos(3F)
intrinsic function. cosh, dcosh: Fortran hyperbolic cosine cosh(3F)
dd: convert and copy a file. . . dd(1)
difference intrinsic/ dim, ddim, idim: Fortran positive . dim(3F)
adb: debugger. . . . . adb(1)
ctrace: C program debugger. . . . . ctrace(1)
fsdb: file system debugger. . . . . fsdb(1M)
sdb: symbolic debugger. . . . . sdb(1)
derez: decompiles a resource file. . . derez(1)
chsh: change default login shell. . . . . chsh(1)

eqnchar: special character definitions for eqn and neqn. . eqnchar(5)
base/ dbminit, fetch, store, delete, firstkey, nextkey: data . dbm(3X)
basename, dirname: deliver portions of pathnames. basename(1)
tail: deliver the last part of a file. . tail(1)
the delta commentary of an SCCS delta. cdc: change . . . . . cdc(1)
delta: make a delta (change) to an SCCS file. delta(1)
delta. cdc: change the delta commentary of an SCCS cdc(1)
rmdel: remove a delta from an SCCS file. . . . rmdel(1)
an SCCS file. delta: make a delta (change) to delta(1)
comb: combine SCCS deltas. . . . . comb(1)
mesg: permit or deny messages. . . . . mesg(1)
file. derez: decompiles a resource . derez(1)
and eqn constructs. deroff: remove nroff/troff, tbl, deroff(1)
crypt, setkey, encrypt: generate DES encryption. . . . . crypt(3C)
whatis: describe what a command is. . whatis(1)
remote: remote host description file. . . . . remote(4)
device-independent troff. font: description files for . . . . . font(5)
format of iwprep(1) character map description files. iwmap: . . iwmap(4)
iwprep: prepare troff description files. . . . . iwprep(1)
troff: description of output language. troff(5)
close: close a file descriptor. . . . . close(2)
dup: duplicate a descriptor. . . . . dup(2)
dup2: duplicate a descriptor. . . . . dup2(3N)
/hasmntopt: get file system descriptor file entry. . . . getmntent(3)
getdtablesize: get descriptor table size . . . . getdtablesize(2)
dc: desk calculator. . . . . dc(1)
file. access: determine accessibility of a . access(2)
file: determine file type. . . . . file(1)
mouse: mouse input device driver. . . . . mouse(7)
extended errors in the specified device. /on/off the reporting of exterr(1M)
lines for finite-width output device. fold: fold long . . . fold(1)
ioctl: control device. . . . . ioctl(2)
devnm: device name. . . . . devnm(1M)
associate named partitions with device nodes. pname: . . . . pname(1M)
font: description files for device-independent troff. . . font(5)
dev_kill: remove special devices from directories. . . dev_kill(1M)
from directories. dev_kill: remove special devices dev_kill(1M)
devnm: device name. . . . . devnm(1M)
intrinsic function. exp, dexp, cexp: Fortran exponential exp(3F)
blocks. df: report number of free disk . df(1)
dual: 3Com Ethernet interface diagnostic. . . . . dual(1M)
terminal line connection. dial: establish an out-going . . dial(3C)
file. dialup: modem escape sequence dialup(4)
wordy sentences; thesaurus for diction. diction, explain: print diction(1)
sentences; thesaurus for/ diction, explain: print wordy . diction(1)
directory comparator. diff: differential file and . . . diff(1)
bdiff: diff large files. . . . . bdiff(1)
comparison. diff3: 3-way differential file . diff3(1)
dim, ddim, idim: Fortran positive difference intrinsic functions. . dim(3F)
sdiff: side-by-side difference program. . . . . sdiff(1)
diffmk: mark differences between files. . . diffmk(1)

comparator. diff: differential file and directory . diff(1)
diff3: 3-way differential file comparison. . diff3(1)
files. diffmk: mark differences between diffmk(1)
difference intrinsic functions. dim, ddim, idim: Fortran positive dim(3F)
complex argument. aimag, dimag: Fortran imaginary part of aimag(3F)
intrinsic function. aint, dint: Fortran integer part . . . aint(3F)
directories. dir: format of System V . . . dir(4)
dircmp: directory comparison. dircmp(1)
install object files in binary directories. cpset: . . . . . cpset(1M)
remove special devices from directories. dev_kill: . . . . dev_kill(1M)
dir: format of System V directories. . . . . dir(4)
rm, rmdir: remove files or directories. . . . . rm(1)
in the files in the given directories. /count characters . sumdir(1)
chdir: change working directory. . . . . chdir(2)
chroot: change root directory. . . . . chroot(2)
uuclean: uucp spool directory clean-up. . . . . uuclean(1M)
diff: differential file and directory comparator. . . . . diff(1)
dircmp: directory comparison. . . . . dircmp(1)
system/ getdirentries: gets directory entries in a file . . . getdirentries(2)
unlink: remove directory entry. . . . . unlink(2)
mkdir: make a directory file. . . . . mkdir(2)
rmdir: remove a directory file. . . . . rmdir(2)
chroot: change root directory for a command. . . chroot(1M)
getpathname of current working directory. getcwd: . . . . . getcwd(3C)
ls: list contents of directory. . . . . ls(1)
mkdir: make a directory. . . . . mkdir(1)
pwd: print working directory name. . . . . pwd(1)
seekdir, rewinddir, closedir: directory operations. /telldir, . directory(3)
ordinary file. mknod: make a directory, or a special or . . . mknod(2)
getwd: get current working directory pathname. . . . . getwd(3)
scandir: scan a directory. . . . . scandir(3)
the yellowpages database and directory structure. ypfiles: . ypfiles(4)
pathnames. basename, dirname: deliver portions of . basename(1)
dis: disassembler. . . . . dis(1)
printers. enable, disable: enable/disable LP . . enable(1)
acct: enable or disable process accounting. . acct(2)
dis: disassembler. . . . . dis(1)
type, modes, speed, and line discipline. getty: set terminal . getty(1M)
routine used to push streams line disciplines. line_push: . . . line_push(3)
line_sane: push streams line disciplines. . . . . line_sane(1M)
diskusg: generate disk accounting data by user ID. diskusg(1M)
df: report number of free disk blocks. . . . . df(1)
diskformat: format a disk. . . . . diskformat(1M)
fd: floppy disk drive interface. . . . . fd(7)
file's in-core state with that on disk. fsync: synchronize a . . fsync(2)
gd: generic disk interface. . . . . gd(7)
read a Macintosh flat file system disk. mfs: . . . . . mfs(1)
dpme: format of disk partition map entries. . . dpme(4)
dp: perform disk partitioning. . . . . dp(1M)
du: summarize disk usage. . . . . du(1)
eject: eject diskette from drive. . . . . eject(1)

diskformat: format a disk. . . diskformat(1M)
data by user ID. diskusg: generate disk accounting diskusg(1M)
mount, umount: mount and dismount file systems. . . . mount(1M)
vedit: screen-oriented (visual) display editor. vi, view, . . . vi(1)
statistics. ncstats: display kernel name cache . . ncstats(1M)
system. domainname: set or display name of current domain domainname(1)
prof: display profile data. . . . . prof(1)
rain: animated raindrops display. . . . . rain(6)
line of a terminal. sysline: display system status on status sysline(1)
worms: animate worms on a display terminal. . . . . worms(6)
hypot: Euclidean distance function. . . . . hypot(3M)
/lcong48: generate uniformly distributed pseudo-random/ . drand48(3C)
rdist: remote file distribution program. . . . . rdist(1)
logarithm intrinsic/ log, alog, dlog, clog: Fortran natural . . log(3F)
intrinsic/ log10, alog10, dlog10: Fortran common logarithm log10(3F)
max, max0, amax0, max1, amax1, dmax1: Fortran maximum-value/ max(3F)
min, min0, amin0, min1, amin1, dmin1: Fortran minimum-value/ min(3F)
intrinsic functions. mod, amod, dmod: Fortran remaindering . mod(3F)
res_mkquery, res_send, res_init, dn_comp, dn_expand: resolver/ resolver(3N)
/res_send, res_init, dn_comp, dn_expand: resolver routines. . resolver(3N)
nearest integer/ anint, dnint, nint, idnint: Fortran . . round(3F)
create a subject-page index for a document. ndx: . . . . . ndx(1)
surface characteristics of a document. style: analyze . . style(1)
a list of subjects from a document. subj: generate . . subj(1)
macros. checkmm: check documents formatted with the mm checkmm(1)
macros. mm: prints documents formatted with the mm mm(1)
mm: macro package for formatting documents. . . . . mm(5)
mmt: typeset documents. . . . . mmt(1)
insert literature references in documents. refer: find and . . refer(1)
nulladm,/ chargefee, ckpacct, dodisk, lastlogin, monacct, . . acctsh(1M)
w: who is on and what they are doing. . . . . w(1)
whodo: who is doing what. . . . . whodo(1M)
get/set name of current network domain. /setdomainname: . . getdomainname(2N)
named: Internet domain name server. . . . . named(1M)
set or display name of current domain system. domainname: domainname(1)
of current domain system. domainname: set or display name domainname(1)
intrinsic/ dprod: Fortran double precision product . . dprod(3F)
strtod: convert string to double-precision number. . . strtod(3C)
/Motorola S-records from downloading into a file. . . . rcvhex(1)
dp: perform disk partitioning. . dp(1M)
map entries. dpme: format of disk partition dpme(4)
product intrinsic function. dprod: Fortran double precision dprod(3F)
nrand48, mrand48, jrand48,/ drand48, erand48, lrand48, . . drand48(3C)
graph: draw a graph. . . . . graph(1G)
grap: pic preprocessor for drawing graphs. . . . . grap(1)
pic: troff preprocessor for drawing pictures. . . . . pic(1)
arithmetic: provide drill in number facts. . . . . arithmetic(6)
eject: eject diskette from drive. . . . . eject(1)
fd: floppy disk drive interface. . . . . fd(7)
console: keyboard/screen driver. . . . . console(7)
mouse: mouse input device driver. . . . . mouse(7)

pty: pseudo terminal driver. . . . . pty(7)
sxt: pseudo-device driver. . . . . sxt(7)
nterm: terminal driving tables for nroff. . . . nterm(5)
intrinsic function. sign, isign, dsign: Fortran transfer-of-sign sign(3F)
intrinsic function. sin, dsin, csin: Fortran sine . . . sin(3F)
intrinsic function. sinh, dsinh: Fortran hyperbolic sine sinh(3F)
intrinsic function. sqrt, dsqrt, csqrt: Fortran square root sqrt(3F)
function. tan, dtan: Fortran tangent intrinsic tan(3F)
intrinsic function. tanh, dtanh: Fortran hyperbolic tangent tanh(3F)
du: summarize disk usage. . . du(1)
diagnostic. dual: 3Com Ethernet interface dual(1M)
rdumpfs: file system dump across the network. . . rdumpfs(1M)
rrestore: restore a file system dump across the network. . . rrestore(1M)
object file. dump: dump selected parts of an dump(1)
dumpfs: incremental file system dump. . . . . dumpfs(1M)
dumpfs: incremental dump format. . . . . dumpfs(4)
od: octal dump. . . . . od(1)
file. dump: dump selected parts of an object dump(1)
tzdump: time zone dumper. . . . . tzdump(1M)
extract error records from dumpfs. errdead: . . . . . errdead(1M)
dumpfs: incremental dump format. dumpfs(4)
dump. dumpfs: incremental file system dumpfs(1M)
kernels. module_dump: dumps out information from A/UX module_dump(1M)
dup: duplicate a descriptor. . . dup(2)
dup2: duplicate a descriptor. . dup2(3N)
dup: duplicate a descriptor. . . . . dup(2)
dup2: duplicate a descriptor. . . . . dup2(3N)
aliens: alien invaders attack the earth. . . . . aliens(6)
echo: echo arguments. . . . . echo(1)
echo: echo arguments. . . . echo(1)
hosts. ping: send ICMP ECHO_REQUEST packets to network ping(1M)
floating-point number to string. ecvt, fcvt, gcvt: convert . . . ecvt(3C)
ed, red: text editor. . . . . ed(1)
end, etext, edata: last locations in program. end(3C)
ex, edit: text editor. . . . . ex(1)
vipw: edit the password file. . . . . vipw(1M)
sact: print current SCCS file editing activity. . . . . sact(1)
ed, red: text editor. . . . . ed(1)
ex, edit: text editor. . . . . ex(1)
ld: link editor for common object files. ld(1)
a.out: common assembler and link editor output. . . . . a.out(4)
sed: stream editor. . . . . sed(1)
screen-oriented (visual) display editor. vi, view, vedit: . . . vi(1)
whoami: print effective current user ID. . . whoami(1)
setregid: set real and effective group ID. . . . . setregid(2)
effective user, real group, and effective group IDs. /real user, getuid(2)
setreuid: set real and effective user ID's. . . . . setreuid(2)
/getgid, getegid: get real user, effective user, real group, and/ getuid(2)
efl: Extended Fortran Language. efl(1)
fsplit: split f77 or efl files. . . . . fsplit(1)
pattern. grep, egrep, fgrep: search a file for a grep(1)

eject: eject diskette from drive. . . eject(1)
eject: eject diskette from drive. eject(1)
insque, remque: insert/remove element from a queue. . . . insque(3N)
input. soelim: eliminate .so's from nroff . . soelim(1)
LP printers. enable, disable: enable/disable enable(1)
accounting. acct: enable or disable process . . acct(2)
enable, disable: enable/disable LP printers. . . enable(1)
transmission/ uuencode, uudecode: encode/decode a binary file for uuencode(1C)
crypt: encode/decode. . . . . crypt(1)
crypt, setkey, encrypt: generate DES encryption. crypt(3C)
setkey, encrypt: generate DES encryption. crypt, . . . . . crypt(3C)
makekey: generate encryption key. . . . . makekey(1)
in program. end, etext, edata: last locations end(3C)
sccs: front end for the SCCS subsystem. . sccs(1)
/getgrgid, getgrnam, setgrent, endgrent, fgetgrent: obtain group/ getgrent(3C)
/gethostbyname, sethostent, endhostent: get network host/ . gethostent(3N)
setmntent, getmntent, addmntent, endmntent, hasmntopt: get file/ getmntent(3)
/getnetbyname, setnetent, endnetent: get network entry. . getnetent(3N)
group/ getnetgrent, setnetgrent, endnetgrent, innetgr: get network getnetgrent(3N)
socket: create an endpoint for communication. . socket(2N)
/getprotobyname, setprotoent, endprotoent: get protocol entry. getprotoent(3N)
getptabent, addptabent, endptabent, setptabent,/ . . . getptabent(3)
/getpwuid, getpwnam, setpwent, endpwent, fgetpwent: get password/ getpwent(3C)
/getservbyname, setservent, endservent: get service entry. . getservent(3N)
/getutline, pututline, setutent, endutent, utmpname: access utmp/ getut(3C)
convert Arabic numerals to English. number: . . . . . number(6)
format of disk partition map entries. dpme: . . . . . dpme(4)
nlist: get entries from name list. . . . nlist(3C)
linenum: line number entries in a common object file. linenum(4)
getdirentries: gets directory entries in a file system/ . . . getdirentries(2)
man: print entries in this manual. . . . . man(1)
man: macros for formatting entries in this manual. . . . . man(5)
/ldlitem: manipulate line number entries of a common object file/ ldlread(3X)
/ldnlseek: seek to line number entries of a section of a common/ ldlseek(3X)
/ldnrseek: seek to relocation entries of a section of a common/ ldrseek(3X)
chfn: change finger entry. . . . . chfn(1)
utmp, wtmp: utmp and wtmp entry formats. . . . . utmp(4)
/fgetgrent: obtain group file entry from a group file. . . . getgrent(3C)
endhostent: get network host entry. /sethostent, . . . . . gethostent(3N)
get file system descriptor file entry. /endmntent, hasmntopt: getmntent(3)
setnetent, endnetent: get network entry. /getnetbyname, . . . . getnetent(3N)
innetgr: get network group entry. /setnetgrent, endnetgrent, getnetgrent(3N)
endprotoent: get protocol entry. /setprotoent, . . . . . getprotoent(3N)
get partition table file entry. /setptabent, numbptabent: getptabent(3)
fgetpwent: get password file entry. /setpwent, endpwent, . getpwent(3C)
endservent: get service entry. /setservent, . . . . . getservent(3N)
utmpname: access utmp file entry. /setutent, endutent, . . getut(3C)
/the index of a symbol table entry of a common object file. ldtbindex(3X)
/read an indexed symbol table entry of a common object file. ldtbread(3X)
putpwent: write password file entry. . . . . putpwent(3C)
unlink: remove directory entry. . . . . unlink(2)

execution. env: set environment for command env(1)
environ: user environment. . . environ(5)
profile: setting up an environment at login time. . . profile(4)
environ: user environment. . . . . environ(5)
execution. env: set environment for command . . env(1)
A/UX kernel from the standalone environment. launch: launch an launch(8)
getenv: return value for environment name. . . . . getenv(3C)
printenv: print the current environment. . . . . printenv(1)
putenv: change or add value to environment. . . . . putenv(3C)
interpreter for the standalone environment. sash: a command sash(8)
getenv: return Fortran environment variable. . . . . getenv(3F)
special character definitions for eqn and neqn. eqnchar: . . . eqnchar(5)
remove nroff/troff, tbl, and eqn constructs. deroff: . . . deroff(1)
troff. eqn: format mathematical text for eqn(1)
definitions for eqn and neqn. eqnchar: special character . . eqnchar(5)
mrand48, jrand48,/ drand48, erand48, lrand48, nrand48, . . drand48(3C)
complementary error function. erf, erfc: error function and . erf(3M)
complementary error/ erf, erfc: error function and . . . erf(3M)
from dumpfs. errdead: extract error records . errdead(1M)
errdemon: error-logging daemon. errdemon(1M)
errfile: error-log file format. . errfile(4)
system error messages. perror, errno, sys_errlist, sys_nerr: . . perror(3C)
error: error-logging interface. . error(7)
error function. erf, erfc: error function and complementary erf(3M)
error function and complementary error function. erf, erfc: . . . erf(3M)
source. mkstr: create an error message file by massaging C mkstr(1)
sys_errlist, sys_nerr: system error messages. perror, errno, perror(3C)
introduction to system calls and error numbers. intro: . . . . intro(2)
errdead: extract error records from dumpfs. . . errdead(1M)
matherr: error-handling function. . . . matherr(3M)
errfile: error-log file format. . . . . errfile(4)
errdemon: error-logging daemon. . . . errdemon(1M)
errstop: terminate the error-logging daemon. . . . errstop(1M)
error: error-logging interface. . . . error(7)
errpt: process a report of logged errors. . . . . errpt(1M)
/on/off the reporting of extended errors in the specified device. . exterr(1M)
spellin, hashcheck: find spelling errors. spell, hashmake, . . . spell(1)
errors. errpt: process a report of logged errpt(1M)
error-logging daemon. errstop: terminate the . . . . errstop(1M)
autorobots: escape from the automatic robots. autorobots(6)
robots: escape from the robots. . . . robots(6)
dialup: modem escape sequence file. . . . . dialup(4)
chase: try to escape the killer robots. . . . chase(6)
administration. escher: autorecovery . . . . escher(1M)
line connection. dial: establish an out-going terminal dial(3C)
tty_add, tty_kill: modify the /etc/inittab file. . . . . tty_add(1M)
program. end, etext, edata: last locations in . end(3C)
Ethernet address.. etheraddr: read an interface's . etheraddr(1M)
etheraddr: read an interface's Ethernet address.. . . . . etheraddr(1M)
ae: 3Com 10 Mb/s Ethernet interface. . . . . ae(5)
dual: 3Com Ethernet interface diagnostic. . dual(1M)

hypot: Euclidean distance function. . hypot(3M)
expression. expr: evaluate arguments as an . . expr(1)
test: condition evaluation command. . . . . test(1)
ex, edit: text editor. . . . . ex(1)
lpq: spool queue examination program. . . . . lpq(1)
reading or/ locking: provide exclusive file regions for . . locking(2)
execl, execv, execle, execve,. . exec(2)
execl, execv, execle, execve,. . . . . exec(2)
application. launch: execute a Macintosh binary . launch(1)
regcmp, regex: compile and execute a regular expression. . regcmp(3X)
construct argument list(s) and execute command. xargs: . . xargs(1)
at, batch: execute commands at a later time. at(1)
env: set environment for command execution. . . . . env(1)
sleep: suspend execution for an interval. . . sleep(1)
sleep: suspend execution for interval. . . . . sleep(3C)
monitor: prepare execution profile. . . . . monitor(3C)
rexecd: remote execution server. . . . . rexecd(1M)
profil: execution time profile. . . . profil(2)
uux: UNIX-to-UNIX system command execution. . . . . uux(1C)
execl, execv, execle, execve,. . . . exec(2)
execl, execv, execle, execve,. . . . . exec(2)
create a new file or rewrite an existing one. creat: . . . . . creat(2)
exit, _exit: terminate process. . exit(2)
exit, _exit: terminate process. . . . exit(2)
exponential intrinsic function. exp, dexp, cexp: Fortran . . . exp(3F)
exponential, logarithm, power,/ exp, log, log10, pow, sqrt: . . exp(3M)
pack, pcat, unpack: compress and expand files. . . . . pack(1)
versa. expand, unexpand: expand tabs to spaces, and vice expand(1)
spaces, and vice versa. expand, unexpand: expand tabs to expand(1)
thesaurus for diction. diction, explain: print wordy sentences; diction(1)
/dble, cmplx, dcmplx, ichar, char: explicit Fortran type conversion. ftype(3F)
exp, dexp, cexp: Fortran exponential intrinsic function. exp(3F)
exp, log, log10, pow, sqrt: exponential, logarithm, power,/ exp(3M)
exports: NFS file systems being exported. . . . . exports(4)
exported. exports: NFS file systems being exports(4)
expression. expr: evaluate arguments as an expr(1)
routines. regexp: regular expression compile and match regexp(5)
regcmp: regular expression compile. . . . . regcmp(1)
expr: evaluate arguments as an expression. . . . . expr(1)
compile and execute a regular expression. regcmp, regex: . regcmp(3X)
addbib: create or extend bibliographic database. addbib(1)
/turn on/off the reporting of extended errors in the specified/ exterr(1M)
efl: Extended Fortran Language. . efl(1)
greek: graphics for the extended TTY-37 type-box. . greek(5)
xdr: library routines for external data representation. . xdr(3N)
of extended errors in the/ exterr: turn on/off the reporting exterr(1M)
dumpfs. errdead: extract error records from . . errdead(1M)
to implement shared/ xstr: extract strings from C programs xstr(1)
f77: Fortran 77 compiler. . . f77(1)
fsplit: split f77 or efl files. . . . . fsplit(1)
absolute/ floor, ceil, fmod, fabs: floor, ceiling, remainder, floor(3M)

introduction to miscellaneous facilities. intro: . . . . . intro(5)
BSD-compatible software signal facilities. sigvec: optional . . sigvec(2)
report interprocess communication facilities status. ipcs: . . . . ipcs(1)
factor: factor a number. . . . . factor(1)
factor: factor a number. . . . factor(1)
provide drill in number facts. arithmetic: . . . . . arithmetic(6)
pstat: print system facts. . . . . pstat(1M)
true, false: provide truth values. . . true(1)
inet: Internet protocol family. . . . . inet(5f)
data in a machine independent fashion. /access long integer . sputl(3X)
finc: fast incremental backup. . . . finc(1M)
/calloc, mallopt, mallinfo: fast main memory allocator. . malloc(3X)
abort: generate an IOT fault. . . . . abort(3C)
a file. chown, fchown: change owner and group of chown(2)
stream. fclose, fflush: close or flush a . fclose(3S)
fcntl: file control. . . . . fcntl(2)
fcntl: file control options. . . fcntl(5)
floating-point number to/ ecvt, fcvt, gcvt: convert . . . . . ecvt(3C)
fd: floppy disk drive interface. fd(7)
fopen, freopen, fdopen: open a stream. . . . fopen(3S)
status inquiries. ferror, feof, clearerr, fileno: stream . ferror(3S)
stream status inquiries. ferror, feof, clearerr, fileno: . ferror(3S)
nextkey: data base/ dbminit, fetch, store, delete, firstkey, . dbm(3X)
head: give first few lines. . . . . head(1)
statistics for a file system. ff: list file names and . . . . ff(1M)
fclose, fflush: close or flush a stream. fclose(3S)
operations. bcopy, bcmp, bzero, ffs: bit and byte string . . . . bstring(3)
word from a/ getc, getchar, fgetc, getw: get character or . getc(3S)
/getgrnam, setgrent, endgrent, fgetgrent: obtain group file/ . getgrent(3C)
/getpwnam, setpwent, endpwent, fgetpwent: get password file/ . getpwent(3C)
stream. gets, fgets: get a string from a . . . gets(3S)
pattern. grep, egrep, fgrep: search a file for a . . . grep(1)
cut: cut out selected fields of each line of a file. . . cut(1)
times. utime: set file access and modification . utime(2)
ldfcn: common object file access routines. . . . . ldfcn(3X)
determine accessibility of a file. access: . . . . . access(2)
diff: differential file and directory comparator. . diff(1)
tar: tape file archiver. . . . . tar(1)
cpio: copy file archives in and out. . . . cpio(1)
mkstr: create an error message file by massaging C source. . mkstr(1)
pwck, grpck: password/group file checkers. . . . . pwck(1M)
chmod: change mode of file. . . . . chmod(2)
change owner and group of a file. chown, fchown: . . . . chown(2)
colrm: remove columns from a file. . . . . colrm(1)
magic: file command's magic number file. magic(4)
diff3: 3-way differential file comparison. . . . . diff3(1)
fcntl: file control. . . . . fcntl(2)
fcntl: file control options. . . . . fcntl(5)
conv: object file converter. . . . . conv(1)
rcp: remote file copy. . . . . rcp(1C)
public UNIX-to-UNIX system file copy. uuto, uupick: . . . uuto(1C)

core: format of core image file. . . . . core(4)
umask: set and get file creation mask. . . . . umask(2)
selected fields of each line of a file. cut: cut out . . . . . cut(1)
dd: convert and copy a file. . . . . dd(1)
make a delta (change) to an SCCS file. delta: . . . . . delta(1)
derez: decompiles a resource file. . . . . derez(1)
close: close a file descriptor. . . . . close(2)
file: determine file type. . . . file(1)
dialup: modem escape sequence file. . . . . dialup(4)
rdist: remote file distribution program. . . rdist(1)
dump selected parts of an object file. dump: . . . . . dump(1)
sact: print current SCCS file editing activity. . . . . sact(1)
/endgrent, fgetgrent: obtain group file entry from a group file. . . getgrent(3C)
get file system descriptor file entry. /hasmntopt: . . . getmntent(3)
numbptabent: get partition table file entry. /setptabent, . . . . getptabent(3)
endpwent, fgetpwent: get password file entry. /getpwnam, setpwent, getpwent(3C)
endutent, utmpname: access utmp file entry. /pututline, setutent, getut(3C)
putpwent: write password file entry. . . . . putpwent(3C)
an advisory lock on an open file. flock: apply or remove . flock(2)
ctags: maintain a tags file for a C program. . . . . ctags(1)
grep, egrep, fgrep: search a file for a pattern. . . . . grep(1)
ldaopen: open a common object file for reading. ldopen, . . . ldopen(3X)
aliases: aliases file for sendmail. . . . . aliases(4)
/uudecode: encode/decode a binary file for transmission via mail. . uuencode(1C)
acct: per-process accounting file format. . . . . acct(4)
Adobe PostScript font metrics file format. afm: . . . . . afm(7)
ar: common archive file format. . . . . ar(4)
errfile: error-log file format. . . . . errfile(4)
postscript: print file format. . . . . postscript(4)
intro: introduction to file formats. . . . . intro(4)
fpr: print Fortran file. . . . . fpr(1)
on character frequencies in a file. freq: report . . . . . freq(1)
take: takes a file from a remote machine. . take(1C)
number entries of a common object file function. /manipulate line ldlread(3X)
get: get a version of an SCCS file. . . . . get(1)
group file entry from a group file. /fgetgrent: obtain . . . . getgrent(3C)
group: group file. . . . . group(4)
nfs_getfh: get a file handle. . . . . nfs_getfh(2)
files. filehdr: file header for common object filehdr(4)
file. ldfhread: read the file header of a common object ldfhread(3X)
ldohseek: seek to the optional file header of a common object/ ldohseek(3X)
which: locate a program file including aliases and paths. which(1)
split: split a file into pieces. . . . . split(1)
issue: issue identification file. . . . . issue(4)
header of a member of an archive file. ldahread: read the archive ldahread(3X)
ldaclose: close a common object file. ldclose, . . . . . ldclose(3X)
file header of a common object file. ldfhread: read the . . . ldfhread(3X)
retrieve symbol name for object file. ldgetname: . . . . . ldgetname(3X)
of a section of a common object file. /to line number entries . ldlseek(3X)
file header of a common object file. /seek to the optional . . ldohseek(3X)
of a section of a common object file. /seek to relocation entries ldrseek(3X)

section header of a common object file. /read an indexed/named . ldshread(3X)
section of a common object file. /seek to an indexed/named ldsseek(3X)
table entry of a common object file. /the index of a symbol . . ldtbindex(3X)
table entry of a common object file. /read an indexed symbol . ldtbread(3X)
symbol table of a common object file. ldtbseek: seek to the . . ldtbseek(3X)
number entries in a common object file. linenum: line . . . . . linenum(4)
link: link to a file. . . . . link(2)
file command's magic number file. magic: . . . . . magic(4)
makedbm: make a yellow pages dbm file. . . . . makedbm(1M)
mkdir: make a directory file. . . . . mkdir(2)
mknod: build special file. . . . . mknod(1M)
or a special or ordinary file. mknod: make a directory, mknod(2)
file system. ff: list file names and statistics for a . ff(1M)
data base for the mail aliases file. newaliases: rebuild the . newaliases(1M)
change the format of a text file. newform: . . . . . newform(1)
print name list of common object file. nm: . . . . . nm(1)
null: the null file. . . . . null(7)
/find the slot in the utmp file of the current user. . . . ttyslot(3C)
put: puts a file onto a remote machine. . put(1C)
fuser: identify processes using a file or file structure. . . . . fuser(1M)
creat: create a new file or rewrite an existing one. creat(2)
passwd: password file. . . . . passwd(4)
files or subsequent lines of one file. /same lines of several . . paste(1)
viewing. more: file perusal filter for CRT . . more(1)
terminals. pg: file perusal filter for soft-copy pg(1)
rewind, ftell: reposition a file pointer in a stream. fseek, fseek(3S)
lseek: move read/write file pointer. . . . . lseek(2)
prs: print an SCCS file. . . . . prs(1)
ptab: partition table file. . . . . ptab(4)
S-records from downloading into a file. /translates Motorola . . rcvhex(1)
read, readv: read from file. . . . . read(2)
locking: provide exclusive file regions for reading or/ . . locking(2)
information for a common object file. reloc: relocation . . . . reloc(4)
remote: remote host description file. . . . . remote(4)
rename: change the name of a file. . . . . rename(2)
resolver: resolver configuration file. . . . . resolver(4)
rev: reverse lines of a file. . . . . rev(1)
remove a delta from an SCCS file. rmdel: . . . . . rmdel(1)
rmdir: remove a directory file. . . . . rmdir(2)
bfs: big file scanner. . . . . bfs(1)
compare two versions of an SCCS file. sccsdiff: . . . . . sccsdiff(1)
sccsfile: format of an SCCS file. . . . . sccsfile(4)
header for a common object file. scnhdr: section . . . . scnhdr(4)
creator of a Macintosh resource file. settc: set the type and . . settc(1)
stat, fstat, lstat: get file status. . . . . stat(2)
in an object, or other binary file. /find the printable strings strings(1)
number information from an object file. /strip symbol and line . . strip(1)
processes using a file or file structure. fuser: identify . fuser(1M)
checksum and block count of a file. sum: print . . . . . sum(1)
syms: common object file symbol table format. . . . syms(4)
symlink: make symbolic link to a file. . . . . symlink(2)

interactive repair. fsck: file system consistency check and fsck(1M)
fsdb: file system debugger. . . . . fsdb(1M)
entry. /endmntent, hasmntopt: get file system descriptor file . . getmntent(3)
mfs: read a Macintosh flat file system disk. . . . . mfs(1)
network. rdumpfs: file system dump across the . rdumpfs(1M)
network. rrestore: restore a file system dump across the . rrestore(1M)
dumpfs: incremental file system dump. . . . . dumpfs(1M)
file names and statistics for a file system. ff: list . . . . . ff(1M)
fsmount: mount an NFS file system. . . . . fsmount(2)
/gets directory entries in a file system independent format getdirentries(2)
mkfs1b: construct a file system. . . . . mkfs1b(1M)
mkfs: construct a file system. . . . . mkfs(1M)
mount: mount a file system. . . . . mount(3)
autorecovery: standalone file system repair. . . . . autorecovery(8)
restore: incremental file system restore. . . . . restore(1M)
nfsstat: Network File System statistics. . . . nfsstat(1M)
statfs: get file system statistics. . . . statfs(2)
ustat: get file system statistics. . . . ustat(2)
mtab: mounted file system table. . . . . mtab(4)
rmtab: remotely mounted file system table. . . . . rmtab(4)
umount: unmount a file system. . . . . umount(2)
umount: unmount a file system. . . . . umount(3)
unmount: remove a file system. . . . . unmount(2)
exports: NFS file systems being exported. . exports(4)
time. dcopy: copy file systems for optimal access dcopy(1M)
fstab: static information about file systems. . . . . fstab(4)
mount, umount: mount and dismount file systems. . . . . mount(1M)
volcopy, labelit: copy file systems with label checking. volcopy(1M)
tail: deliver the last part of a file. . . . . tail(1)
term: format of compiled term file.. . . . . term(4)
tmpfile: create a temporary file. . . . . tmpfile(3S)
create a name for a temporary file. tmpnam, tempnam: . . . tmpnam(3S)
uusend: send a file to a remote host. . . . uusend(1C)
truncate, ftruncate: truncate a file to a specified length. . . truncate(2)
and modification times of a file. touch: update access . . touch(1)
kermit: kermit file transfer. . . . . kermit(1C)
ftp: ARPANET file transfer program. . . . ftp(1N)
tftp: trivial file transfer program. . . . tftp(1C)
ftpd: DARPA Internet File Transfer Protocol server. . ftpd(1M)
tftpd: DARPA Trivial File Transfer Protocol server. . tftpd(1M)
ftw: walk a file tree. . . . . ftw(3C)
tty_kill: modify the /etc/inittab file. tty_add, . . . . . tty_add(1M)
file: determine file type. . . . . file(1)
undo a previous get of an SCCS file. unget: . . . . . unget(1)
uniq: report repeated lines in a file. . . . . uniq(1)
val: validate SCCS file. . . . . val(1)
vipw: edit the password file. . . . . vipw(1M)
write, writev: write on a file. . . . . write(2)
modifying yellow pages password file. yppasswdd: server for . . yppasswdd(1M)
object files. filehdr: file header for common filehdr(4)
ctermid: generate filename for terminal. . . . . ctermid(3S)

mktemp: make a unique filename. . . . . mktemp(3C)
ferror, feof, clearerr, fileno: stream status inquiries. ferror(3S)
and print process accounting file(s). acctcom: search . . . acctcom(1M)
merge or add total accounting files. acctmerg: . . . . . acctmerg(1M)
admin: create and administer SCCS files. . . . . admin(1)
ccat: compress and uncompress files, and cat them. /uncompact, compact(1)
a.out header for common object files. aouthdr.h: . . . . . aouthdr(4)
bdiff: diff large files. . . . . bdiff(1)
updater: update files between two machines. . updater(1)
cat: concatenate and print files. . . . . cat(1)
checknr: check nroff/troff files. . . . . checknr(1)
cmp: compare two files. . . . . cmp(1)
reject lines common to two sorted files. comm: select or . . . . comm(1)
cp: copy files. . . . . cp(1)
diffmk: mark differences between files. . . . . diffmk(1)
file header for common object files. filehdr: . . . . . filehdr(4)
find: find files. . . . . find(1)
troff. font: description files for device-independent . font(5)
frec: recover files from a backup tape. . . . frec(1M)
format specification in text files. fspec: . . . . . fspec(4)
fsplit: split f77 or efl files. . . . . fsplit(1)
hex: translates object files. . . . . hex(1)
cpset: install object files in binary directories. . . cpset(1M)
/sum and count characters in the files in the given directories. . sumdir(1)
disk. fsync: synchronize a file's in-core state with that on fsync(2)
character map description files. /format of iwprep(1) . . iwmap(4)
iwprep: prepare troff description files. . . . . iwprep(1)
ld: link editor for common object files. . . . . ld(1)
lockf: record locking on files. . . . . lockf(3C)
cross-reference listing of macro files. macref: produce . . . . macref(1)
mv: move or rename files. . . . . mv(1)
rm, rmdir: remove files or directories. . . . . rm(1)
/merge same lines of several files or subsequent lines of one/ paste(1)
pcat, unpack: compress and expand files. pack, . . . . . pack(1)
pr: print files. . . . . pr(1)
section sizes of common object files. size: print . . . . . size(1)
sort: sort and/or merge files. . . . . sort(1)
reports version number of files. version: . . . . . version(1)
what: identify SCCS files. . . . . what(1)
II troff postprocessor filter. daiw: Apple ImageWriter daiw(1)
more: file perusal filter for CRT viewing. . . . more(1)
pg: file perusal filter for soft-copy terminals. . pg(1)
greek: select terminal filter. . . . . greek(1)
iw2: Apple ImageWriter print filter. . . . . iw2(1)
nl: line numbering filter. . . . . nl(1)
previewing. colcrt: filter nroff output for terminal colcrt(1)
col: filter reverse linefeeds. . . . col(1)
/psinterface: TranScript spooler filters for PostScript printers. . transcript(1M)
tplot: graphics filters. . . . . tplot(1G)
finc: fast incremental backup. . finc(1M)
references in documents. refer: find and insert literature . . . refer(1)

find: find files. . . . . find(1)
find: find files. . . . . find(1)
hyphen: find hyphenated words. . . . hyphen(1)
ttyname, isatty: find name of a terminal. . . . ttyname(3C)
object library. lorder: find ordering relation for an . lorder(1)
index for a bibliography, find references in a/ /inverted . lookbib(1)
hashmake, spellin, hashcheck: find spelling errors. spell, . . spell(1)
object, or other binary/ strings: find the printable strings in an strings(1)
the current user. ttyslot: find the slot in the utmp file of ttyslot(3C)
chfn: change finger entry. . . . . chfn(1)
program. finger: user information lookup finger(1)
fold: fold long lines for finite-width output device. . . fold(1)
dbminit, fetch, store, delete, firstkey, nextkey: data base/ . dbm(3X)
fish: play "Go Fish". . . . . fish(6)
fish: play "Go Fish". . . . . fish(6)
tee: pipe fitting. . . . . tee(1)
mfs: read a Macintosh flat file system disk. . . . . mfs(1)
ichar,/ int, ifix, idint, real, float, sngl, dble, cmplx, dcmplx, ftype(3F)
atof: convert ASCII string to floating-point number. . . . atof(3C)
ecvt, fcvt, gcvt: convert floating-point number to string. ecvt(3C)
ldexp, modf: manipulate parts of floating-point numbers. frexp, frexp(3C)
advisory lock on an open file. flock: apply or remove an . . flock(2)
ceiling, remainder, absolute/ floor, ceil, fmod, fabs: floor, . floor(3M)
floor, ceil, fmod, fabs: floor, ceiling, remainder,/ . . floor(3M)
fd: floppy disk drive interface. . . fd(7)
cflow: generate C flowgraph. . . . . cflow(1)
fclose, fflush: close or flush a stream. . . . . fclose(3S)
remainder, absolute/ floor, ceil, fmod, fabs: floor, ceiling, . . floor(3M)
fmt: simple text formatter. . . fmt(1)
finite-width output device. fold: fold long lines for . . . fold(1)
output device. fold: fold long lines for finite-width fold(1)
device-independent troff. font: description files for . . font(5)
afm: Adobe PostScript font metrics file format. . . . afm(7)
stream. fopen, freopen, fdopen: open a fopen(3S)
map. yppush: force propagation of a changed YP yppush(1M)
fork: create a new process. . . fork(2)
diskformat: format a disk. . . . . diskformat(1M)
acct: per-process accounting file format. . . . . acct(4)
PostScript font metrics file format. afm: Adobe . . . . afm(7)
ar: common archive file format. . . . . ar(4)
indent: indent and format C program source. . . indent(1)
cml: configuration master list format. . . . . cml(4)
dumpfs: incremental dump format. . . . . dumpfs(4)
errfile: error-log file format. . . . . errfile(4)
in a file system independent format /gets directory entries . getdirentries(2)
nroff. neqn: format mathematical text for . neqn(1)
troff. eqn: format mathematical text for . eqn(1)
inode: format of a System V inode. . inode(4)
volume. fs: format of a System V system . fs(4)
newform: change the format of a text file. . . . newform(1)
sccsfile: format of an SCCS file. . . . sccsfile(4)

bzb: format of Block Zero Blocks. . bzb(4)
term: format of compiled term file.. . term(4)
core: format of core image file. . . core(4)
cpio: format of cpio archive. . . . cpio(4)
entries. dpme: format of disk partition map . dpme(4)
description files. iwmap: format of iwprep(1) character map iwmap(4)
dir: format of System V directories. dir(4)
postscript: print file format. . . . . postscript(4)
to POSTSCRIPT format. /intermediate format . psdit(1)
files. fspec: format specification in text . . fspec(4)
common object file symbol table format. syms: . . . . . syms(4)
tbl: format tables for nroff or troff. tbl(1)
psdit: convert troff intermediate format to/ . . . . . psdit(1)
intro: introduction to file formats. . . . . intro(4)
utmp, wtmp: utmp and wtmp entry formats. . . . . utmp(4)
scanf, fscanf, sscanf: convert formatted input. . . . . scanf(3S)
/vfprintf, vsprintf: print formatted output of a varargs/ vprintf(3S)
printf, fprintf, sprintf: print formatted output. . . . . printf(3S)
checkmm: check documents formatted with the mm macros. checkmm(1)
mm: prints documents formatted with the mm macros. mm(1)
fmt: simple text formatter. . . . . fmt(1)
mptx: the macro package for formatting a permuted index. . mptx(5)
otroff: text formatting and typesetting. . . otroff(1)
troff: text formatting and typesetting. . . troff(1)
mm: macro package for formatting documents. . . . mm(5)
manual. man: macros for formatting entries in this . . . man(5)
nroff: text formatting language. . . . . nroff(1)
ms: text formatting macros. . . . . ms(5)
f77: Fortran 77 compiler. . . . . f77(1)
abs, iabs, dabs, cabs, zabs: Fortran absolute value. . . . abs(3F)
system signal. signal: specify Fortran action on receipt of a . signal(3F)
function. acos, dacos: Fortran arccosine intrinsic . . acos(3F)
function. asin, dasin: Fortran arcsine intrinsic . . . asin(3F)
function. atan2, datan2: Fortran arctangent intrinsic . . atan2(3F)
function. atan, datan: Fortran arctangent intrinsic . . atan(3F)
or, xor, not, lshift, rshift: Fortran bitwise boolean/ and, . bool(3F)
getarg: return Fortran command-line argument. getarg(3F)
intrinsic/ log10, alog10, dlog10: Fortran common logarithm . . log10(3F)
intrinsic/ conjg, dconjg: Fortran complex conjugate . . conjg(3F)
function. cos, dcos, ccos: Fortran cosine intrinsic . . . cos(3F)
intrinsic function. dprod: Fortran double precision product dprod(3F)
getenv: return Fortran environment variable. getenv(3F)
function. exp, dexp, cexp: Fortran exponential intrinsic . exp(3F)
fpr: print Fortran file. . . . . fpr(1)
intrinsic function. cosh, dcosh: Fortran hyperbolic cosine . . cosh(3F)
function. sinh, dsinh: Fortran hyperbolic sine intrinsic sinh(3F)
intrinsic function. tanh, dtanh: Fortran hyperbolic tangent . . tanh(3F)
argument. aimag, dimag: Fortran imaginary part of complex aimag(3F)
function. aint, dint: Fortran integer part intrinsic . aint(3F)
efl: Extended Fortran Language. . . . . efl(1)
/max0, amax0, max1, amax1, dmax1: Fortran maximum-value functions. max(3F)

/min0, amin0, min1, amin1, dmin1: Fortran minimum-value functions. min(3F)
intrinsic/ log, alog, dlog, clog: Fortran natural logarithm . . log(3F)
anint, dnint, nint, idnint: Fortran nearest integer/ . . . round(3F)
intrinsic/ dim, ddim, idim: Fortran positive difference . . dim(3F)
abort: terminate Fortran program. . . . . abort(3F)
functions. mod, amod, dmod: Fortran remaindering intrinsic mod(3F)
sin, dsin, csin: Fortran sine intrinsic function. sin(3F)
function. sqrt, dsqrt, csqrt: Fortran square root intrinsic . sqrt(3F)
len: return length of Fortran string. . . . . len(3F)
index: return location of Fortran substring. . . . . index(3F)
issue a shell command from Fortran. system: . . . . . system(3F)
function. tan, dtan: Fortran tangent intrinsic . . . tan(3F)
mclock: return Fortran time accounting. . . . mclock(3F)
intrinsic/ sign, isign, dsign: Fortran transfer-of-sign . . . sign(3F)
dcmplx, ichar, char: explicit Fortran type conversion. /cmplx, ftype(3F)
generator. irand, srand, rand: Fortran uniform random-number rand(3F)
hopefully interesting, adage. fortune: print a random, . . . fortune(6)
fpr: print Fortran file. . . . . fpr(1)
output. printf, fprintf, sprintf: print formatted printf(3S)
word on a stream. putc, putchar, fputc, putw: put character or . putc(3S)
puts, fputs: put a string on a stream. puts(3S)
input/output. fread, fwrite: binary . . . . fread(3S)
tape. frec: recover files from a backup frec(1M)
df: report number of free disk blocks. . . . . df(1)
main memory allocator. malloc, free, realloc, calloc, cfree: . . malloc(3C)
mallinfo: fast main/ malloc, free, realloc, calloc, mallopt, . malloc(3X)
fopen, freopen, fdopen: open a stream. fopen(3S)
frequencies in a file. freq: report on character . . . freq(1)
freq: report on character frequencies in a file. . . . . freq(1)
parts of floating-point numbers. frexp, ldexp, modf: manipulate frexp(3C)
if mail arrives and who it is from. biff: be notified . . . . biff(1)
from: who is my mail from?. . . . . from(1)
from: who is my mail from?. . from(1)
sccs: front end for the SCCS subsystem. sccs(1)
volume. fs: format of a System V system fs(4)
input. scanf, fscanf, sscanf: convert formatted scanf(3S)
check and interactive repair. fsck: file system consistency . fsck(1M)
fsdb: file system debugger. . . fsdb(1M)
a file pointer in a stream. fseek, rewind, ftell: reposition fseek(3S)
generation numbers. fsirand: install random inode . fsirand(1M)
system. fsmount: mount an NFS file . fsmount(2)
text files. fspec: format specification in . fspec(4)
fsplit: split f77 or efl files. . . fsplit(1)
file systems. fstab: static information about fstab(4)
stat, fstat, lstat: get file status. . . stat(2)
in-core state with that on disk. fsync: synchronize a file's . . fsync(2)
in a stream. fseek, rewind, ftell: reposition a file pointer . fseek(3S)
communication package. ftok: standard interprocess . . ftok(3C)
program. ftp: ARPANET file transfer . ftp(1N)
Transfer Protocol server. ftpd: DARPA Internet File . . ftpd(1M)
specified length. truncate, ftruncate: truncate a file to a . truncate(2)

ftw: walk a file tree. . . . . ftw(3C)
shutdown: shut down part of a full-duplex connection. . . . shutdown(2N)
Fortran arccosine intrinsic function. acos, dacos: . . . . acos(3F)
Fortran integer part intrinsic function. aint, dint: . . . . . aint(3F)
function. erf, erfc: error function and complementary error erf(3M)
dasin: Fortran arcsine intrinsic function. asin, . . . . . asin(3F)
Fortran arctangent intrinsic function. atan2, datan2: . . . atan2(3F)
Fortran arctangent intrinsic function. atan, datan: . . . . atan(3F)
complex conjugate intrinsic function. conjg, dconjg: Fortran conjg(3F)
ccos: Fortran cosine intrinsic function. cos, dcos, . . . . cos(3F)
hyperbolic cosine intrinsic function. cosh, dcosh: Fortran cosh(3F)
precision product intrinsic function. dprod: Fortran double dprod(3F)
function and complementary error function. erf, erfc: error . . . erf(3M)
Fortran exponential intrinsic function. exp, dexp, cexp: . . exp(3F)
gamma: log gamma function. . . . . gamma(3M)
hypot: Euclidean distance function. . . . . hypot(3M)
entries of a common object file function. /manipulate line number ldlread(3X)
common logarithm intrinsic function. /dlog10: Fortran . . log10(3F)
natural logarithm intrinsic function. /dlog, clog: Fortran . log(3F)
matherr: error-handling function. . . . . matherr(3M)
prof: profile within a function. . . . . prof(5)
transfer-of-sign intrinsic function. /isign, dsign: Fortran sign(3F)
csin: Fortran sine intrinsic function. sin, dsin, . . . . . sin(3F)
Fortran hyperbolic sine intrinsic function. sinh, dsinh: . . . . sinh(3F)
Fortran square root intrinsic function. sqrt, dsqrt, csqrt: . . sqrt(3F)
dtan: Fortran tangent intrinsic function. tan, . . . . . tan(3F)
hyperbolic tangent intrinsic function. tanh, dtanh: Fortran tanh(3F)
math: math functions and constants. . . . math(5)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)
rshift: Fortran bitwise boolean functions. /or, xor, not, lshift, bool(3F)
positive difference intrinsic functions. /ddim, idim: Fortran dim(3F)
logarithm, power, square root functions. /sqrt: exponential, . exp(3M)
remainder, absolute value functions. /fabs: floor, ceiling, floor(3M)
llt: string comparision intrinsic functions. lge, lgt, lle, . . . lge(3F)
dmax1: Fortran maximum-value functions. /amax0, max1, amax1, max(3F)
dmin1: Fortran minimum-value functions. /amin0, min1, amin1, min(3F)
Fortran remaindering intrinsic functions. mod, amod, dmod: mod(3F)
300, 300s: handle special functions of DASI 300 and 300s/ 300(1)
terminal. 450: handle special functions of the DASI 450 . . 450(1)
idnint: Fortran nearest integer functions. anint, dnint, nint, . round(3F)
sinh, cosh, tanh: hyperbolic functions. . . . . sinh(3M)
slots: ROM library functions. . . . . slots(3X)
acos, atan, atan2: trigonometric functions. sin, cos, tan, asin, . trig(3M)
curses5.0: BSD-style screen functions with "optimal" cursor/ curses5.0(3X)
file or file structure. fuser: identify processes using a fuser(1M)
fread, fwrite: binary input/output. . . fread(3S)
connect accounting records. fwtmp, wtmpfix: manipulate . fwtmp(1M)
cribbage: the card game cribbage. . . . . cribbage(6)
moo: guessing game. . . . . moo(6)
back: the game of backgammon. . . . back(6)
bj: the game of black jack. . . . . bj(6)

craps: the game of craps. . . . . craps(6)
wump: the game of hunt-the-wumpus. . . wump(6)
life: play the game of life. . . . . life(6)
trek: trekkie game. . . . . trek(6)
worm: play the growing worm game. . . . . worm(6)
gamma: log gamma function. . . . . gamma(3M)
gamma: log gamma function. . gamma(3M)
number to string. ecvt, fcvt, gcvt: convert floating-point . ecvt(3C)
gd: generic disk interface. . . gd(7)
termio: general terminal interface. . . termio(7)
a document. subj: generate a list of subjects from subj(1)
maze: generate a maze. . . . . maze(6)
abort: generate an IOT fault. . . . . abort(3C)
cflow: generate C flowgraph. . . . . cflow(1)
cross-reference. cxref: generate C program . . . . cxref(1)
crypt, setkey, encrypt: generate DES encryption. . . crypt(3C)
user ID. diskusg: generate disk accounting data by diskusg(1M)
makekey: generate encryption key. . . . makekey(1)
ctermid: generate filename for terminal. ctermid(3S)
ncheck: generate names from i-numbers. ncheck(1M)
lexical tasks. lex: generate programs for simple . lex(1)
/srand48, seed48, lcong48: generate uniformly distributed/ drand48(3C)
fsirand: install random inode generation numbers. . . . . fsirand(1M)
rand, srand: simple random-number generator. . . . . rand(3C)
Fortran uniform random-number generator. irand, srand, rand: . rand(3F)
gd: generic disk interface. . . . . gd(7)
file. get: get a version of an SCCS . get(1)
command-line argument. getarg: return Fortran . . . . getarg(3F)
character or word from a stream. getc, getchar, fgetc, getw: get . getc(3S)
character or word from a/ getc, getchar, fgetc, getw: get . . . getc(3S)
compatibility mode. setcompat, getcompat: set or get process . setcompat(2)
working directory. getcwd: get pathname of current getcwd(3C)
entries in a file system/ getdirentries: gets directory . getdirentries(2)
get/set name of current network/ getdomainname, setdomainname: getdomainname(2N)
table size getdtablesize: get descriptor . getdtablesize(2)
user,/ getuid, geteuid, getgid, getegid: get real user, effective getuid(2)
environment variable. getenv: return Fortran . . . . getenv(3F)
environment name. getenv: return value for . . . getenv(3C)
real user, effective/ getuid, geteuid, getgid, getegid: get . getuid(2)
effective user,/ getuid, geteuid, getgid, getegid: get real user, . getuid(2)
setgrent, endgrent, fgetgrent:/ getgrent, getgrgid, getgrnam, . getgrent(3C)
endgrent, fgetgrent:/ getgrent, getgrgid, getgrnam, setgrent, . getgrent(3C)
fgetgrent:/ getgrent, getgrgid, getgrnam, setgrent, endgrent, . getgrent(3C)
getgroups: get group access list. getgroups(2)
sethostent,/ gethostent, gethostbyaddr, gethostbyname, gethostent(3N)
gethostent, gethostbyaddr, gethostbyname, sethostent,/ . gethostent(3N)
gethostbyname, sethostent,/ gethostent, gethostbyaddr, . . gethostent(3N)
unique identifier of current/ gethostid, sethostid: get/set . . gethostid(2N)
name of current host. gethostname, sethostname: get/set gethostname(2N)
value of interval timer. getitimer, setitimer: get/set . . getitimer(2)
getlogin: get login name. . . getlogin(3C)

hasmntopt: get file/ setmntent, getmntent, addmntent, endmntent, getmntent(3)
setnetent, endnetent:/ getnetent, getnetbyaddr, getnetbyname, . getnetent(3N)
getnetent, getnetbyaddr, getnetbyname, setnetent,/ . . getnetent(3N)
getnetbyname, setnetent,/ getnetent, getnetbyaddr, . . . getnetent(3N)
endnetgrent, innetgr: get/ getnetgrent, setnetgrent, . . . getnetgrent(3N)
argument vector. getopt: get option letter from . getopt(3C)
getopt: parse command options. getopt(1)
getpass: read a password. . . getpass(3C)
connected peer. getpeername: get name of . . getpeername(2N)
process group, and/ getpid, getpgrp, getppid: get process, . getpid(2)
process, process group, and/ getpid, getpgrp, getppid: get . getpid(2)
group, and/ getpid, getpgrp, getppid: get process, process . getpid(2)
getprotoent, getprotobynumber, getprotobyname, setprotoent,/ getprotoent(3N)
setprotoent,/ getprotoent, getprotobynumber, getprotobyname, getprotoent(3N)
getprotobyname, setprotoent,/ getprotoent, getprotobynumber, getprotoent(3N)
endptabent, setptabent,/ getptabent, addptabent, . . . getptabent(3)
getpw: get name from UID. . getpw(3C)
setpwent, endpwent, fgetpwent:/ getpwent, getpwuid, getpwnam, getpwent(3C)
fgetpwent:/ getpwent, getpwuid, getpwnam, setpwent, endpwent, getpwent(3C)
endpwent, fgetpwent:/ getpwent, getpwuid, getpwnam, setpwent, getpwent(3C)
system/ getdirentries: gets directory entries in a file . getdirentries(2)
stream. gets, fgets: get a string from a gets(3S)
getservent, getservbyport, getservbyname, setservent,/ . getservent(3N)
setservent,/ getservent, getservbyport, getservbyname, getservent(3N)
getservbyname, setservent,/ getservent, getservbyport, . . getservent(3N)
gettimeofday, settimeofday: get/set date and time. . . . . gettimeofday(2)
gethostname, sethostname: get/set name of current host. . gethostname(2N)
getdomainname, setdomainname: get/set name of current network/ getdomainname(2N)
current/ gethostid, sethostid: get/set unique identifier of . . gethostid(2N)
getitimer, setitimer: get/set value of interval timer. getitimer(2)
getsockname: get socket name. getsockname(2N)
set options on sockets. getsockopt, setsockopt: get and getsockopt(2N)
get/set date and time. gettimeofday, settimeofday: . gettimeofday(2)
and terminal settings used by getty. gettydefs: speed . . . gettydefs(4)
speed, and line discipline. getty: set terminal type, modes, getty(1M)
ct: spawn getty to a remote terminal. . . ct(1C)
settings used by getty. gettydefs: speed and terminal . gettydefs(4)
get real user, effective user,/ getuid, geteuid, getgid, getegid: getuid(2)
pututline, setutent, endutent,/ getutent, getutid, getutline, . . getut(3C)
setutent, endutent,/ getutent, getutid, getutline, pututline, . getut(3C)
endutent,/ getutent, getutid, getutline, pututline, setutent, . getut(3C)
a stream. getc, getchar, fgetc, getw: get character or word from getc(3S)
directory pathname. getwd: get current working . . getwd(3)
head: give first few lines. . . . . head(1)
characters in the files in the given directories. /sum and count sumdir(1)
tzsetwall:/ ctime, localtime, gmtime, asctime, tzset, . . . ctime(3)
fish: play "Go Fish". . . . . fish(6)
setjmp, longjmp: non-local goto. . . . . setjmp(3C)
drawing graphs. grap: pic preprocessor for . . grap(1)
graph: draw a graph. . . . . graph(1G)
graph: draw a graph. . . . . graph(1G)

sag: system activity graph. . . . . sag(1G)
tplot: graphics filters. . . . . tplot(1G)
type-box. greek: graphics for the extended TTY-37 greek(5)
plot: graphics interface. . . . . plot(4)
plot: graphics interface subroutines. plot(3X)
mvt: typeset view graphs and slides. . . . . mvt(1)
pic preprocessor for drawing graphs. grap: . . . . . grap(1)
TTY-37 type-box. greek: graphics for the extended greek(5)
greek: select terminal filter. . greek(1)
for a pattern. grep, egrep, fgrep: search a file grep(1)
getgroups: get group access list. . . . . getgroups(2)
initgroups: initialize group access list. . . . . initgroups(3)
setgroups: set group access list. . . . . setgroups(2)
/real user, effective user, real group, and effective group IDs. getuid(2)
/getppid: get process, process group, and parent process IDs. getpid(2)
chown, chgrp: change owner or group. . . . . chown(1)
endnetgrent, innetgr: get network group entry. /setnetgrent, . . getnetgrent(3N)
/endgrent, fgetgrent: obtain group file entry from a group/ getgrent(3C)
obtain group file entry from a group file. /endgrent, fgetgrent: getgrent(3C)
group: group file. . . . . group(4)
group: group file. . . . . group(4)
setpgrp: set process group ID. . . . . setpgrp(2)
setregid: set real and effective group ID. . . . . setregid(2)
id: print user and group IDs and names. . . . id(1)
user, real group, and effective gro up IDs. /real user, effective getuid(2)
setuid, setgid: set user and group IDs. . . . . setuid(2)
setuid, setgid: set user and group IDs. . . . . setuid(3)
killpg: send signal to a process group. . . . . killpg(3N)
groups: show group memberships. . . . groups(1)
newgrp: log in to a new group. . . . . newgrp(1)
chown, fchown: change owner and group of a file. . . . . chown(2)
send a signal to a process or a group of processes. kill: . . . kill(2)
netgroup: list of network groups. . . . . netgroup(4)
maintain, update, and regenerate groups of programs. make: . make(1)
groups: show group memberships. groups(1)
worm: play the growing worm game. . . . . worm(6)
checkers. pwck, grpck: password/group file . . pwck(1M)
ssignal, gsignal: software signals. . . ssignal(3C)
hangman: guess the word. . . . . hangman(6)
moo: guessing game. . . . . moo(6)
nfs_getfh: get a file handle. . . . . nfs_getfh(2)
300 and 300s/ 300, 300s: handle special functions of DASI 300(1)
DASI 450 terminal. 450: handle special functions of the 450(1)
varargs: handle variable argument list. . varargs(3X)
block information for bad block handling. altblk: alternate . . altblk(4)
package. curses: CRT screen handling and optimization . . curses(3X)
hangman: guess the word. . . hangman(6)
nohup: run a command immune to hangups. . . . . nohup(1)
hcreate, hdestroy: manage hash search tables. hsearch, . hsearch(3C)
spell, hashmake, spellin, hashcheck: find spelling errors. spell(1)
find spelling errors. spell, hashmake, spellin, hashcheck: spell(1)

/getmntent, addmntent, endmntent, hasmntopt: get file system/ . . getmntent(3)
search tables. hsearch, hcreate, hdestroy: manage hash hsearch(3C)
tables. hsearch, hcreate, hdestroy: manage hash search hsearch(3C)
head: give first few lines. . . head(1)
scnhdr: section header for a common object file. scnhdr(4)
aouthdr.h: a.out header for common object files. aouthdr(4)
filehdr: file header for common object files. filehdr(4)
ldfhread: read the file header of a common object file. ldfhread(3X)
/seek to the optional file header of a common object file. ldohseek(3X)
/read an indexed/named section header of a common object file. ldshread(3X)
file. ldahread: read the archive header of a member of an archive ldahread(3X)
help: ask for help in using SCCS. help(1)
help: ask for help in using SCCS. . . . . help(1)
a YP map from some YP server to here. ypxfr: transfer . . . . ypxfr(1M)
hex: translates object files. . . hex(1)
fortune: print a random, hopefully interesting, adage. . fortune(6)
/ntohs: convert values between host and network byte order. . byteorder(3N)
remote: remote host description file. . . . . remote(4)
endhostent: get network host entry. /sethostent, . . . gethostent(3N)
unique identifier of current host. /sethostid: get/set . . . gethostid(2N)
get/set name of current host. gethostname, sethostname: gethostname(2N)
master?. ypwhich: which host is the YP server or map . ypwhich(1)
hosts: host name data base. . . . . hosts(4)
phones: remote host phone number data base. . phones(4)
(RPC version). rup: show host status of local machines . rup(1C)
ruptime: show host status of local machines. . ruptime(1N)
or print identifier of current host system. hostid: set . . . hostid(1N)
set or print name of current host system. hostname: . . . hostname(1N)
uusend: send a file to a remote host. . . . . uusend(1C)
of a YP map is at a YP server host. yppoll: what version . . yppoll(1M)
of current host system. hostid: set or print identifier . hostid(1N)
current host system. hostname: set or print name of hostname(1N)
hosts: host name data base. . . hosts(4)
hosts.equiv: list of trusted hosts. . . . . hosts.equiv(4)
ECHO_REQUEST packets to network hosts. ping: send ICMP . . . ping(1M)
hosts. hosts.equiv: list of trusted . . hosts.equiv(4)
manage hash search tables. hsearch, hcreate, hdestroy: . . hsearch(3C)
convert values between host and/ htonl, htons, ntohl, ntohs: . . byteorder(3N)
values between host and/ htonl, htons, ntohl, ntohs: convert . byteorder(3N)
wump: the game of hunt-the-wumpus. . . . . wump(6)
function. cosh, dcosh: Fortran hyperbolic cosine intrinsic . . cosh(3F)
sinh, cosh, tanh: hyperbolic functions. . . . . sinh(3M)
function. sinh, dsinh: Fortran hyperbolic sine intrinsic . . . sinh(3F)
function. tanh, dtanh: Fortran hyperbolic tangent intrinsic . tanh(3F)
hyphen: find hyphenated words. hyphen(1)
hyphen: find hyphenated words. . . . . hyphen(1)
function. hypot: Euclidean distance . . hypot(3M)
absolute value. abs, iabs, dabs, cabs, zabs: Fortran abs(3F)
arguments. iargc: return command line . . iargc(3F)
/float, sngl, dble, cmplx, dcmplx, ichar, char: explicit Fortran/ . ftype(3F)
network hosts. ping: send ICMP ECHO_REQUEST packets to ping(1M)

disk accounting data by user ID. diskusg: generate . . . . diskusg(1M)
semaphore set, or shared memory ID. ipcrm: remove message queue, ipcrm(1)
names. id: print user and group IDs and id(1)
setpgrp: set process group ID. . . . . setpgrp(2)
set real and effective group ID. setregid: . . . . . setregid(2)
su: substitute user ID. . . . . su(1)
print effective current user ID. whoami: . . . . . whoami(1)
issue: issue identification file. . . . . issue(4)
/sethostid: get/set unique identifier of current host. . . gethostid(2N)
system. hostid: set or print identifier of current host . . . hostid(1N)
or file structure. fuser: identify processes using a file . fuser(1M)
what: identify SCCS files. . . . . what(1)
intrinsic functions. dim, ddim, idim: Fortran positive difference dim(3F)
cmplx, dcmplx, ichar,/ int, ifix, idint, real, float, sngl, dble, . . ftype(3F)
functions. anint, dnint, nint, idnint: Fortran nearest integer round(3F)
id: print user and group IDs and names. . . . . id(1)
process group, and parent process IDs. /getppid: get process, . . getpid(2)
real group, and effective group IDs. /real user, effective user, getuid(2)
set real and effective user ID's. setreuid: . . . . . setreuid(2)
setgid: set user and group IDs. setuid, . . . . . setuid(2)
setgid: set user and group IDs. setuid, . . . . . setuid(3)
interface parameters. ifconfig: configure network . ifconfig(1M)
dble, cmplx, dcmplx, ichar,/ int, ifix, idint, real, float, sngl, . . ftype(3F)
daiw: Apple ImageWriter II troff postprocessor filter. . . daiw(1)
core: format of core image file. . . . . core(4)
postprocessor/ daiw: Apple ImageWriter II troff . . . . daiw(1)
iw2: Apple ImageWriter print filter. . . . iw2(1)
argument. aimag, dimag: Fortran imaginary part of complex . . aimag(3F)
nohup: run a command immune to hangups. . . . . nohup(1)
/strings from C programs to implement shared strings. . . xstr(1)
which: locate a program file including aliases and paths. . which(1)
fsync: synchronize a file's in-core state with that on disk. fsync(2)
finc: fast incremental backup. . . . . finc(1M)
dumpfs: incremental dump format. . . dumpfs(4)
dumpfs: incremental file system dump. dumpfs(1M)
restore: incremental file system restore. restore(1M)
source. indent: indent and format C program . indent(1)
program source. indent: indent and format C . indent(1)
long integer data in a machine independent fashion. /access . sputl(3X)
entries in a file system independent format /directory getdirentries(2)
/tgetstr, tgoto, tputs: terminal independent operation routines. termcap(3X)
lookbib, indxbib: build inverted index for a bibliography, find/ lookbib(1)
ndx: create a subject-page index for a document. . . . . ndx(1)
package for formatting a permuted index. mptx: the macro . . . mptx(5)
a common/ ldtbindex: compute the index of a symbol table entry of ldtbindex(3X)
ptx: make permuted index. . . . . ptx(1)
substring. index: return location of Fortran index(3F)
common object/ ldtbread: read an indexed symbol table entry of a ldtbread(3X)
a/ ldshread, ldnshread: read an indexed/named section header of ldshread(3X)
ldsseek, ldnsseek: seek to an indexed/named section of a/ , ldsseek(3X)
teletypes. last: indicate last logins of users and last(1)

a bibliography, find/ lookbib, indxbib: build inverted index for lookbib(1)
inet: Internet protocol family. . inet(5f)
servers: Inet server data base. . . . . servers(4)
inet_ntoa, inet_makeaddr,/ inet_addr, inet_network, . . . inet(3N)
inetd: Internet services daemon. inetd(1M)
/inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof: Internet/ inet(3N)
/inet_network, inet_ntoa, inet_makeaddr, inet_lnaof,/ . inet(3N)
/inet_makeaddr, inet_lnaof, inet_netof: Internet address/ . inet(3N)
inet_makeaddr,/ inet_addr, inet_network, inet_ntoa, . . . inet(3N)
inet_addr, inet_network, inet_ntoa, inet_makeaddr,/ . . inet(3N)
fstab: static information about file systems. fstab(4)
badblk: set or update bad block information. . . . . badblk(1M)
file. reloc: relocation information for a common object reloc(4)
altblk: alternate block information for bad block/ . . altblk(4)
/strip symbol and line number information from an object file. strip(1)
module_dump: dumps out information from A/UX kernels. module_dump(1M)
finger: user information lookup program. . finger(1)
lpstat: print LP status information. . . . . lpstat(1)
rpcinfo: report RPC information. . . . . rpcinfo(1M)
tzfile: time zone information. . . . . tzfile(4)
system-specific configuration information. uvar: returns . . uvar(2)
inittab: script for the init process. . . . . inittab(4)
initialization. init, telinit: process control . . init(1M)
access list. initgroups: initialize group . . initgroups(3)
init, telinit: process control initialization. . . . . init(1M)
/bcheckrc, rc, powerfail: system initialization shell scripts. . . brc(1M)
initgroups: initialize group access list. . . initgroups(3)
socket. connect: initiate a connection on a . . connect(2N)
popen, pclose: initiate pipe to/from a process. popen(3S)
process. inittab: script for the init . . . inittab(4)
/setnetgrent, endnetgrent, innetgr: get network group entry. getnetgrent(3N)
clri: clear inode. . . . . clri(1M)
inode. inode: format of a System V . inode(4)
fsirand: install random inode generation numbers. . . fsirand(1M)
inode: format of a System V inode. . . . . inode(4)
mouse: mouse input device driver. . . . . mouse(7)
fscanf, sscanf: convert formatted input. scanf, . . . . . scanf(3S)
eliminate .so's from nroff input. soelim: . . . . . soelim(1)
ungetc: push character back into input stream. . . . . ungetc(3S)
fread, fwrite: binary input/output. . . . . fread(3S)
clearerr, fileno: stream status inquiries. ferror, feof, . . . . ferror(3S)
uustat: uucp status inquiry and job control. . . . uustat(1C)
documents. refer: find and insert literature references in . refer(1)
queue. insque, remque: insert/remove element from a . insque(3N)
element from a queue. insque, remque: insert/remove insque(3N)
install: install commands. . . . . install(1M)
install: install commands. . . install(1M)
directories. cpset: install object files in binary . . cpset(1M)
numbers. fsirand: install random inode generation fsirand(1M)
ypinit: build and install yellow pages database. . ypinit(1M)
sngl, dble, cmplx, dcmplx,/ int, ifix, idint, real, float, . . . ftype(3F)

abs: return integer absolute value. . . . abs(3C)
a64l, l64a: convert between long integer and base-64 ASCII string. a64l(3C)
sputl, sgetl: access long integer data in a machine/ . . sputl(3X)
nint, idnint: Fortran nearest integer functions. anint, dnint, round(3F)
aint, dint: Fortran integer part intrinsic function. aint(3F)
atol, atoi: convert string to integer. strtol, . . . . . strtol(3C)
/ltol3: convert between 3-byte integers and long integers. . . l3tol(3C)
between 3-byte integers and long integers. l3tol, ltol3: convert . l3tol(3C)
bcopy: interactive block copy. . . . bcopy(1M)
system. mailx: interactive message processing mailx(1)
file system consistency check and interactive repair. fsck: . . . fsck(1M)
print a random, hopefully interesting, adage. fortune: . . fortune(6)
ae: 3Com 10 Mb/s Ethernet interface. . . . . ae(5)
dual: 3Com Ethernet interface diagnostic. . . . . dual(1M)
error: error-logging interface. . . . . error(7)
fd: floppy disk drive interface. . . . . fd(7)
gd: generic disk interface. . . . . gd(7)
lo: software loopback network interface. . . . . lo(5)
memory/time of day clock interface. nvram: nonvolatile . nvram(7)
ifconfig: configure network interface parameters. . . . . ifconfig(1M)
plot: graphics interface. . . . . plot(4)
set42sig: set 4.2 BSD signal interface. . . . . set42sig(3)
streams: ioctl interface. . . . . streams(7)
plot: graphics interface subroutines. . . . . plot(3X)
swap: swap administrative interface. . . . . swap(1M)
termio: general terminal interface. . . . . termio(7)
telnet: user interface to the TELNET protocol. telnet(1C)
tty: controlling terminal interface. . . . . tty(7)
ypclnt: yellow pages client interface. . . . . ypclnt(3N)
etheraddr: read an interface's Ethernet address.. . etheraddr(1M)
psdit: convert troff intermediate format to/ . . . psdit(1)
/inet_lnaof, inet_netof: Internet address manipulation/ inet(3N)
named: Internet domain name server. . named(1M)
server. ftpd: DARPA Internet File Transfer Protocol ftpd(1M)
inet: Internet protocol family. . . . inet(5f)
ip: Internet Protocol. . . . . ip(5P)
sendmail: send mail over the Internet. . . . . sendmail(1M)
inetd: Internet services daemon. . . inetd(1M)
Protocol. tcp: Internet Transmission Control tcp(5P)
udp: Internet User Datagram Protocol. udp(5P)
spline: interpolate smooth curve. . . spline(1G)
characters. asa: interpret ASA carriage control asa(1)
environment. sash: a command interpreter for the standalone . sash(8)
sno: SNOBOL interpreter. . . . . sno(1)
tc: troff output interpreter. . . . . tc(1)
csh: C shell, a command interpreter with C-like syntax. csh(1)
pipe: create an interprocess channel. . . . . pipe(2)
facilities status. ipcs: report interprocess communication . ipcs(1)
package. ftok: standard interprocess communication . ftok(3C)
blocked signals and wait for interrupt. /atomically release . sigpause(2)
sleep: suspend execution for an interval. . . . . sleep(1)

sleep: suspend execution for interval. . . . . sleep(3C)
setitimer: get/set value of interval timer. getitimer, . . getitimer(2)
acos, dacos: Fortran arccosine intrinsic function. . . . . acos(3F)
aint, dint: Fortran integer part intrinsic function. . . . . aint(3F)
asin, dasin: Fortran arcsine intrinsic function. . . . . asin(3F)
atan2, datan2: Fortran arctangent intrinsic function. . . . . atan2(3F)
atan, datan: Fortran arctangent intrinsic function. . . . . atan(3F)
dconjg: Fortran complex conjugate intrinsic function. conjg, . . conjg(3F)
cos, dcos, ccos: Fortran cosine intrinsic function. . . . . cos(3F)
dcosh: Fortran hyperbolic cosine intrinsic function. cosh, . . . cosh(3F)
Fortran double precision product intrinsic function. dprod: . . dprod(3F)
dexp, cexp: Fortran exponential intrinsic function. exp, . . . exp(3F)
dlog10: Fortran common logarithm intrinsic function. /alog10, . . log10(3F)
clog: Fortran natural logarithm intrinsic function. /alog, dlog, log(3F)
dsign: Fortran transfer-of-sign intrinsic function. sign, isign, . sign(3F)
sin, dsin, csin: Fortran sine intrinsic function. . . . . sin(3F)
dsinh: Fortran hyperbolic sine intrinsic function. sinh, . . . sinh(3F)
dsqrt, csqrt: Fortran square root intrinsic function. sqrt, . . . sqrt(3F)
tan, dtan: Fortran tangent intrinsic function. . . . . tan(3F)
dtanh: Fortran hyperbolic tangent intrinsic function. tanh, . . . tanh(3F)
idim: Fortran positive difference intrinsic functions. dim, ddim, dim(3F)
lgt, lle, llt: string comparision intrinsic functions. lge, . . . lge(3F)
amod, dmod: Fortran remaindering intrinsic functions. mod, . . mod(3F)
formats. intro: introduction to file . . . intro(4)
miscellaneous facilities. intro: introduction to . . . . intro(5)
subroutines and libraries. intro: introduction to . . . . intro(3)
calls and error numbers. intro: introduction to system . intro(2)
intro: introduction to file formats. . intro(4)
facilities. intro: introduction to miscellaneous . intro(5)
libraries. intro: introduction to subroutines and intro(3)
error numbers. intro: introduction to system calls and intro(2)
ncheck: generate names from i-numbers. . . . . ncheck(1M)
aliens: alien invaders attack the earth. . . aliens(6)
lookbib, indxbib: build inverted index for a/ . . . . lookbib(1)
select: synchronous I/O multiplexing. . . . . select(2N)
ioctl: control device. . . . . ioctl(2)
streams: ioctl interface. . . . . streams(7)
abort: generate an IOT fault. . . . . abort(3C)
ip: Internet Protocol. . . . . ip(5P)
semaphore set, or shared memory/ ipcrm: remove message queue, ipcrm(1)
communication facilities status. ipcs: report interprocess . . . ipcs(1)
uniform random-number generator. irand, srand, rand: Fortran . . rand(3F)
whatis: describe what a command is. . . . . whatis(1)
/islower, isdigit, isxdigit, isalnum, isspace, ispunct,/ . . ctype(3C)
isdigit, isxdigit, isalnum,/ isalpha, isupper, islower, . . ctype(3C)
/isprint, isgraph, iscntrl, isascii: classify characters. . . ctype(3C)
ttyname, isatty: find name of a terminal. ttyname(3C)
/ispunct, isprint, isgraph, iscntrl, isascii: classify/ . . . ctype(3C)
isalpha, isupper, islower, isdigit, isxdigit, isalnum,/ . . ctype(3C)
/isspace, ispunct, isprint, isgraph, iscntrl, isascii:/ . . . ctype(3C)
transfer-of-sign intrinsic/ sign, isign, dsign: Fortran . . . . sign(3F)

isalnum,/ isalpha, isupper, islower, isdigit, isxdigit, . . . ctype(3C)
/isalnum, isspace, ispunct, isprint, isgraph, iscntrl,/ . . . ctype(3C)
/isxdigit, isalnum, isspace, ispunct, isprint, isgraph,/ . . ctype(3C)
/isdigit, isxdigit, isalnum, isspace, ispunct, isprint,/ . . . ctype(3C)
Fortran. system: issue a shell command from . system(3F)
system: issue a shell command. . . . system(3S)
issue: issue identification file. . . . issue(4)
issue: issue identification file. . issue(4)
isxdigit, isalnum,/ isalpha, isupper, islower, isdigit, . . . ctype(3C)
/isupper, islower, isdigit, isxdigit, isalnum, isspace,/ . . ctype(3C)
news: print local news items. . . . . news(1)
filter. iw2: Apple ImageWriter print iw2(1)
character map description files. iwmap: format of iwprep(1) . iwmap(4)
files. iwprep: prepare troff description iwprep(1)
description/ iwmap: format of iwprep(1) character map . . . iwmap(4)
functions. j0, j1, jn, y0, y1, yn: Bessel . bessel(3M)
functions. j0, j1, jn, y0, y1, yn: Bessel . . . bessel(3M)
bj: the game of black jack. . . . . bj(6)
j0, j1, jn, y0, y1, yn: Bessel functions. bessel(3M)
uustat: uucp status inquiry and job control. . . . . uustat(1C)
spooling queue. lprm: remove jobs from the line printer . . lprm(1)
operator. join: relational database . . . join(1)
/lrand48, nrand48, mrand48, jrand48, srand48, seed48,/ . . drand48(3C)
parameters for tuning. kconfig: change a kernel's . . kconfig(1M)
kermit: kermit file transfer. . . . . kermit(1C)
kermit: kermit file transfer. . . kermit(1C)
build a new up-to-date kernel. autoconfig: . . . . . autoconfig(1M)
launch: launch an A/UX kernel from the standalone/ . launch(8)
ncstats: display kernel name cache statistics. . ncstats(1M)
rstatd: kernel statistics server. . . . rstatd(1M)
dumps out information from A/UX kernels. module_dump: . . . module_dump(1M)
kconfig: change a kernel's parameters for tuning. kconfig(1M)
makekey: generate encryption key. . . . . makekey(1)
console: keyboard/screen driver. . . . console(7)
print the value of one or more keys from a YP map. ypmatch: ypmatch(1)
apropos: locate commands by keyword lookup. . . . . apropos(1)
killall: kill all active processes. . . . killall(1M)
or a group of processes. kill: send a signal to a process kill(2)
kill: terminate a process. . . . kill(1)
processes. killall: kill all active . . . . killall(1M)
chase: try to escape the killer robots. . . . . chase(6)
group. killpg: send signal to a process killpg(3N)
mem, kmem: core memory. . . . . mem(7)
quiz: test your knowledge. . . . . quiz(6)
language. ksh: Korn shell, a command programming ksh(1)
programming language. ksh: Korn shell, a command . ksh(1)
3-byte integers and long/ l3tol, ltol3: convert between . l3tol(3C)
integer and base-64 ASCII/ a64l, l64a: convert between long . . a64l(3C)
labelit: copy file systems with label checking. volcopy, . . volcopy(1M)
label checking. volcopy, labelit: copy file systems with volcopy(1M)
pattern scanning and processing language. awk: . . . . . awk(1)

arbitrary-precision arithmetic language. bc: . . . . . bc(1)
efl: Extended Fortran Language. . . . . efl(1)
Korn shell, a command programming language. ksh: . . . . . ksh(1)
nroff: text formatting language. . . . . nroff(1)
cpp: the C language preprocessor. . . . cpp(1)
command programming language. /standard/restricted sh(1)
troff: description of output language. . . . . troff(5)
banner7: print large banner on printer. . . . banner7(1)
bdiff: diff large files. . . . . bdiff(1)
users and teletypes. last: indicate last logins of . . last(1)
chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm,/ . acctsh(1M)
at, batch: execute commands at a later time. . . . . at(1)
standalone environment. launch: launch an A/UX kernel from the launch(8)
binary application. launch: execute a Macintosh . launch(1)
from the standalone environment. launch: launch an A/UX kernel launch(8)
statistics. lav: print load average . . . lav(1)
shl: shell layer manager. . . . . shl(1)
/jrand48, srand48, seed48, lcong48: generate uniformly/ . drand48(3C)
files. ld: link editor for common object ld(1)
file. ldclose, ldaclose: close a common object ldclose(3X)
of a member of an archive file. ldahread: read the archive header ldahread(3X)
file for reading. ldopen, ldaopen: open a common object ldopen(3X)
object file. ldclose, ldaclose: close a common ldclose(3X)
floating-point numbers. frexp, ldexp, modf: manipulate parts of frexp(3C)
routines. ldfcn: common object file access ldfcn(3X)
a common object file. ldfhread: read the file header of ldfhread(3X)
for object file. ldgetname: retrieve symbol name ldgetname(3X)
number entries of a/ ldlread, ldlinit, ldlitem: manipulate line ldlread(3X)
entries of a/ ldlread, ldlinit, ldlitem: manipulate line number ldlread(3X)
manipulate line number entries/ ldlread, ldlinit, ldlitem: . . . ldlread(3X)
number entries of a section of a/ ldlseek, ldnlseek: seek to line . ldlseek(3X)
entries of a section of/ ldlseek, ldnlseek: seek to line number . ldlseek(3X)
entries of a section of/ ldrseek, ldnrseek: seek to relocation . ldrseek(3X)
section header of a/ ldshread, ldnshread: read an indexed/named ldshread(3X)
indexed/named section/ ldsseek, ldnsseek: seek to an . . . . ldsseek(3X)
file header of a common object/ ldohseek: seek to the optional . ldohseek(3X)
object file for reading. ldopen, ldaopen: open a common ldopen(3X)
relocation entries of a section/ ldrseek, ldnrseek: seek to . . ldrseek(3X)
indexed/named section header of/ ldshread, ldnshread: read an . ldshread(3X)
indexed/named section of a/ ldsseek, ldnsseek: seek to an . ldsseek(3X)
symbol table entry of a common/ ldtbindex: compute the index of a ldtbindex(3X)
table entry of a common object/ ldtbread: read an indexed symbol ldtbread(3X)
table of a common object file. ldtbseek: seek to the symbol . ldtbseek(3X)
remind you when you have to leave. leave: . . . . . leave(1)
to leave. leave: remind you when you have leave(1)
string. len: return length of Fortran . len(3F)
len: return length of Fortran string. . . . len(3F)
truncate a file to a specified length. truncate, ftruncate: . . truncate(2)
getopt: get option letter from argument vector. . getopt(3C)
lexical tasks. lex: generate programs for simple lex(1)
lex: generate programs for simple lexical tasks. . . . . lex(1)

lsearch, lfind: linear search and update. lsearch(3C)
comparision intrinsic functions. lge, lgt, lle, llt: string . . . . lge(3F)
intrinsic functions. lge, lgt, lle, llt: string comparision . lge(3F)
introduction to subroutines and libraries. intro: . . . . . intro(3)
slots: ROM library functions. . . . . slots(3X)
ordering relation for an object library. lorder: find . . . . . lorder(1)
archives. ar: archive and library maintainer for portable ar(1)
data representation. xdr: library routines for external . xdr(3N)
procedure calls. rpc: library routines for remote . . rpc(3N)
life: play the game of life. . . . . life(6)
life: play the game of life. . . life(6)
ulimit: get and set user limits. . . . . ulimit(2)
iargc: return command line arguments. . . . . iargc(3F)
establish an out-going terminal line connection. dial: . . . . dial(3C)
terminal type, modes, speed, and line discipline. getty: set . . getty(1M)
routine used to push streams line disciplines. line_push: . . line_push(3)
line_sane: push streams line disciplines. . . . . line_sane(1M)
line: read one line. . . . . line(1)
object file. linenum: line number entries in a common linenum(4)
/ldlinit, ldlitem: manipulate line number entries of a common/ ldlread(3X)
of a/ ldlseek, ldnlseek: seek to line number entries of a section ldlseek(3X)
object/ strip: strip symbol and line number information from an strip(1)
nl: line numbering filter. . . . . nl(1)
cut out selected fields of each line of a file. cut: . . . . . cut(1)
display system status on status line of a terminal. sysline: . . sysline(1)
cancel: send/cancel requests to a line printer. lp, . . . . . lp(1)
lpr: send requests to a line printer. . . . . lpr(1)
lprm: remove jobs from the line printer spooling queue. . lprm(1)
line: read one line. . . . . line(1)
lsearch, lfind: linear search and update. . . lsearch(3C)
col: filter reverse linefeeds. . . . . col(1)
common object file. linenum: line number entries in a linenum(4)
streams line disciplines. line_push: routine used to push line_push(3)
comm: select or reject lines common to two sorted files. comm(1)
device. fold: fold long lines for finite-width output . fold(1)
head: give first few lines. . . . . head(1)
uniq: report repeated lines in a file. . . . . uniq(1)
rev: reverse lines of a file. . . . . rev(1)
of several files or subsequent lines of one file. /same lines . paste(1)
subsequent/ paste: merge same lines of several files or . . . paste(1)
disciplines. line_sane: push streams line . line_sane(1M)
files. ld: link editor for common object ld(1)
a.out: common assembler and link editor output. . . . . a.out(4)
link: link to a file. . . . . link(2)
read value of a symbolic link. readlink: . . . . . readlink(2)
link: link to a file. . . . . link(2)
symlink: make symbolic link to a file. . . . . symlink(2)
ln: make links. . . . . ln(1)
lint: a C program checker. . . lint(1)
ls: list contents of directory. . . ls(1)
for a file system. ff: list file names and statistics . ff(1M)

cml: configuration master list format. . . . . cml(4)
getgroups: get group access list. . . . . getgroups(2)
initialize group access list. initgroups: . . . . . initgroups(3)
nlist: get entries from name list. . . . . nlist(3C)
nm: print name list of common object file. . . nm(1)
netgroup: list of network groups. . . . netgroup(4)
subj: generate a list of subjects from a document. subj(1)
hosts.equiv: list of trusted hosts. . . . . hosts.equiv(4)
system. users: compact list of users who are on the . . users(1)
setgroups: set group access list. . . . . setgroups(2)
varargs: handle variable argument list. . . . . varargs(3X)
output of a varargs argument list. /vsprintf: print formatted . vprintf(3S)
socket. listen: listen for connections on a . . listen(2N)
a socket. listen: listen for connections on listen(2N)
macref: produce cross-reference listing of macro files. . . . . macref(1)
xargs: construct argument list(s) and execute command. . xargs(1)
refer: find and insert literature references in/ . . . refer(1)
intrinsic functions. lge, lgt, lle, llt: string comparision . . lge(3F)
functions. lge, lgt, lle, llt: string comparision intrinsic lge(3F)
ln: make links. . . . . ln(1)
interface. lo: software loopback network lo(5)
lav: print load average statistics. . . . lav(1)
rup: show host status of local machines (RPC version). rup(1C)
rusers: who's logged in on local machines (RPC version). rusers(1N)
ruptime: show host status of local machines. . . . . ruptime(1N)
rwho: who's logged in on local machines. . . . . rwho(1N)
news: print local news items. . . . . news(1)
tzset, tzsetwall: convert/ ctime, localtime, gmtime, asctime, . ctime(3)
aliases and paths. which: locate a program file including which(1)
lookup. apropos: locate commands by keyword apropos(1)
manual for program. whereis: locate source, binary, and/or . whereis(1)
index: return location of Fortran substring. . index(3F)
end, etext, edata: last locations in program. . . . . end(3C)
apply or remove an advisory lock on an open file. flock: . . flock(2)
memory. plock: lock process, text, or data in . plock(2)
lockf: record locking on files. . lockf(3C)
lockf: record locking on files. . . . . lockf(3C)
regions for reading or writing. locking: provide exclusive file locking(2)
natural logarithm intrinsic/ log, alog, dlog, clog: Fortran . log(3F)
gamma: log gamma function. . . . . gamma(3M)
newgrp: log in to a new group. . . . . newgrp(1)
exponential, logarithm,/ exp, log, log10, pow, sqrt: . . . . exp(3M)
common logarithm intrinsic/ log10, alog10, dlog10: Fortran log10(3F)
logarithm, power,/ exp, log, log10, pow, sqrt: exponential, exp(3M)
/alog10, dlog10: Fortran common logarithm intrinsic function. . log10(3F)
/alog, dlog, clog: Fortran natural logarithm intrinsic function. . log(3F)
/log10, pow, sqrt: exponential, logarithm, power, square root/ exp(3M)
errpt: process a report of logged errors. . . . . errpt(1M)
version). rusers: who's logged in on local machines (RPC rusers(1N)
rwho: who's logged in on local machines. . rwho(1N)
getlogin: get login name. . . . . getlogin(3C)

logname: get login name. . . . . logname(1)
cuserid: get character login name of the user. . . . cuserid(3S)
logname: return login name of user. . . . . logname(3X)
yppasswd: change login password in yellow pages. yppasswd(1)
passwd: change login password. . . . . passwd(1)
rlogin: remote login. . . . . rlogin(1N)
rlogind: remote login server. . . . . rlogind(1M)
chsh: change default login shell. . . . . chsh(1)
login: sign on. . . . . login(1)
setting up an environment at login time. profile: . . . . . profile(4)
last: indicate last logins of users and teletypes. . last(1)
logname: get login name. . . logname(1)
user. logname: return login name of logname(3X)
setjmp, longjmp: non-local goto. . . setjmp(3C)
index for a bibliography, find/ lookbib, indxbib: build inverted lookbib(1)
locate commands by keyword lookup. apropos: . . . . . apropos(1)
finger: user information lookup program. . . . . finger(1)
lo: software loopback network interface. . lo(5)
for an object library. lorder: find ordering relation . lorder(1)
nice: run a command at low priority. . . . . nice(1)
to a line printer. lp, cancel: send/cancel requests lp(1)
enable, disable: enable/disable LP printers. . . . . enable(1)
/lpshut, lpmove: start/stop the LP request scheduler and move/ lpsched(1M)
accept: allow LP requests. . . . . accept(1M)
reject: prevent LP requests. . . . . reject(1M)
lpadmin: configure the LP spooling system. . . . . lpadmin(1M)
lpstat: print LP status information. . . . lpstat(1)
spooling system. lpadmin: configure the LP . . lpadmin(1M)
scheduler and/ lpsched, lpshut, lpmove: start/stop the LP request lpsched(1M)
program. lpq: spool queue examination . lpq(1)
printer. lpr: send requests to a line . . lpr(1)
printer spooling queue. lprm: remove jobs from the line lprm(1)
start/stop the LP request/ lpsched, lpshut, lpmove: . . . lpsched(1M)
request scheduler and/ lpsched, lpshut, lpmove: start/stop the LP lpsched(1M)
information. lpstat: print LP status . . . . lpstat(1)
jrand48,/ drand48, erand48, lrand48, nrand48, mrand48, . drand48(3C)
ls: list contents of directory. . ls(1)
update. lsearch, lfind: linear search and lsearch(3C)
pointer. lseek: move read/write file . . lseek(2)
boolean/ and, or, xor, not, lshift, rshift: Fortran bitwise . bool(3F)
stat, fstat, lstat: get file status. . . . . stat(2)
integers and long/ l3tol, ltol3: convert between 3-byte . l3tol(3C)
m4: macro processor. . . . . m4(1)
u3b15, vax: provide truth value/ m68k, pdp11, u3b, u3b2, u3b5, machid(1)
/access long integer data in a machine independent fashion. sputl(3X)
put: puts a file onto a remote machine. . . . . put(1C)
take: takes a file from a remote machine. . . . . take(1C)
values: machine-dependent values. . . values(5)
rup: show host status of local machines (RPC version). . . rup(1C)
rusers: who's logged in on local machines (RPC version). . . rusers(1N)
show host status of local machines. ruptime: . . . . . ruptime(1N)

rwho: who's logged in on local machines. . . . . rwho(1N)
updater: update files between two machines. . . . . updater(1)
launch: execute a Macintosh binary application. launch(1)
mfs: read a Macintosh flat file system disk. mfs(1)
set the type and creator of a Macintosh resource file. settc: settc(1)
listing of macro files. macref: produce cross-reference macref(1)
cross-reference listing of macro files. macref: produce . macref(1)
permuted index. mptx: the macro package for formatting a mptx(5)
documents. mm: macro package for formatting mm(5)
viewgraphs and/ mv: a troff macro package for typesetting mv(5)
m4: macro processor. . . . . m4(1)
documents formatted with the mm macros. checkmm: check . . checkmm(1)
this manual. man: macros for formatting entries in man(5)
documents formatted with the mm macros. mm: prints . . . . mm(1)
ms: text formatting macros. . . . . ms(5)
time chip. mactime: set the system time/real mactime(1M)
number file. magic: file command's magic . magic(4)
magic: file command's magic number file. . . . . magic(4)
rebuild the data base for the mail aliases file. newaliases: . newaliases(1M)
biff: be notified if mail arrives and who it is from. biff(1)
from: who is my mail from?. . . . . from(1)
rmail: send mail to users or read mail. mail, . . . . . mail(1)
sendmail: send mail over the Internet. . . . . sendmail(1M)
or read mail. mail, rmail: send mail to users mail(1)
mail, rmail: send mail to users or read mail. . . mail(1)
binary file for transmission via mail. /uudecode: encode/decode a uuencode(1C)
processing system. mailx: interactive message . . mailx(1)
free, realloc, calloc, cfree: main memory allocator. malloc, malloc(3C)
calloc, mallopt, mallinfo: fast main memory allocator. /realloc, malloc(3X)
program. ctags: maintain a tags file for a C . . ctags(1)
groups of programs. make: maintain, update, and regenerate make(1)
ar: archive and library maintainer for portable archives. ar(1)
regenerate groups of programs. make: maintain, update, and . make(1)
file. makedbm: make a yellow pages dbm makedbm(1M)
makekey: generate encryption key. makekey(1)
/free, realloc, calloc, mallopt, mallinfo: fast main memory/ . malloc(3X)
cfree: main memory allocator. malloc, free, realloc, calloc, . malloc(3C)
mallopt, mallinfo: fast main/ malloc, free, realloc, calloc, . malloc(3X)
malloc, free, realloc, calloc, mallopt, mallinfo: fast main/ . malloc(3X)
entries in this manual. man: macros for formatting . man(5)
manual. man: print entries in this . . . man(1)
tsearch, tfind, tdelete, twalk: manage binary search trees. . tsearch(3C)
hsearch, hcreate, hdestroy: manage hash search tables. . . hsearch(3C)
shl: shell layer manager. . . . . shl(1)
records. fwtmp, wtmpfix: manipulate connect accounting fwtmp(1M)
a/ ldlread, ldlinit, ldlitem: manipulate line number entries of ldlread(3X)
frexp, ldexp, modf: manipulate parts of/ . . . . frexp(3C)
tp: manipulate tape archive. . . . tp(1)
route: manually manipulate the routing tables. . route(1M)
/inet_netof: Internet address manipulation routines. . . . inet(3N)
locate source, binary, and/or manual for program. whereis: whereis(1)

man: print entries in this manual. . . . . man(1)
for formatting entries in this manual. man: macros . . . . man(5)
tables. route: manually manipulate the routing route(1M)
format of iwprep(1) character map description files. iwmap: iwmap(4)
dpme: format of disk partition map entries. . . . . dpme(4)
ypxfr: transfer a YP map from some YP server to here. ypxfr(1M)
yppoll: what version of a YP map is at a YP server host. . . yppoll(1M)
which host is the YP server or map master?. ypwhich: . . . ypwhich(1)
ascii: map of ASCII character set. . ascii(5)
of one or more keys from a YP map. ypmatch: print the value ypmatch(1)
force propagation of a changed YP map. yppush: . . . . . yppush(1M)
DARPA port to RPC program number mapper. portmap: . . . . . portmap(1M)
diffmk: mark differences between files. diffmk(1)
sigsetmask: set current signal mask. . . . . sigsetmask(2)
umask: set and get file creation mask. . . . . umask(2)
create an error message file by massaging C source. mkstr: . mkstr(1)
cml: configuration master list format. . . . . cml(4)
host is the YP server or map master?. ypwhich: which . . ypwhich(1)
regular expression compile and match routines. regexp: . . . regexp(5)
math: math functions and constants. . math(5)
constants. math: math functions and . . math(5)
neqn: format mathematical text for nroff. . neqn(1)
eqn: format mathematical text for troff. . . eqn(1)
matherr: error-handling function. matherr(3M)
dmax1: Fortran maximum-value/ max, max0, amax0, max1, amax1, max(3F)
Fortran maximum-value/ max, max0, amax0, max1, amax1, dmax1: max(3F)
maximum-value/ max, max0, amax0, max1, amax1, dmax1: Fortran max(3F)
max1, amax1, dmax1: Fortran maximum-value functions. /amax0, max(3F)
maze: generate a maze. . . . maze(6)
maze: generate a maze. . . . . maze(6)
ae: 3Com 10 Mb/s Ethernet interface. . . . ae(5)
accounting. mclock: return Fortran time . mclock(3F)
bcd: convert to antique media. . . . . bcd(6)
mem, kmem: core memory. . mem(7)
/read the archive header of a member of an archive file. . . ldahread(3X)
groups: show group memberships. . . . . groups(1)
memset: memory operations. memccpy, memchr, memcmp, memcpy, memory(3C)
memory operations. memccpy, memchr, memcmp, memcpy, memset: memory(3C)
operations. memccpy, memchr, memcmp, memcpy, memset: memory memory(3C)
memccpy, memchr, memcmp, memcpy, memset: memory/ . memory(3C)
realloc, calloc, cfree: main memory allocator. malloc, free, malloc(3C)
mallopt, mallinfo: fast main memory allocator. /calloc, . . malloc(3X)
shmctl: shared memory control operations. . shmctl(2)
queue, semaphore set, or shared memory ID. ipcrm: remove message ipcrm(1)
mem, kmem: core memory. . . . . mem(7)
memchr, memcmp, memcpy, memset: memory operations. memccpy, memory(3C)
shmop, shmat, shmdt: shared memory operations. . . . . shmop(2)
lock process, text, or data in memory. plock: . . . . . plock(2)
shmget: get shared memory segment. . . . . shmget(2)
interface. nvram: nonvolatile memory/time of day clock . . nvram(7)
memccpy, memchr, memcmp, memcpy, memset: memory operations. . memory(3C)

sort: sort and/or merge files. . . . . sort(1)
files. acctmerg: merge or add total accounting acctmerg(1M)
or subsequent lines of/ paste: merge same lines of several files paste(1)
mesg: permit or deny messages. mesg(1)
msgctl: message control operations. . msgctl(2)
source. mkstr: create an error message file by massaging C . mkstr(1)
recvfrom, recvmsg: receive a message from a socket. recv, . recv(2N)
send, sendto, sendmsg: send a message from a socket. . . . send(2N)
msgop, msgsnd, msgsrv: message operations. . . . . msgop(2)
mailx: interactive message processing system. . mailx(1)
msgget: get message queue. . . . . msgget(2)
shared memory ID. ipcrm: remove message queue, semaphore set, or ipcrm(1)
mesg: permit or deny messages. . . . . mesg(1)
sys_nerr: system error messages. /errno, sys_errlist, . perror(3C)
afm: Adobe PostScript font metrics file format. . . . . afm(7)
system disk. mfs: read a Macintosh flat file mfs(1)
dmin1: Fortran minimum-value/ min, min0, amin0, min1, amin1, min(3F)
Fortran minimum-value/ min, min0, amin0, min1, amin1, dmin1: min(3F)
minimum-value/ min, min0, amin0, min1, amin1, dmin1: Fortran . min(3F)
min1, amin1, dmin1: Fortran minimum-value functions. /amin0, min(3F)
/overview of accounting and miscellaneous accounting/ . . acct(1M)
intro: introduction to miscellaneous facilities. . . . intro(5)
mkdir: make a directory file. . mkdir(2)
mkdir: make a directory. . . . mkdir(1)
mkfs: construct a file system. . mkfs(1M)
mkfs1b: construct a file system. mkfs1b(1M)
mknod: build special file. . . mknod(1M)
special or ordinary file. mknod: make a directory, or a mknod(2)
file by massaging C source. mkstr: create an error message mkstr(1)
mktemp: make a unique filename. mktemp(3C)
documents. mm: macro package for formatting mm(5)
documents formatted with the mm macros. checkmm: check checkmm(1)
documents formatted with the mm macros. mm: prints . . . mm(1)
with the mm macros. mm: prints documents formatted mm(1)
mmt: typeset documents. . . mmt(1)
remaindering intrinsic/ mod, amod, dmod: Fortran . . mod(3F)
chmod: change mode. . . . . chmod(1)
chmod: change mode of file. . . . . chmod(2)
set or get process compatibility mode. setcompat, getcompat: . setcompat(2)
dialup: modem escape sequence file. . dialup(4)
getty: set terminal type, modes, speed, and line/ . . . getty(1M)
bs: a compiler/interpreter for modest-sized programs. . . . bs(1)
floating-point/ frexp, ldexp, modf: manipulate parts of . . frexp(3C)
touch: update access and modification times of a file. . touch(1)
utime: set file access and modification times. . . . . utime(2)
tty_add, tty_kill: modify the /etc/inittab file. . . tty_add(1M)
file. yppasswdd: server for modifying yellow pages password yppasswdd(1M)
information from A/UX kernels. module_dump: dumps out . . module_dump(1M)
/ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp,/ . acctsh(1M)
profile. monitor: prepare execution . . monitor(3C)
uusub: monitor uucp network. . . . uusub(1M)

moo: guessing game. . . . . moo(6)
viewing. more: file perusal filter for CRT more(1)
functions with "optimal" cursor motion. /BSD-style screen . . curses5.0(3X)
downloading/ rcvhex: translates Motorola S-records from . . rcvhex(1)
mount: mount a file system. . . . . mount(3)
fsmount: mount an NFS file system. . . fsmount(2)
mount, umount: mount and dismount file systems. mount(1M)
mount: mount a file system. . mount(3)
mountd: NFS mount request server. . . . . mountd(1M)
file systems. mount, umount: mount and dismount mount(1M)
mountd: NFS mount request server. mountd(1M)
mtab: mounted file system table. . . mtab(4)
rmtab: remotely mounted file system table. . . rmtab(4)
showmount: show all remote mounts. . . . . showmount(1M)
mouse: mouse input device driver. . . mouse(7)
mouse: mouse input device driver. mouse(7)
mv: move or rename files. . . . . mv(1)
lseek: move read/write file pointer. . lseek(2)
the LP request scheduler and move requests. /start/stop . . lpsched(1M)
formatting a permuted index. mptx: the macro package for . mptx(5)
/erand48, lrand48, nrand48, mrand48, jrand48, srand48,/ . drand48(3C)
ms: text formatting macros. . ms(5)
operations. msgctl: message control . . . msgctl(2)
msgget: get message queue. . msgget(2)
operations. msgop, msgsnd, msgsrv: message msgop(2)
operations. msgop, msgsnd, msgsrv: message . . msgop(2)
msgop, msgsnd, msgsrv: message operations. . msgop(2)
mtab: mounted file system table. mtab(4)
select: synchronous I/O multiplexing. . . . . select(2N)
typesetting viewgraphs and/ mv: a troff macro package for mv(5)
mv: move or rename files. . . mv(1)
slides. mvt: typeset view graphs and . mvt(1)
ncstats: display kernel name cache statistics. . . . . ncstats(1M)
hosts: host name data base. . . . . hosts(4)
networks: network name data base. . . . . networks(4N)
protocols: protocol name data base. . . . . protocols(4N)
services: service name data base. . . . . services(4N)
devnm: device name. . . . . devnm(1M)
tmpnam, tempnam: create a name for a temporary file. . . tmpnam(3S)
ldgetname: retrieve symbol name for object file. . . . ldgetname(3X)
getpw: get name from UID. . . . . getpw(3C)
return value for environment name. getenv: . . . . . getenv(3C)
getlogin: get login name. . . . . getlogin(3C)
getsockname: get socket name. . . . . getsockname(2N)
nlist: get entries from name list. . . . . nlist(3C)
nm: print name list of common object file. nm(1)
logname: get login name. . . . . logname(1)
rename: change the name of a file. . . . . rename(2)
ttyname, isatty: find name of a terminal. . . . . ttyname(3C)
getpeername: get name of connected peer. . . . getpeername(2N)
domainname: set or display name of current domain system. domainname(1)

gethostname, sethostname: get/set name of current host. . . . . gethostname(2N)
hostname: set or print name of current host system. . hostname(1N)
/setdomainname: get/set name of current network domain. getdomainname(2N)
uname: print name of current system. . . . uname(1)
uname: get name of current system. . . . uname(2)
cuserid: get character login name of the user. . . . . cuserid(3S)
logname: return login name of user. . . . . logname(3X)
pwd: print working directory name. . . . . pwd(1)
named: Internet domain name server. . . . . named(1M)
bind: bind a name to a socket. . . . . bind(2N)
tty: get the terminal's name. . . . . tty(1)
server. named: Internet domain name named(1M)
nodes. pname: associate named partitions with device . pname(1M)
system. ff: list file names and statistics for a file . ff(1M)
term: conventional names for terminals. . . . . term(5)
ncheck: generate names from i-numbers. . . . ncheck(1M)
id: print user and group IDs and names. . . . . id(1)
log, alog, dlog, clog: Fortran natural logarithm intrinsic/ . . log(3F)
i-numbers. ncheck: generate names from . ncheck(1M)
cache statistics. ncstats: display kernel name . ncstats(1M)
for a document. ndx: create a subject-page index ndx(1)
/dnint, nint, idnint: Fortran nearest integer functions. . . round(3F)
character definitions for eqn and neqn. eqnchar: special . . . eqnchar(5)
for nroff. neqn: format mathematical text neqn(1)
netgroup: list of network groups. netgroup(4)
netstat: show network status. . netstat(1N)
convert values between host and network byte order. /ntohs: . byteorder(3N)
get/set name of current network domain. /setdomainname: getdomainname(2N)
setnetent, endnetent: get network entry. /getnetbyname, getnetent(3N)
nfsstat: Network File System statistics. nfsstat(1M)
/endnetgrent, innetgr: get network group entry. . . . . getnetgrent(3N)
netgroup: list of network groups. . . . . netgroup(4)
/sethostent, endhostent: get network host entry. . . . . gethostent(3N)
send ICMP ECHO_REQUEST packets to network hosts. ping: . . . . ping(1M)
lo: software loopback network interface. . . . . lo(5)
ifconfig: configure network interface parameters. ifconfig(1M)
networks: network name data base. . . . networks(4N)
file system dump across the network. rdumpfs: . . . . . rdumpfs(1M)
routed: network routing daemon. . . routed(1M)
a file system dump across the network. rrestore: restore . . rrestore(1M)
rwall: write to all users over a network running B-NET software. rwall(1M)
rwalld: network rwall server. . . . . rwalld(1M)
netstat: show network status. . . . . netstat(1N)
uusub: monitor uucp network. . . . . uusub(1M)
networks: network name data base. networks(4N)
for the mail aliases file. newaliases: rebuild the data base newaliases(1M)
text file. newform: change the format of a newform(1)
newgrp: log in to a new group. newgrp(1)
news: print local news items. . . . . news(1)
news: print local news items. . news(1)
/fetch, store, delete, firstkey, nextkey: data base subroutines. dbm(3X)

nfsd, biod: NFS daemons. . . . . nfsd(1M)
nfssvc, async_daemon: NFS daemons. . . . . nfssvc(2)
fsmount: mount an NFS file system. . . . . fsmount(2)
exports: NFS file systems being exported. exports(4)
mountd: NFS mount request server. . . mountd(1M)
nfsd, biod: NFS daemons. . . nfsd(1M)
nfs_getfh: get a file handle. . nfs_getfh(2)
statistics. nfsstat: Network File System . nfsstat(1M)
daemons. nfssvc, async_daemon: NFS . nfssvc(2)
process. nice: change priority of a . . nice(2)
priority. nice: run a command at low . nice(1)
integer functions. anint, dnint, nint, idnint: Fortran nearest . round(3F)
nl: line numbering filter. . . . nl(1)
list. nlist: get entries from name . nlist(3C)
object file. nm: print name list of common nm(1)
change current A/UX system nodename. chgnod: . . . . chgnod(1M)
named partitions with device nodes. pname: associate . . . pname(1M)
hangups. nohup: run a command immune to nohup(1)
setjmp, longjmp: non-local goto. . . . . setjmp(3C)
clock interface. nvram: nonvolatile memory/time of day nvram(7)
bitwise boolean/ and, or, xor, not, lshift, rshift: Fortran . . bool(3F)
it is from. biff: be notified if mail arrives and who biff(1)
drand48, erand48, lrand48, nrand48, mrand48, jrand48,/ . drand48(3C)
soelim: eliminate .so's from nroff input. . . . . soelim(1)
format mathematical text for nroff. neqn: . . . . . neqn(1)
terminal driving tables for nroff. nterm: . . . . . nterm(5)
tbl: format tables for nroff or troff. . . . . tbl(1)
previewing. colcrt: filter nroff output for terminal . . . colcrt(1)
nroff: text formatting language. nroff(1)
checknr: check nroff/troff files. . . . . checknr(1)
constructs. deroff: remove nroff/troff, tbl, and eqn . . . deroff(1)
for nroff. nterm: terminal driving tables nterm(5)
between host and/ htonl, htons, ntohl, ntohs: convert values . byteorder(3N)
host and/ htonl, htons, ntohl, ntohs: convert values between byteorder(3N)
null: the null file. . . . . null(7)
null: the null file. . . . . null(7)
/dodisk, lastlogin, monacct, nulladm, prctmp, prdaily,/ . . acctsh(1M)
ASCII string to floating-point number. atof: convert . . . . atof(3C)
to English. number: convert Arabic numerals number(6)
phones: remote host phone number data base. . . . . phones(4)
file. linenum: line number entries in a common object linenum(4)
/ldlinit, ldlitem: manipulate line number entries of a common object/ ldlread(3X)
ldlseek, ldnlseek: seek to line number entries of a section of a/ ldlseek(3X)
factor: factor a number. . . . . factor(1)
arithmetic: provide drill in number facts. . . . . arithmetic(6)
magic: file command's magic number file. . . . . magic(4)
strip: strip symbol and line number information from an object/ strip(1)
DARPA port to RPC program number mapper. portmap: . . portmap(1M)
version: reports version number of files. . . . . version(1)
df: report number of free disk blocks. . df(1)
string to double-precision number. strtod: convert . . . strtod(3C)

gcvt: convert floating-point number to string. ecvt, fcvt, . ecvt(3C)
nl: line numbering filter. . . . . nl(1)
distributed pseudo-random numbers. /generate uniformly drand48(3C)
parts of floating-point numbers. /ldexp, modf: manipulate frexp(3C)
install random inode generation numbers. fsirand: . . . . . fsirand(1M)
to system calls and error numbers. intro: introduction . intro(2)
file/ /endptabent, setptabent, numbptabent: get partition table getptabent(3)
number: convert Arabic numerals to English. . . . . number(6)
day clock interface. nvram: nonvolatile memory/time of nvram(7)
ldfcn: common object file access routines. . . ldfcn(3X)
conv: object file converter. . . . . conv(1)
dump: dump selected parts of an object file. . . . . dump(1)
ldopen, ldaopen: open a common object file for reading. . . . . ldopen(3X)
line number entries of a common object file function. /manipulate ldlread(3X)
ldclose, ldaclose: close a common object file. . . . . ldclose(3X)
read the file header of a common object file. ldfhread: . . . . ldfhread(3X)
retrieve symbol name for object file. ldgetname: . . . ldgetname(3X)
entries of a section of a common object file. /seek to line number ldlseek(3X)
optional file header of a common object file. /seek to the . . . ldohseek(3X)
entries of a section of a common object file. /seek to relocation ldrseek(3X)
section header of a common object file. /an indexed/named ldshread(3X)
section of a common object file. /an indexed/named ldsseek(3X)
a symbol table entry of a common object file. /the index of . . . ldtbindex(3X)
symbol table entry of a common object file. /read an indexed . ldtbread(3X)
to the symbol table of a common object file. ldtbseek: seek . . ldtbseek(3X)
line number entries in a common object file. linenum: . . . . linenum(4)
nm: print name list of common object file. . . . . nm(1)
information for a common object file. reloc: relocation . reloc(4)
section header for a common object file. scnhdr: . . . . . scnhdr(4)
line number information from an object file. /strip symbol and . strip(1)
syms: common object file symbol table format. syms(4)
a.out header for common object files. aouthdr.h: . . . aouthdr(4)
filehdr: file header for common object files. . . . . filehdr(4)
hex: translates object files. . . . . hex(1)
directories. cpset: install object files in binary . . . . cpset(1M)
ld: link editor for common object files. . . . . ld(1)
print section sizes of common object files. size: . . . . . size(1)
find ordering relation for an object library. lorder: . . . . lorder(1)
/find the printable strings in an object, or other binary file. . . strings(1)
/setgrent, endgrent, fgetgrent: obtain group file entry from a/ getgrent(3C)
od: octal dump. . . . . od(1)
od: octal dump. . . . . od(1)
login: sign on. . . . . login(1)
serial: the on-board serial ports. . . . . serial(7)
a new file or rewrite an existing one. creat: create . . . . . creat(2)
errors in the/ exterr: turn on/off the reporting of extended exterr(1M)
put: puts a file onto a remote machine. . . . put(1C)
reading. ldopen, ldaopen: open a common object file for ldopen(3X)
fopen, freopen, fdopen: open a stream. . . . . fopen(3S)
or remove an advisory lock on an open file. flock: apply . . . . flock(2)
open: open for reading or writing. . open(2)

writing. open: open for reading or . . . open(2)
seekdir, rewinddir, closedir:/ opendir, readdir, telldir, . . . directory(3)
tputs: terminal independent operation routines. /tgoto, . . termcap(3X)
bzero, ffs: bit and byte string operations. bcopy, bcmp, . . bstring(3)
rewinddir, closedir: directory operations. /telldir, seekdir, . directory(3)
memcmp, memcpy, memset: memory operations. memccpy, memchr, memory(3C)
msgctl: message control operations. . . . . msgctl(2)
msgop, msgsnd, msgsrv: message operations. . . . . msgop(2)
semctl: semaphore control operations. . . . . semctl(2)
semop: semaphore operations. . . . . semop(2)
shmctl: shared memory control operations. . . . . shmctl(2)
shmat, shmdt: shared memory operations. shmop, . . . . . shmop(2)
strspn, strcspn, strtok: string operations. /strrchr, strpbrk, . string(3C)
join: relational database operator. . . . . join(1)
dcopy: copy file systems for optimal access time. . . . . dcopy(1M)
/BSD-style screen functions with "optimal" cursor motion. . . curses5.0(3X)
curses: CRT screen handling and optimization package. . . . . curses(3X)
vector. getopt: get option letter from argument . getopt(3C)
signal facilities. sigvec: optional BSD-compatible software sigvec(2)
object/ ldohseek: seek to the optional file header of a common ldohseek(3X)
fcntl: file control options. . . . . fcntl(5)
stty: set the options for a terminal. . . . . stty(1)
getopt: parse command options. . . . . getopt(1)
setsockopt: get and set options on sockets. getsockopt, getsockopt(2N)
Fortran bitwise boolean/ and, or, xor, not, lshift, rshift: . . . bool(3F)
between host and network byte order. /ntohs: convert values . byteorder(3N)
library. lorder: find ordering relation for an object lorder(1)
make a directory, or a special or ordinary file. mknod: . . . . mknod(2)
prepare constant-width text for otroff. cw, checkcw: . . . . cw(1)
typesetting. otroff: text formatting and . . otroff(1)
cpio: copy file archives in and out. . . . . cpio(1)
connection. dial: establish an out-going terminal line . . . dial(3C)
common assembler and link editor output. a.out: . . . . . a.out(4)
fold long lines for finite-width output device. fold: . . . . fold(1)
colcrt: filter nroff output for terminal previewing. colcrt(1)
tc: troff output interpreter. . . . . tc(1)
troff: description of output language. . . . . troff(5)
list. /vsprintf: print formatted output of a varargs argument . vprintf(3S)
fprintf, sprintf: print formatted output. printf, . . . . . printf(3S)
ssp: make output single spaced. . . . . ssp(1)
/acctdusg, accton, acctwtmp: overview of accounting and/ . acct(1M)
chown, fchown: change owner and group of a file. . . chown(2)
chown, chgrp: change owner or group. . . . . chown(1)
expand files. pack, pcat, unpack: compress and pack(1)
screen handling and optimization package. curses: CRT . . . curses(3X)
index. mptx: the macro package for formatting a permuted mptx(5)
mm: macro package for formatting documents. mm(5)
viewgraphs and/ mv: a troff macro package for typesetting . . . mv(5)
interprocess communication package. ftok: standard . . . ftok(3C)
sa1, sa2: system activity report package. sadc, . . . . . sadc(1M)
spray: spray packets. . . . . spray(1M)

ping: send ICMP ECHO_REQUEST packets to network hosts. . . ping(1M)
pagesize: print system page size. . . . . pagesize(1)
ypclnt: yellow pages client interface. . . . . ypclnt(3N)
ypinit: build and install yellow pages database. . . . . ypinit(1M)
ypmake: rebuild yellow pages database. . . . . ypmake(1M)
makedbm: make a yellow pages dbm file. . . . . makedbm(1M)
server for modifying yellow pages password file. yppasswdd: yppasswdd(1M)
ypserv, ypbind: yellow pages server and binder/ . . . ypserv(1M)
change login password in yellow pages. yppasswd: . . . . . yppasswd(1)
pagesize: print system page size. pagesize(1)
terminal. 4014: paginator for the Tektronix 4014 4014(1)
kconfig: change a kernel's parameters for tuning. . . . . kconfig(1M)
configure network interface parameters. ifconfig: . . . . ifconfig(1M)
getprocess, process group, and parent process IDs. /getppid: . getpid(2)
getopt: parse command options. . . . getopt(1)
aint, dint: Fortran integer part intrinsic function. . . . aint(3F)
tail: deliver the last part of a file. . . . . tail(1)
shutdown: shut down part of a full-duplex connection. shutdown(2N)
aimag, dimag: Fortran imaginary part of complex argument. . . aimag(3F)
ypset: point ypbind at a particular server. . . . . ypset(1M)
dpme: format of disk partition map entries. . . . . dpme(4)
/setptabent, numbptabent: get partition table file entry. . . . getptabent(3)
ptab: partition table file. . . . . ptab(4)
dp: perform disk partitioning. . . . . dp(1M)
pname: associate named partitions with device nodes. . pname(1M)
dump: dump selected parts of an object file. . . . . dump(1)
frexp, ldexp, modf: manipulate parts of floating-point numbers. frexp(3C)
passwd: change login password. passwd(1)
passwd: password file. . . . passwd(4)
endpwent, fgetpwent: get password file entry. /setpwent, getpwent(3C)
putpwent: write password file entry. . . . . putpwent(3C)
passwd: password file. . . . . passwd(4)
vipw: edit the password file. . . . . vipw(1M)
server for modifying yellow pages password file. yppasswdd: . . yppasswdd(1M)
getpass: read a password. . . . . getpass(3C)
yppasswd: change login password in yellow pages. . . yppasswd(1)
passwd: change login password. . . . . passwd(1)
pwck, grpck: password/group file checkers. pwck(1M)
several files or subsequent/ paste: merge same lines of . . paste(1)
get current working directory pathname. getwd: . . . . . getwd(3)
directory. getcwd: get pathname of current working . getcwd(3C)
dirname: deliver portions of pathnames. basename, . . . basename(1)
file including aliases and paths. which: locate a program which(1)
egrep, fgrep: search a file for a pattern. grep, . . . . . grep(1)
language. awk: pattern scanning and processing awk(1)
signal. pause: suspend process until . pause(2)
files. pack, pcat, unpack: compress and expand pack(1)
process. popen, pclose: initiate pipe to/from a . popen(3S)
vax: provide truth value/ m68k, pdp11, u3b, u3b2, u3b5, u3b15, machid(1)
get name of connected peer. getpeername: . . . . . getpeername(2N)
dp: perform disk partitioning. . . dp(1M)

mesg: permit or deny messages. . . mesg(1)
macro package for formatting a permuted index. mptx: the . . mptx(5)
ptx: make permuted index. . . . . ptx(1)
format. acct: per-process accounting file . . acct(4)
acctcms: command summary from per-process accounting records. acctcms(1M)
sys_nerr: system error messages. perror, errno, sys_errlist, . . perror(3C)
more: file perusal filter for CRT viewing. more(1)
terminals. pg: file perusal filter for soft-copy . . pg(1)
soft-copy terminals. pg: file perusal filter for . . . pg(1)
phones: remote host phone number data base. . . phones(4)
data base. phones: remote host phone number phones(4)
for the Autologic APS-5 phototypesetter. /Postprocessor daps(1)
physical addresses. phys: allow a process to access phys(2)
phys: allow a process to access physical addresses. . . . . phys(2)
graphs. grap: pic preprocessor for drawing . grap(1)
drawing pictures. pic: troff preprocessor for . . pic(1)
troff preprocessor for drawing pictures. pic: . . . . . pic(1)
split: split a file into pieces. . . . . split(1)
packets to network hosts. ping: send ICMP ECHO_REQUEST ping(1M)
channel. pipe: create an interprocess . pipe(2)
tee: pipe fitting. . . . . tee(1)
popen, pclose: initiate pipe to/from a process. . . . popen(3S)
fish: play "Go Fish". . . . . fish(6)
life: play the game of life. . . . . life(6)
worm: play the growing worm game. worm(6)
data in memory. plock: lock process, text, or . plock(2)
plot: graphics interface. . . . plot(4)
subroutines. plot: graphics interface . . . plot(3X)
with device nodes. pname: associate named partitions pname(1M)
server. ypset: point ypbind at a particular . . ypset(1M)
rewind, ftell: reposition a file pointer in a stream. fseek, . . fseek(3S)
lseek: move read/write file pointer. . . . . lseek(2)
to/from a process. popen, pclose: initiate pipe . . popen(3S)
mapper. portmap: DARPA port to RPC program number . portmap(1M)
data base of terminal types by port. ttytype: . . . . . ttytype(4)
and library maintainer for portable archives. ar: archive . ar(1)
basename, dirname: deliver portions of pathnames. . . . basename(1)
program number mapper. portmap: DARPA port to RPC portmap(1M)
serial: the on-board serial ports. . . . . serial(7)
dim, ddim, idim: Fortran positive difference intrinsic/ . dim(3F)
banner: make posters. . . . . banner(1)
daiw: Apple ImageWriter II troff postprocessor filter. . . . . daiw(1)
APS-5 phototypesetter. daps: Postprocessor for the Autologic daps(1)
format. afm: Adobe PostScript font metrics file . . afm(7)
postscript: print file format. . postscript(4)
/TranScript spooler filters for PostScript printers. . . . . transcript(1M)
logarithm,/ exp, log, log10, pow, sqrt: exponential, . . . exp(3M)
powerdown: power down the system. . . . powerdown(1M)
/sqrt: exponential, logarithm, power, square root functions. . exp(3M)
powerdown: power down the system. powerdown(1M)
shell/ brc, bcheckrc, rc, powerfail: system initialization brc(1M)

pr: print files. . . . . pr(1)
/lastlogin, monacct, nulladm, prctmp, prdaily, prtacct,/ . . acctsh(1M)
/monacct, nulladm, prctmp, prdaily, prtacct, shutacct,/ . . acctsh(1M)
function. dprod: Fortran double precision product intrinsic . . dprod(3F)
otroff. cw, checkcw: prepare constant-width text for cw(1)
monitor: prepare execution profile. . . monitor(3C)
iwprep: prepare troff description files. . iwprep(1)
cpp: the C language preprocessor. . . . . cpp(1)
grap: pic preprocessor for drawing graphs. grap(1)
pictures. pic: troff preprocessor for drawing . . pic(1)
reject: prevent LP requests. . . . . reject(1M)
filter nroff output for terminal previewing. colcrt: . . . . . colcrt(1)
unget: undo a previous get of an SCCS file. . unget(1)
types: primitive system data types. . types(5)
interesting, adage. fortune: print a random, hopefully . . fortune(6)
prs: print an SCCS file. . . . . prs(1)
date: print and set the date. . . . . date(1)
cal: print calendar. . . . . cal(1)
a file. sum: print checksum and block count of sum(1)
activity. sact: print current SCCS file editing sact(1)
whoami: print effective current user ID. whoami(1)
man: print entries in this manual. . man(1)
postscript: print file format. . . . . postscript(4)
cat: concatenate and print files. . . . . cat(1)
pr: print files. . . . . pr(1)
iw2: Apple ImageWriter print filter. . . . . iw2(1)
vprintf, vfprintf, vsprintf: print formatted output of a/ . . vprintf(3S)
printf, fprintf, sprintf: print formatted output. . . . printf(3S)
fpr: print Fortran file. . . . . fpr(1)
system. hostid: set or print identifier of current host . hostid(1N)
banner7: print large banner on printer. . banner7(1)
lav: print load average statistics. . lav(1)
news: print local news items. . . . news(1)
lpstat: print LP status information. . lpstat(1)
file. nm: print name list of common object nm(1)
system. hostname: set or print name of current host . . hostname(1N)
uname: print name of current system. . uname(1)
acctcom: search and print process accounting file(s). acctcom(1M)
object files. size: print section sizes of common size(1)
pstat: print system facts. . . . . pstat(1M)
pagesize: print system page size. . . . pagesize(1)
printenv: print the current environment. printenv(1)
keys from a YP map. ypmatch: print the value of one or more ypmatch(1)
names. id: print user and group IDs and . id(1)
ypcat: print values in a YP data base. ypcat(1)
for diction. diction, explain: print wordy sentences; thesaurus diction(1)
pwd: print working directory name. pwd(1)
or other/ strings: find the printable strings in an object, . strings(1)
environment. printenv: print the current . . printenv(1)
banner7: print large banner on printer. . . . . banner7(1)
send/cancel requests to a line printer. lp, cancel: . . . . . lp(1)

lpr: send requests to a line printer. . . . . lpr(1)
to a POSTSCRIPT printer. psroff: troff . . . . psroff(1)
lprm: remove jobs from the line printer spooling queue. . . . lprm(1)
disable: enable/disable LP printers. enable, . . . . . enable(1)
spooler filters for PostScript printers. /TranScript . . . . transcript(1M)
formatted output. printf, fprintf, sprintf: print . . printf(3S)
the mm macros. mm: prints documents formatted with mm(1)
nice: run a command at low priority. . . . . nice(1)
nice: change priority of a process. . . . . nice(2)
rpc: library routines for remote procedure calls. . . . . rpc(3N)
boot: startup procedures. . . . . boot(8)
/startup, turnacct: shell procedures for accounting. . . acctsh(1M)
errors. errpt: process a report of logged . . errpt(1M)
acct: enable or disable process accounting. . . . . acct(2)
acctprc1, acctprc2: process accounting. . . . . acctprc(1M)
acctcom: search and print process accounting file(s). . . acctcom(1M)
times: get process and child process times. times(2)
setcompat, getcompat: set or get process compatibility mode. . setcompat(2)
init, telinit: process control initialization. . init(1M)
timex: time a command; report process data and system activity. timex(1)
exit, _exit: terminate process. . . . . exit(2)
fork: create a new process. . . . . fork(2)
/getpgrp, getppid: get process, process group, and parent process/ getpid(2)
setpgrp: set process group ID. . . . . setpgrp(2)
killpg: send signal to a process group. . . . . killpg(3N)
process group, and parent process IDs. /get process, . . getpid(2)
inittab: script for the init process. . . . . inittab(4)
kill: terminate a process. . . . . kill(1)
nice: change priority of a process. . . . . nice(2)
kill: send a signal to a process or a group of processes. kill(2)
pclose: initiate pipe to/from a process. popen, . . . . . popen(3S)
getpid, getpgrp, getppid: get process, process group, and/ . getpid(2)
ps: report process status. . . . . ps(1)
plock: lock process, text, or data in memory. plock(2)
times: get process and child process times. . . . . times(2)
addresses. phys: allow a process to access physical . . phys(2)
wait: wait for child process to stop or terminate. . wait(2)
wait3: wait for child process to stop or terminate. . wait3(2N)
ptrace: process trace. . . . . ptrace(2)
pause: suspend process until signal. . . . . pause(2)
signal to a process or a group of processes. kill: send a . . . . kill(2)
killall: kill all active processes. . . . . killall(1M)
structure. fuser: identify processes using a file or file . fuser(1M)
yellow pages server and binder processes. ypserv, ypbind: . . ypserv(1M)
awk: pattern scanning and processing language. . . . . awk(1)
shutdown: terminate all processing. . . . . shutdown(1M)
mailx: interactive message processing system. . . . . mailx(1)
m4: macro processor. . . . . m4(1)
vax: provide truth value about processor type. /u3b5, u3b15, machid(1)
alarm: set a process's alarm clock. . . . alarm(2)
of macro files. macref: produce cross-reference listing macref(1)

dprod: Fortran double precision product intrinsic function. . . dprod(3F)
prof: display profile data. . . prof(1)
prof: profile within a function. prof(5)
profil: execution time profile. . profil(2)
prof: display profile data. . . . . prof(1)
monitor: prepare execution profile. . . . . monitor(3C)
profil: execution time profile. . . . . profil(2)
environment at login time. profile: setting up an . . . . profile(4)
prof: profile within a function. . . prof(5)
abort: terminate Fortran program. . . . . abort(3F)
assert: verify program assertion. . . . . assert(3X)
cb: C program beautifier. . . . . cb(1)
lint: a C program checker. . . . . lint(1)
cxref: generate C program cross-reference. . . cxref(1)
maintain a tags file for a C program. ctags: . . . . . ctags(1)
ctrace: C program debugger. . . . . ctrace(1)
etext, edata: last locations in program. end, . . . . . end(3C)
and paths. which: locate a program file including aliases . which(1)
finger: user information lookup program. . . . . finger(1)
ftp: ARPANET file transfer program. . . . . ftp(1N)
lpq: spool queue examination program. . . . . lpq(1)
portmap: DARPA port to RPC program number mapper. . . portmap(1M)
rdist: remote file distribution program. . . . . rdist(1)
sdiff: side-by-side difference program. . . . . sdiff(1)
indent: indent and format C program source. . . . . indent(1)
tftp: trivial file transfer program. . . . . tftp(1C)
screen color. scr_color: program to change the terminal's scr_color(1)
units: conversion program. . . . . units(1)
source, binary, and/or manual for program. whereis: locate . . whereis(1)
ksh: Korn shell, a command programming language. . . . ksh(1)
standard/restricted command programming language. /shell, sh(1)
for modest-sized programs. /a compiler/interpreter bs(1)
tasks. lex: generate programs for simple lexical . lex(1)
update, and regenerate groups of programs. make: maintain, . . make(1)
xstr: extract strings from C programs to implement shared/ xstr(1)
and clean up after A/UX Toolbox programs. /set up for . . . . toolboxd(1M)
yppush: force propagation of a changed YP map. yppush(1M)
arp: Address Resolution Protocol. . . . . arp(5P)
setprotoent, endprotoent: get protocol entry. /getprotobyname, getprotoent(3N)
inet: Internet protocol family. . . . . inet(5f)
ip: Internet Protocol. . . . . ip(5P)
protocols: protocol name data base. . . protocols(4N)
DARPA Internet File Transfer Protocol server. ftpd: . . . . ftpd(1M)
telnetd: DARPA TELNET protocol server. . . . . telnetd(1M)
DARPA Trivial File Transfer Protocol server. tftpd: . . . tftpd(1M)
Internet Transmission Control Protocol. tcp: . . . . . tcp(5P)
user interface to the TELNET protocol. telnet: . . . . . telnet(1C)
trpt: transliterate protocol trace. . . . . trpt(1M)
udp: Internet User Datagram Protocol. . . . . udp(5P)
base. protocols: protocol name data protocols(4N)
arithmetic: provide drill in number facts. . arithmetic(6)

for reading or writing. locking: provide exclusive file regions . locking(2)
u3b, u3b2, u3b5, u3b15, vax: provide truth value about/ /pdp11, machid(1)
true, false: provide truth values. . . . . true(1)
prs: print an SCCS file. . . . prs(1)
/nulladm, prctmp, prdaily, prtacct, shutacct, startup,/ . . acctsh(1M)
ps: report process status. . . . ps(1)
/troff intermediate format to POSTSCRIPT/ . . . . . psdit(1)
printer. psroff: troff to a POSTSCRIPT . . . . . psroff(1)
spooler/ pscomm, psrv, pstext, psbanner, psinterface: TranScript transcript(1M)
psinterface: TranScript spooler/ pscomm, psrv, pstext, psbanner, transcript(1M)
format to/ psdit: convert troff intermediate psdit(1)
pty: pseudo terminal driver. . . . pty(7)
sxt: pseudo-device driver. . . . . sxt(7)
generate uniformly distributed pseudo-random numbers. /lcong48: drand48(3C)
pscomm, psrv, pstext, psbanner, psinterface: TranScript spooler/ transcript(1M)
POSTSCRIPT/ psroff: troff to a . . . . . psroff(1)
psinterface: TranScript/ pscomm, psrv, pstext, psbanner, . . . transcript(1M)
pstat: print system facts. . . . pstat(1M)
TranScript spooler/ pscomm, psrv, pstext, psbanner, psinterface: . transcript(1M)
ptab: partition table file. . . . ptab(4)
ptrace: process trace. . . . . ptrace(2)
ptx: make permuted index. . . ptx(1)
pty: pseudo terminal driver. . pty(7)
copy. uuto, uupick: public UNIX-to-UNIX system file uuto(1C)
stream. ungetc: push character back into input ungetc(3S)
line_push: routine used to push streams line disciplines. . line_push(3)
line_sane: push streams line disciplines. . line_sane(1M)
puts, fputs: put a string on a stream. . . . puts(3S)
putc, putchar, fputc, putw: put character or word on a/ . . putc(3S)
machine. put: puts a file onto a remote . put(1C)
character or word on a stream. putc, putchar, fputc, putw: put putc(3S)
character or word on a/ putc, putchar, fputc, putw: put . . . putc(3S)
environment. putenv: change or add value to putenv(3C)
entry. putpwent: write password file . putpwent(3C)
machine. put: puts a file onto a remote . . . put(1C)
stream. puts, fputs: put a string on a . puts(3S)
getutent, getutid, getutline, pututline, setutent, endutent,/ . getut(3C)
stream. putc, putchar, fputc, putw: put character or word on a putc(3S)
checkers. pwck, grpck: password/group file pwck(1M)
name. pwd: print working directory . pwd(1)
qsort: quicker sort. . . . . qsort(3C)
tput: query terminfo database. . . tput(1)
lpq: spool queue examination program. . lpq(1)
insert/remove element from a queue. insque, remque: . . . insque(3N)
from the line printer spooling queue. lprm: remove jobs . . lprm(1)
msgget: get message queue. . . . . msgget(2)
memory ID. ipcrm: remove message queue, semaphore set, or shared ipcrm(1)
qsort: quicker sort. . . . . qsort(3C)
quiz: test your knowledge. . . quiz(6)
rain: animated raindrops display. rain(6)
rain: animated raindrops display. . . . . rain(6)

random-number/ irand, srand, rand: Fortran uniform . . . . rand(3F)
generator. rand, srand: simple random-number rand(3C)
adage. fortune: print a random, hopefully interesting, fortune(6)
fsirand: install random inode generation numbers. fsirand(1M)
rand, srand: simple random-number generator. . . rand(3C)
srand, rand: Fortran uniform random-number generator. irand, rand(3F)
initialization/ brc, bcheckrc, rc, powerfail: system . . . . brc(1M)
routines for returning a stream/ rcmd, rresvport, ruserok: . . rcmd(3N)
rcp: remote file copy. . . . . rcp(1C)
S-records from downloading into/ rcvhex: translates Motorola . rcvhex(1)
program. rdist: remote file distribution . rdist(1)
the network. rdumpfs: file system dump across rdumpfs(1M)
disk. mfs: read a Macintosh flat file system mfs(1)
getpass: read a password. . . . . getpass(3C)
entry of a common/ ldtbread: read an indexed symbol table . ldtbread(3X)
header of a/ ldshread, ldnshread: read an indexed/named section ldshread(3X)
address.. etheraddr: read an interface's Ethernet . etheraddr(1M)
read, readv: read from file. . . . . read(2)
rmail: send mail to users or read mail. mail, . . . . . mail(1)
line: read one line. . . . . line(1)
read, readv: read from file. . . read(2)
member of an archive/ ldahread: read the archive header of a . ldahread(3X)
object file. ldfhread: read the file header of a common ldfhread(3X)
readlink: read value of a symbolic link. . readlink(2)
rewinddir, closedir:/ opendir, readdir, telldir, seekdir, . . . directory(3)
open a common object file for reading. ldopen, ldaopen: . . ldopen(3X)
exclusive file regions for reading or writing. /provide . locking(2)
open: open for reading or writing. . . . . open(2)
symbolic link. readlink: read value of a . . . readlink(2)
read, readv: read from file. . . . . read(2)
lseek: move read/write file pointer. . . . lseek(2)
setregid: set real and effective group ID. . setregid(2)
setreuid: set real and effective user ID's. . setreuid(2)
dcmplx, ichar,/ int, ifix, idint, real, float, sngl, dble, cmplx, . ftype(3F)
/get real user, effective user, real group, and effective group/ getuid(2)
/geteuid, getgid, getegid: get real user, effective user, real/ . getuid(2)
memory allocator. malloc, free, realloc, calloc, cfree: main . . malloc(3C)
mallinfo: fast/ malloc, free, realloc, calloc, mallopt, . . . malloc(3X)
reboot: reboot the system. . . reboot(1M)
reboot: reboot the system. . . reboot(2)
reboot: reboot the system. . . . . reboot(1M)
reboot: reboot the system. . . . . reboot(2)
mail aliases file. newaliases: rebuild the data base for the . newaliases(1M)
ypmake: rebuild yellow pages database. ypmake(1M)
signal: specify what to do upon receipt of a signal. . . . . signal(3)
signal: specify Fortran action on receipt of a system signal. . . signal(3F)
recv, recvfrom, recvmsg: receive a message from a socket. recv(2N)
lockf: record locking on files. . . . lockf(3C)
from per-process accounting records. /command summary . acctcms(1M)
errdead: extract error records from dumpfs. . . . . errdead(1M)
manipulate connect accounting records. fwtmp, wtmpfix: . . fwtmp(1M)

frec: recover files from a backup tape. frec(1M)
a message from a socket. recv, recvfrom, recvmsg: receive recv(2N)
message from a socket. recv, recvfrom, recvmsg: receive a . recv(2N)
socket. recv, recvfrom, recvmsg: receive a message from a recv(2N)
ed, red: text editor. . . . . ed(1)
references in documents. refer: find and insert literature refer(1)
/index for a bibliography, find references in a bibliography. . lookbib(1)
refer: find and insert literature references in documents. . . refer(1)
execute a regular expression. regcmp, regex: compile and . regcmp(3X)
compile. regcmp: regular expression . . regcmp(1)
make: maintain, update, and regenerate groups of programs. make(1)
regular expression. regcmp, regex: compile and execute a . regcmp(3X)
compile and match routines. regexp: regular expression . . regexp(5)
locking: provide exclusive file regions for reading or writing. locking(2)
match routines. regexp: regular expression compile and regexp(5)
regcmp: regular expression compile. . regcmp(1)
regex: compile and execute a regular expression. regcmp, . regcmp(3X)
files. comm: select or reject lines common to two sorted comm(1)
reject: prevent LP requests. . reject(1M)
lorder: find ordering relation for an object library. . lorder(1)
join: relational database operator. . join(1)
for/ sigpause: atomically release blocked signals and wait sigpause(2)
a common object file. reloc: relocation information for reloc(4)
of a/ ldrseek, ldnrseek: seek to relocation entries of a section . ldrseek(3X)
common object file. reloc: relocation information for a . reloc(4)
ceil, fmod, fabs: floor, ceiling, remainder, absolute value/ floor, floor(3M)
mod, amod, dmod: Fortran remaindering intrinsic functions. mod(3F)
leave. leave: remind you when you have to leave(1)
calendar: reminder service. . . . . calendar(1)
for returning a stream to a remote command. /routines . rcmd(3N)
rexec: return stream to a remote command. . . . . rexec(3N)
rexecd: remote execution server. . . . rexecd(1M)
rcp: remote file copy. . . . . rcp(1C)
rdist: remote file distribution program. rdist(1)
remote: remote host description file. . remote(4)
base. phones: remote host phone number data phones(4)
uusend: send a file to a remote host. . . . . uusend(1C)
rlogin: remote login. . . . . rlogin(1N)
rlogind: remote login server. . . . rlogind(1M)
put: puts a file onto a remote machine. . . . . put(1C)
take: takes a file from a remote machine. . . . . take(1C)
showmount: show all remote mounts. . . . . showmount(1M)
rpc: library routines for remote procedure calls. . . . rpc(3N)
file. remote: remote host description remote(4)
remsh: remote shell. . . . . remsh(1N)
remshd: remote shell server. . . . . remshd(1M)
tip: connect to a remote system. . . . . tip(1C)
ct: spawn getty to a remote terminal. . . . . ct(1C)
talkd: remote user communication server. talkd(1M)
table. rmtab: remotely mounted file system . rmtab(4)
rmdel: remove a delta from an SCCS file. rmdel(1)

rmdir: remove a directory file. . . . rmdir(2)
unmount: remove a file system. . . . . unmount(2)
open file. flock: apply or remove an advisory lock on an flock(2)
colrm: remove columns from a file. . colrm(1)
unlink: remove directory entry. . . . unlink(2)
rm, rmdir: remove files or directories. . . rm(1)
spooling queue. lprm: remove jobs from the line printer lprm(1)
set, or shared memory ID. ipcrm: remove message queue, semaphore ipcrm(1)
constructs. deroff: remove nroff/troff, tbl, and eqn deroff(1)
directories. dev_kill: remove special devices from . dev_kill(1M)
from a queue. insque, remque: insert/remove element insque(3N)
remsh: remote shell. . . . . remsh(1N)
remshd: remote shell server. . remshd(1M)
file. rename: change the name of a rename(2)
mv: move or rename files. . . . . mv(1)
standalone file system repair. autorecovery: . . . . autorecovery(8)
consistency check and interactive repair. fsck: file system . . . fsck(1M)
uniq: report repeated lines in a file. . . . uniq(1)
yes: be repetitively affirmative. . . . yes(1)
clock: report CPU time used. . . . clock(3C)
facilities status. ipcs: report interprocess communication ipcs(1)
blocks. df: report number of free disk . . df(1)
errpt: process a report of logged errors. . . . errpt(1M)
in a file. freq: report on character frequencies freq(1)
sadc, sa1, sa2: system activity report package. . . . . sadc(1M)
activity. timex: time a command; report process data and system timex(1)
ps: report process status. . . . . ps(1)
uniq: report repeated lines in a file. . uniq(1)
rpcinfo: report RPC information. . . . rpcinfo(1M)
sar: system activity reporter. . . . . sar(1)
the/ exterr: turn on/off the reporting of extended errors in exterr(1M)
version: reports version number of files. version(1)
stream. fseek, rewind, ftell: reposition a file pointer in a . fseek(3S)
routines for external data representation. xdr: library . . xdr(3N)
/lpshut, lpmove: start/stop the LP request scheduler and move/ . lpsched(1M)
mountd: NFS mount request server. . . . . mountd(1M)
accept: allow LP requests. . . . . accept(1M)
the LP request scheduler and move requests. /lpmove: start/stop . lpsched(1M)
reject: prevent LP requests. . . . . reject(1M)
lp, cancel: send/cancel requests to a line printer. . . lp(1)
lpr: send requests to a line printer. . . lpr(1)
bits to a sensible state. tset, reset: set or reset the teletype . tset(1)
sensible/ tset, reset: set or reset the teletype bits to a . . tset(1)
resolver/ res_mkquery, res_send, res_init, dn_comp, dn_expand: resolver(3N)
dn_comp, dn_expand: resolver/ res_mkquery, res_send, res_init, resolver(3N)
arp: Address Resolution Protocol. . . . . arp(5P)
resolver: resolver configuration file. . . resolver(4)
file. resolver: resolver configuration resolver(4)
res_init, dn_comp, dn_expand: resolver routines. /res_send, . resolver(3N)
derez: decompiles a resource file. . . . . derez(1)
type and creator of a Macintosh resource file. settc: set the . . settc(1)

rez: compile resources. . . . . rez(1)
dn_expand: resolver/ res_mkquery, res_send, res_init, dn_comp, . resolver(3N)
the network. rrestore: restore a file system dump across rrestore(1M)
restore. restore: incremental file system restore(1M)
restore: incremental file system restore. . . . . restore(1M)
file. ldgetname: retrieve symbol name for object ldgetname(3X)
iargc: return command line arguments. iargc(3F)
argument. getarg: return Fortran command-line . getarg(3F)
variable. getenv: return Fortran environment . . getenv(3F)
mclock: return Fortran time accounting. mclock(3F)
abs: return integer absolute value. . abs(3C)
len: return length of Fortran string. len(3F)
substring. index: return location of Fortran . . index(3F)
logname: return login name of user. . . logname(3X)
command. rexec: return stream to a remote . . rexec(3N)
name. getenv: return value for environment . getenv(3C)
stat: data returned by stat system call. . stat(5)
/rresvport, ruserok: routines for returning a stream to a remote/ rcmd(3N)
configuration information. uvar: returns system-specific . . . uvar(2)
rev: reverse lines of a file. . . rev(1)
col: filter reverse linefeeds. . . . . col(1)
rev: reverse lines of a file. . . . . rev(1)
pointer in a stream. fseek, rewind, ftell: reposition a file . fseek(3S)
/readdir, telldir, seekdir, rewinddir, closedir: directory/ directory(3)
creat: create a new file or rewrite an existing one. . . . creat(2)
command. rexec: return stream to a remote rexec(3N)
rexecd: remote execution server. rexecd(1M)
rez: compile resources. . . . rez(1)
rlogin: remote login. . . . . rlogin(1N)
rlogind: remote login server. . rlogind(1M)
directories. rm, rmdir: remove files or . . rm(1)
mail. mail, rmail: send mail to users or read mail(1)
SCCS file. rmdel: remove a delta from an rmdel(1)
rmdir: remove a directory file. rmdir(2)
directories. rm, rmdir: remove files or . . . . rm(1)
system table. rmtab: remotely mounted file . rmtab(4)
escape from the automatic robots. autorobots: . . . . . autorobots(6)
chase: try to escape the killer robots. . . . . chase(6)
robots: escape from the robots. robots(6)
robots: escape from the robots. . . . . robots(6)
database. roffbib: run off bibliographic . roffbib(1)
slots: ROM library functions. . . . slots(3X)
chroot: change root directory. . . . . chroot(2)
chroot: change root directory for a command. chroot(1M)
logarithm, power, square root functions. /exponential, . exp(3M)
dsqrt, csqrt: Fortran square root intrinsic function. sqrt, . sqrt(3F)
routing tables. route: manually manipulate the route(1M)
routed: network routing daemon. routed(1M)
disciplines. line_push: routine used to push streams line line_push(3)
representation. xdr: library routines for external data . . xdr(3N)
calls. rpc: library routines for remote procedure rpc(3N)

to a/ rcmd, rresvport, ruserok: routines for returning a stream rcmd(3N)
Internet address manipulation routines. /inet_netof: . . . . inet(3N)
ldfcn: common object file access routines. . . . . ldfcn(3X)
expression compile and match routines. regexp: regular . . regexp(5)
dn_comp, dn_expand: resolver routines. /res_send, res_init, . resolver(3N)
terminal independent operation routines. /tgetstr, tgoto, tputs: termcap(3X)
routed: network routing daemon. . . . . routed(1M)
route: manually manipulate the routing tables. . . . . route(1M)
rpcinfo: report RPC information. . . . . rpcinfo(1M)
procedure calls. rpc: library routines for remote rpc(3N)
portmap: DARPA port to RPC program number mapper. portmap(1M)
host status of local machines (RPC version). rup: show . . rup(1C)
who's logged in on local machines (RPC version). rusers: . . . rusers(1N)
rpcinfo: report RPC information. rpcinfo(1M)
dump across the network. rrestore: restore a file system . rrestore(1M)
returning a stream to a/ rcmd, rresvport, ruserok: routines for rcmd(3N)
standard/restricted command/ sh, rsh: Bourne shell, . . . . . sh(1)
and, or, xor, not, lshift, rshift: Fortran bitwise boolean/ bool(3F)
rstatd: kernel statistics server. . rstatd(1M)
nice: run a command at low priority. nice(1)
nohup: run a command immune to hangups. nohup(1)
runacct: run daily accounting. . . . . runacct(1M)
roffbib: run off bibliographic database. roffbib(1)
runacct: run daily accounting. runacct(1M)
write to all users over a network running B-NET software. rwall: rwall(1M)
machines (RPC version). rup: show host status of local . rup(1C)
local machines. ruptime: show host status of . ruptime(1N)
stream to a/ rcmd, rresvport, ruserok: routines for returning a rcmd(3N)
rusersd: rusers server. . . . . rusersd(1M)
machines (RPC version). rusers: who's logged in on local rusers(1N)
rusersd: rusers server. . . . . rusersd(1M)
rwalld: network rwall server. . . . . rwalld(1M)
network running B-NET software. rwall: write to all users over a rwall(1M)
rwalld: network rwall server. . rwalld(1M)
machines. rwho: who's logged in on local rwho(1N)
rwhod: system status server. . rwhod(1M)
package. sadc, sa1, sa2: system activity report sadc(1M)
package. sadc, sa1, sa2: system activity report . . sadc(1M)
editing activity. sact: print current SCCS file . sact(1)
report package. sadc, sa1, sa2: system activity sadc(1M)
sag: system activity graph. . . sag(1G)
sar: system activity reporter. . sar(1)
the standalone environment. sash: a command interpreter for sash(8)
allocation. brk, sbrk: change data segment space brk(2)
scandir: scan a directory. . . . . scandir(3)
scandir: scan a directory. . . scandir(3)
formatted input. scanf, fscanf, sscanf: convert . scanf(3S)
bfs: big file scanner. . . . . bfs(1)
awk: pattern scanning and processing language. awk(1)
change the delta commentary of an SCCS delta. cdc: . . . . . cdc(1)
comb: combine SCCS deltas. . . . . comb(1)

make a delta (change) to an SCCS file. delta: . . . . . delta(1)
sact: print current SCCS file editing activity. . . sact(1)
get: get a version of an SCCS file. . . . . get(1)
prs: print an SCCS file. . . . . prs(1)
rmdel: remove a delta from an SCCS file. . . . . rmdel(1)
compare two versions of an SCCS file. sccsdiff: . . . . sccsdiff(1)
sccsfile: format of an SCCS file. . . . . sccsfile(4)
unget: undo a previous get of an SCCS file. . . . . unget(1)
val: validate SCCS file. . . . . val(1)
admin: create and administer SCCS files. . . . . admin(1)
what: identify SCCS files. . . . . what(1)
subsystem. sccs: front end for the SCCS . sccs(1)
help: ask for help in using SCCS. . . . . help(1)
sccs: front end for the SCCS subsystem. . . . . sccs(1)
an SCCS file. sccsdiff: compare two versions of sccsdiff(1)
sccsfile: format of an SCCS file. sccsfile(4)
/lpmove: start/stop the LP request scheduler and move requests. . lpsched(1M)
common object file. scnhdr: section header for a . scnhdr(4)
terminal's screen color. scr_color: program to change the scr_color(1)
clear: clear terminal screen. . . . . clear(1)
program to change the terminal's screen color. scr_color: . . . scr_color(1)
cursor/ curses5.0: BSD-style screen functions with "optimal" curses5.0(3X)
package. curses: CRT screen handling and optimization curses(3X)
twinkle: twinkle stars on the screen. . . . . twinkle(6)
editor. vi, view, vedit: screen-oriented (visual) display vi(1)
inittab: script for the init process. . . inittab(4)
terminal session. script: make typescript of . . script(1)
system initialization shell scripts. /rc, powerfail: . . . brc(1M)
sdb: symbolic debugger. . . . sdb(1)
program. sdiff: side-by-side difference . sdiff(1)
grep, egrep, fgrep: search a file for a pattern. . . grep(1)
bsearch: binary search a sorted table. . . . . bsearch(3C)
accounting file(s). acctcom: search and print process . . . acctcom(1M)
lsearch, lfind: linear search and update. . . . . lsearch(3C)
hcreate, hdestroy: manage hash search tables. hsearch, . . . hsearch(3C)
tdelete, twalk: manage binary search trees. tsearch, tfind, . . tsearch(3C)
object file. scnhdr: section header for a common . scnhdr(4)
/ldnshread: read an indexed/named section header of a common object/ ldshread(3X)
/seek to line number entries of a section of a common object file. ldlseek(3X)
/seek to relocation entries of a section of a common object file. ldrseek(3X)
/seek to an indexed/named section of a common object file. ldsseek(3X)
files. size: print section sizes of common object size(1)
sed: stream editor. . . . . sed(1)
/mrand48, jrand48, srand48, seed48, lcong48: generate/ . . drand48(3C)
of a common/ ldsseek, ldnsseek: seek to an indexed/named section ldsseek(3X)
section of a/ ldlseek, ldnlseek: seek to line number entries of a ldlseek(3X)
section of a/ ldrseek, ldnrseek: seek to relocation entries of a . ldrseek(3X)
of a common object/ ldohseek: seek to the optional file header ldohseek(3X)
common object file. ldtbseek: seek to the symbol table of a . ldtbseek(3X)
opendir, readdir, telldir, seekdir, rewinddir, closedir:/ . directory(3)
shmget: get shared memory segment. . . . . shmget(2)

brk, sbrk: change data segment space allocation. . . brk(2)
two sorted files. comm: select or reject lines common to comm(1)
multiplexing. select: synchronous I/O . . . select(2N)
greek: select terminal filter. . . . . greek(1)
file. cut: cut out selected fields of each line of a cut(1)
dump: dump selected parts of an object file. dump(1)
semctl: semaphore control operations. semctl(2)
semop: semaphore operations. . . . semop(2)
ID. ipcrm: remove message queue, semaphore set, or shared memory ipcrm(1)
semget: get set of semaphores. . . . . semget(2)
operations. semctl: semaphore control . . semctl(2)
semget: get set of semaphores. semget(2)
semop: semaphore operations. semop(2)
uusend: send a file to a remote host. . uusend(1C)
send, sendto, sendmsg: send a message from a socket. send(2N)
group of processes. kill: send a signal to a process or a kill(2)
network hosts. ping: send ICMP ECHO_REQUEST packets to ping(1M)
sendmail: send mail over the Internet. . sendmail(1M)
mail, rmail: send mail to users or read mail. mail(1)
lpr: send requests to a line printer. lpr(1)
message from a socket. send, sendto, sendmsg: send a send(2N)
killpg: send signal to a process group. killpg(3N)
printer. lp, cancel: send/cancel requests to a line . lp(1)
aliases: aliases file for sendmail. . . . . aliases(4)
Internet. sendmail: send mail over the . sendmail(1M)
socket. send, sendto, sendmsg: send a message from a send(2N)
from a socket. send, sendto, sendmsg: send a message send(2N)
or reset the teletype bits to a sensible state. tset, reset: set . tset(1)
diction, explain: print wordy sentences; thesaurus for diction. diction(1)
dialup: modem escape sequence file. . . . . dialup(4)
serial: the on-board serial ports. . . . . serial(7)
ports. serial: the on-board serial . . serial(7)
ypserv, ypbind: yellow pages server and binder processes. . ypserv(1M)
comsat: biff(1) server. . . . . comsat(1M)
servers: Inet server data base. . . . . servers(4)
password file. yppasswdd: server for modifying yellow pages yppasswdd(1M)
Internet File Transfer Protocol server. ftpd: DARPA . . . . ftpd(1M)
version of a YP map is at a YP server host. yppoll: what . . yppoll(1M)
mountd: NFS mount request server. . . . . mountd(1M)
named: Internet domain name server. . . . . named(1M)
ypwhich: which host is the YP server or map master?. . . . ypwhich(1)
remshd: remote shell server. . . . . remshd(1M)
rexecd: remote execution server. . . . . rexecd(1M)
rlogind: remote login server. . . . . rlogind(1M)
rstatd: kernel statistics server. . . . . rstatd(1M)
rusersd: rusers server. . . . . rusersd(1M)
rwalld: network rwall server. . . . . rwalld(1M)
rwhod: system status server. . . . . rwhod(1M)
sprayd: spray server. . . . . sprayd(1M)
talkd: remote user communication server. . . . . talkd(1M)
telnetd: DARPA TELNET protocol server. . . . . telnetd(1M)

Trivial File Transfer Protocol server. tftpd: DARPA . . . tftpd(1M)
transfer a YP map from some YP server to here. ypxfr: . . . . ypxfr(1M)
point ypbind at a particular server. ypset: . . . . . ypset(1M)
servers: Inet server data base. . servers(4)
calendar: reminder service. . . . . calendar(1)
setservent, endservent: get service entry. /getservbyname, getservent(3N)
services: service name data base. . . . services(4N)
inetd: Internet services daemon. . . . . inetd(1M)
services: service name data base. services(4N)
make typescript of terminal session. script: . . . . . script(1)
set42sig: set 4.2 BSD signal interface. . set42sig(3)
alarm: set a process's alarm clock. . alarm(2)
umask: set and get file creation mask. . umask(2)
context. sigstack: set and/or get signal stack . . sigstack(2)
ascii: map of ASCII character set. . . . . ascii(5)
sigsetmask: set current signal mask. . . . sigsetmask(2)
execution. env: set environment for command env(1)
times. utime: set file access and modification utime(2)
setgroups: set group access list. . . . . setgroups(2)
apply: apply a command to a set of arguments. . . . . apply(1)
semget: get set of semaphores. . . . . semget(2)
getsockopt, setsockopt: get and set options on sockets. . . . getsockopt(2N)
domain system. domainname: set or display name of current domainname(1)
mode. setcompat, getcompat: set or get process compatibility setcompat(2)
current host system. hostid: set or print identifier of . . . hostid(1N)
system. hostname: set or print name of current host hostname(1N)
a sensible state. tset, reset: set or reset the teletype bits to tset(1)
remove message queue, semaphore set, or shared memory ID. ipcrm: ipcrm(1)
information. badblk: set or update bad block . . . badblk(1M)
setpgrp: set process group ID. . . . . setpgrp(2)
setregid: set real and effective group ID. setregid(2)
setreuid: set real and effective user ID's. setreuid(2)
tabs: set tabs on a terminal. . . . . tabs(1)
and line discipline. getty: set terminal type, modes, speed, getty(1M)
date: print and set the date. . . . . date(1)
stty: set the options for a terminal. . stty(1)
chip. mactime: set the system time/real time . mactime(1M)
Macintosh resource file. settc: set the type and creator of a . settc(1)
stime: set time. . . . . stime(2)
A/UX Toolbox/ toolboxdaemon: set up for and clean up after . toolboxd(1M)
setuid, setgid: set user and group IDs. . . . setuid(2)
setuid, setgid: set user and group IDs. . . . setuid(3)
ulimit: get and set user limits. . . . . ulimit(2)
interface. set42sig: set 4.2 BSD signal . set42sig(3)
to a stream. setbuf, setvbuf: assign buffering setbuf(3S)
process compatibility mode. setcompat, getcompat: set or get setcompat(2)
current network/ getdomainname, setdomainname: get/set name of getdomainname(2N)
setuid, setgid: set user and group IDs. setuid(2)
setuid, setgid: set user and group IDs. setuid(3)
getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent:/ . getgrent(3C)
setgroups: set group access list. setgroups(2)

/gethostbyaddr, gethostbyname, sethostent, endhostent: get/ . . gethostent(3N)
identifier of current/ gethostid, sethostid: get/set unique . . . gethostid(2N)
current host. gethostname, sethostname: get/set name of . gethostname(2N)
interval timer. getitimer, setitimer: get/set value of . . getitimer(2)
setjmp, longjmp: non-local goto. setjmp(3C)
encryption. crypt, setkey, encrypt: generate DES crypt(3C)
endmntent, hasmntopt: get file/ setmntent, getmntent, addmntent, getmntent(3)
/getnetbyaddr, getnetbyname, setnetent, endnetent: get network/ getnetent(3N)
innetgr: get/ getnetgrent, setnetgrent, endnetgrent, . . . getnetgrent(3N)
setpgrp: set process group ID. setpgrp(2)
/getprotobynumber, getprotobyname, setprotoent, endprotoent: get/ . getprotoent(3N)
/addptabent, endptabent, setptabent, numbptabent: get/ . getptabent(3)
getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent:/ getpwent(3C)
group ID. setregid: set real and effective setregid(2)
user ID's. setreuid: set real and effective setreuid(2)
/getservbyport, getservbyname, setservent, endservent: get/ . . getservent(3N)
on sockets. getsockopt, setsockopt: get and set options getsockopt(2N)
of a Macintosh resource file. settc: set the type and creator . settc(1)
time. gettimeofday, settimeofday: get/set date and gettimeofday(2)
login time. profile: setting up an environment at . profile(4)
gettydefs: speed and terminal settings used by getty. . . . . gettydefs(4)
group IDs. setuid, setgid: set user and . . setuid(2)
group IDs. setuid, setgid: set user and . . setuid(3)
/getutid, getutline, pututline, setutent, endutent, utmpname:/ getut(3C)
stream. setbuf, setvbuf: assign buffering to a . setbuf(3S)
of/ paste: merge same lines of several files or subsequent lines paste(1)
in a machine independent/ sputl, sgetl: access long integer data sputl(3X)
standard/restricted command/ sh, rsh: Bourne shell, . . . . sh(1)
shmctl: shared memory control operations. shmctl(2)
message queue, semaphore set, or shared memory ID. ipcrm: remove ipcrm(1)
shmop, shmat, shmdt: shared memory operations. . . shmop(2)
shmget: get shared memory segment. . . shmget(2)
from C programs to implement shared strings. /extract strings xstr(1)
C-like syntax. csh: C shell, a command interpreter with csh(1)
language. ksh: Korn shell, a command programming ksh(1)
chsh: change default login shell. . . . . chsh(1)
system: issue a shell command from Fortran. . system(3F)
system: issue a shell command. . . . . system(3S)
shl: shell layer manager. . . . . shl(1)
/shutacct, startup, turnacct: shell procedures for accounting. acctsh(1M)
remsh: remote shell. . . . . remsh(1N)
powerfail: system initialization shell scripts. /bcheckrc, rc, . . brc(1M)
remshd: remote shell server. . . . . remshd(1M)
command/ sh, rsh: Bourne shell, standard/restricted . . . sh(1)
shl: shell layer manager. . . . shl(1)
operations. shmop, shmat, shmdt: shared memory shmop(2)
operations. shmctl: shared memory control shmctl(2)
shmop, shmat, shmdt: shared memory operations. shmop(2)
segment. shmget: get shared memory . shmget(2)
memory operations. shmop, shmat, shmdt: shared . shmop(2)
showmount: show all remote mounts. . . . showmount(1M)

groups: show group memberships. . . groups(1)
machines (RPC version). rup: show host status of local . . . rup(1C)
machines. ruptime: show host status of local . . . ruptime(1N)
uptime: show how long system has been up. uptime(1)
netstat: show network status. . . . . netstat(1N)
mounts. showmount: show all remote . showmount(1M)
connection. shutdown: shut down part of a full-duplex shutdown(2N)
shell/ /prctmp, prdaily, prtacct, shutacct, startup, turnacct: . . acctsh(1M)
full-duplex connection. shutdown: shut down part of a shutdown(2N)
processing. shutdown: terminate all . . . shutdown(1M)
sdiff: side-by-side difference program. sdiff(1)
sigblock: block signals. . . . sigblock(2)
transfer-of-sign intrinsic/ sign, isign, dsign: Fortran . . sign(3F)
login: sign on. . . . . login(1)
optional BSD-compatible software signal facilities. sigvec: . . . sigvec(2)
set42sig: set 4.2 BSD signal interface. . . . . set42sig(3)
sigsetmask: set current signal mask. . . . . sigsetmask(2)
pause: suspend process until signal. . . . . pause(2)
what to do upon receipt of a signal. signal: specify . . . . signal(3)
action on receipt of a system signal. signal: specify Fortran signal(3F)
receipt of a system signal. signal: specify Fortran action on signal(3F)
receipt of a signal. signal: specify what to do upon signal(3)
sigstack: set and/or get signal stack context. . . . . sigstack(2)
killpg: send signal to a process group. . . killpg(3N)
processes. kill: send a signal to a process or a group of kill(2)
/atomically release blocked signals and wait for interrupt. . sigpause(2)
sigblock: block signals. . . . . sigblock(2)
ssignal, gsignal: software signals. . . . . ssignal(3C)
blocked signals and wait for/ sigpause: atomically release . sigpause(2)
mask. sigsetmask: set current signal . sigsetmask(2)
stack context. sigstack: set and/or get signal . sigstack(2)
software signal facilities. sigvec: optional BSD-compatible sigvec(2)
lex: generate programs for simple lexical tasks. . . . . lex(1)
rand, srand: simple random-number generator. rand(3C)
fmt: simple text formatter. . . . . fmt(1)
atan2: trigonometric functions. sin, cos, tan, asin, acos, atan, . trig(3M)
intrinsic function. sin, dsin, csin: Fortran sine . . sin(3F)
sin, dsin, csin: Fortran sine intrinsic function. . . . sin(3F)
sinh, dsinh: Fortran hyperbolic sine intrinsic function. . . . sinh(3F)
ssp: make output single spaced. . . . . ssp(1)
functions. sinh, cosh, tanh: hyperbolic . sinh(3M)
sine intrinsic function. sinh, dsinh: Fortran hyperbolic sinh(3F)
get descriptor table size getdtablesize: . . . . . getdtablesize(2)
pagesize: print system page size. . . . . pagesize(1)
common object files. size: print section sizes of . . size(1)
size: print section sizes of common object files. . size(1)
interval. sleep: suspend execution for an sleep(1)
interval. sleep: suspend execution for . sleep(3C)
for typesetting viewgraphs and slides. /a troff macro package mv(5)
mvt: typeset view graphs and slides. . . . . mvt(1)
current user. ttyslot: find the slot in the utmp file of the . . ttyslot(3C)

slots: ROM library functions. . slots(3X)
spline: interpolate smooth curve. . . . . spline(1G)
int, ifix, idint, real, float, sngl, dble, cmplx, dcmplx, ichar,/ ftype(3F)
sno: SNOBOL interpreter. . . sno(1)
sno: SNOBOL interpreter. . . . . sno(1)
accept: accept a connection on a socket. . . . . accept(2N)
bind: bind a name to a socket. . . . . bind(2N)
initiate a connection on a socket. connect: . . . . . connect(2N)
communication. socket: create an endpoint for . socket(2N)
listen for connections on a socket. listen: . . . . . listen(2N)
getsockname: get socket name. . . . . getsockname(2N)
recvmsg: receive a message from a socket. recv, recvfrom, . . . recv(2N)
sendmsg: send a message from a socket. send, sendto, . . . . send(2N)
get and set options on sockets. getsockopt, setsockopt: getsockopt(2N)
nroff input. soelim: eliminate .so's from . soelim(1)
pg: file perusal filter for soft-copy terminals. . . . . pg(1)
interface. lo: software loopback network . . lo(5)
over a network running B-NET software. /write to all users . rwall(1M)
sigvec: optional BSD-compatible software signal facilities. . . sigvec(2)
ssignal, gsignal: software signals. . . . . ssignal(3C)
sort: sort and/or merge files. . . . sort(1)
sortbib: sort bibliographic database. . sortbib(1)
qsort: quicker sort. . . . . qsort(3C)
sort: sort and/or merge files. . sort(1)
tsort: topological sort. . . . . tsort(1)
database. sortbib: sort bibliographic . . sortbib(1)
or reject lines common to two sorted files. comm: select . . comm(1)
bsearch: binary search a sorted table. . . . . bsearch(3C)
soelim: eliminate .so's from nroff input. . . . . soelim(1)
program. whereis: locate source, binary, and/or manual for whereis(1)
indent and format C program source. indent: . . . . . indent(1)
error message file by massaging C source. mkstr: create an . . . mkstr(1)
brk, sbrk: change data segment space allocation. . . . . brk(2)
ssp: make output single spaced. . . . . ssp(1)
expand, unexpand: expand tabs to spaces, and vice versa. . . . expand(1)
ct: spawn getty to a remote terminal. ct(1C)
eqn and neqn. eqnchar: special character definitions for eqnchar(5)
dev_kill: remove special devices from directories. dev_kill(1M)
mknod: build special file. . . . . mknod(1M)
300s/ 300, 300s: handle special functions of DASI 300 and 300(1)
terminal. 450: handle special functions of the DASI 450 450(1)
mknod: make a directory, or a special or ordinary file. . . . mknod(2)
fspec: format specification in text files. . . fspec(4)
of extended errors in the specified device. /the reporting exterr(1M)
ftruncate: truncate a file to a specified length. truncate, . . truncate(2)
of a system signal. signal: specify Fortran action on receipt signal(3F)
of a signal. signal: specify what to do upon receipt signal(3)
getty: set terminal type, modes, speed, and line discipline. . . getty(1M)
by getty. gettydefs: speed and terminal settings used gettydefs(4)
hashcheck: find spelling errors. spell, hashmake, spellin, . . . spell(1)
errors. spell, hashmake, spellin, hashcheck: find spelling spell(1)

spellin, hashcheck: find spelling errors. /hashmake, . spell(1)
spline: interpolate smooth curve. spline(1G)
split: split a file into pieces. . . . . split(1)
csplit: context split. . . . . csplit(1)
fsplit: split f77 or efl files. . . . . fsplit(1)
split: split a file into pieces. . split(1)
uuclean: uucp spool directory clean-up. . . uuclean(1M)
lpq: spool queue examination program. lpq(1)
/psbanner, psinterface: TranScript spooler filters for PostScript/ . transcript(1M)
remove jobs from the line printer spooling queue. lprm: . . . lprm(1)
lpadmin: configure the LP spooling system. . . . . lpadmin(1M)
spray: spray packets. . . . . spray(1M)
sprayd: spray server. . . . . sprayd(1M)
spray: spray packets. . . . . spray(1M)
sprayd: spray server. . . . . sprayd(1M)
printf, fprintf, sprintf: print formatted output. printf(3S)
data in a machine independent/ sputl, sgetl: access long integer sputl(3X)
square root intrinsic function. sqrt, dsqrt, csqrt: Fortran . . sqrt(3F)
power,/ exp, log, log10, pow, sqrt: exponential, logarithm, . exp(3M)
exponential, logarithm, power, square root functions. /sqrt: . exp(3M)
sqrt, dsqrt, csqrt: Fortran square root intrinsic function. . sqrt(3F)
random-number generator. irand, srand, rand: Fortran uniform . rand(3F)
generator. rand, srand: simple random-number rand(3C)
/nrand48, mrand48, jrand48, srand48, seed48, lcong48:/ . . drand48(3C)
rcvhex: translates Motorola S-records from downloading into a/ rcvhex(1)
scanf, fscanf, sscanf: convert formatted input. scanf(3S)
signals. ssignal, gsignal: software . . ssignal(3C)
ssp: make output single spaced. ssp(1)
sigstack: set and/or get signal stack context. . . . . sigstack(2)
launch an A/UX kernel from the standalone environment. launch: launch(8)
a command interpreter for the standalone environment. sash: sash(8)
autorecovery: standalone file system repair. . autorecovery(8)
communication package. ftok: standard interprocess . . . . ftok(3C)
sh, rsh: Bourne shell, standard/restricted command/ . sh(1)
twinkle: twinkle stars on the screen. . . . . twinkle(6)
lpsched, lpshut, lpmove: start/stop the LP request/ . . lpsched(1M)
boot: startup procedures. . . . . boot(8)
prdaily, prtacct, shutacct, startup, turnacct: shell/ /prctmp, acctsh(1M)
system call. stat: data returned by stat . . stat(5)
status. stat, fstat, lstat: get file . . . stat(2)
stat: data returned by stat system call. . . . . stat(5)
the teletype bits to a sensible state. tset, reset: set or reset . tset(1)
statistics. statfs: get file system . . . . statfs(2)
systems. fstab: static information about file . fstab(4)
ff: list file names and statistics for a file system. . . ff(1M)
lav: print load average statistics. . . . . lav(1)
display kernel name cache statistics. ncstats: . . . . . ncstats(1M)
nfsstat: Network File System statistics. . . . . nfsstat(1M)
rstatd: kernel statistics server. . . . . rstatd(1M)
statfs: get file system statistics. . . . . statfs(2)
ustat: get file system statistics. . . . . ustat(2)

lpstat: print LP status information. . . . . lpstat(1)
feof, clearerr, fileno: stream status inquiries. ferror, . . . ferror(3S)
uustat: uucp status inquiry and job control. uustat(1C)
communication facilities status. /report interprocess . . ipcs(1)
sysline: display system status on status line of a terminal. . . . sysline(1)
netstat: show network status. . . . . netstat(1N)
version). rup: show host status of local machines (RPC rup(1C)
ruptime: show host status of local machines. . . . ruptime(1N)
sysline: display system status on status line of a/ . . . sysline(1)
ps: report process status. . . . . ps(1)
rwhod: system status server. . . . . rwhod(1M)
stat, fstat, lstat: get file status. . . . . stat(2)
stime: set time. . . . . stime(2)
wait: wait for child process to stop or terminate. . . . . wait(2)
wait3: wait for child process to stop or terminate. . . . . wait3(2N)
data base/ dbminit, fetch, store, delete, firstkey, nextkey: dbm(3X)
strcpy, strncpy, strlen, strchr,/ strcat, strncat, strcmp, strncmp, string(3C)
/strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn,/ string(3C)
strlen, strchr,/ strcat, strncat, strcmp, strncmp, strcpy, strncpy, string(3C)
strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr,/ string(3C)
strchr, strrchr, strpbrk, strspn, strcspn, strtok: string/ /strlen, . string(3C)
sed: stream editor. . . . . sed(1)
fclose, fflush: close or flush a stream. . . . . fclose(3S)
fopen, freopen, fdopen: open a stream. . . . . fopen(3S)
reposition a file pointer in a stream. fseek, rewind, ftell: . fseek(3S)
get character or word from a stream. /getchar, fgetc, getw: . getc(3S)
gets, fgets: get a string from a stream. . . . . gets(3S)
putw: put character or word on a stream. putc, putchar, fputc, . putc(3S)
puts, fputs: put a string on a stream. . . . . puts(3S)
setvbuf: assign buffering to a stream. setbuf, . . . . . setbuf(3S)
ferror, feof, clearerr, fileno: stream status inquiries. . . . ferror(3S)
/ruserok: routines for returning a stream to a remote command. . rcmd(3N)
rexec: return stream to a remote command. . rexec(3N)
push character back into input stream. ungetc: . . . . . ungetc(3S)
streams: ioctl interface. . . . streams(7)
line_push: routine used to push streams line disciplines. . . . line_push(3)
line_sane: push streams line disciplines. . . . line_sane(1M)
long integer and base-64 ASCII string. /l64a: convert between a64l(3C)
functions. lge, lgt, lle, llt: string comparision intrinsic . lge(3F)
convert floating-point number to string. ecvt, fcvt, gcvt: . . . ecvt(3C)
gets, fgets: get a string from a stream. . . . . gets(3S)
len: return length of Fortran string. . . . . len(3F)
puts, fputs: put a string on a stream. . . . . puts(3S)
bcmp, bzero, ffs: bit and byte string operations. bcopy, . . bstring(3)
strpbrk, strspn, strcspn, strtok: string operations. /strrchr, . . string(3C)
number. strtod: convert string to double-precision . . strtod(3C)
atof: convert ASCII string to floating-point number. atof(3C)
strtol, atol, atoi: convert string to integer. . . . . strtol(3C)
strings in an object, or other/ strings: find the printable . . strings(1)
implement shared/ xstr: extract strings from C programs to . . xstr(1)
strings: find the printable strings in an object, or other/ . strings(1)

C programs to implement shared strings. /extract strings from . xstr(1)
number information from an/ strip: strip symbol and line . . strip(1)
information from an/ strip: strip symbol and line number . strip(1)
/strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk,/ string(3C)
strncpy, strlen, strchr,/ strcat, strncat, strcmp, strncmp, strcpy, string(3C)
strchr,/ strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, string(3C)
/strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr,/ string(3C)
/strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok:/ string(3C)
/strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn,/ . . . string(3C)
/strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok: string/ . string(3C)
double-precision number. strtod: convert string to . . . strtod(3C)
/strpbrk, strspn, strcspn, strtok: string operations. . . . string(3C)
string to integer. strtol, atol, atoi: convert . . . strtol(3C)
processes using a file or file structure. fuser: identify . . . fuser(1M)
database and directory structure. /the yellowpages . ypfiles(4)
terminal. stty: set the options for a . . . stty(1)
characteristics of a document. style: analyze surface . . . . style(1)
su: substitute user ID. . . . . su(1)
from a document. subj: generate a list of subjects subj(1)
document. ndx: create a subject-page index for a . . . ndx(1)
subj: generate a list of subjects from a document. . . subj(1)
intro: introduction to subroutines and libraries. . . intro(3)
firstkey, nextkey: data base subroutines. /store, delete, . . dbm(3X)
plot: graphics interface subroutines. . . . . plot(3X)
/same lines of several files or subsequent lines of one file. . paste(1)
su: substitute user ID. . . . . su(1)
index: return location of Fortran substring. . . . . index(3F)
sccs: front end for the SCCS subsystem. . . . . sccs(1)
files in the given/ sumdir: sum and count characters in the sumdir(1)
count of a file. sum: print checksum and block sum(1)
in the files in the given/ sumdir: sum and count characters sumdir(1)
du: summarize disk usage. . . . du(1)
accounting/ acctcms: command summary from per-process . . acctcms(1M)
sync: update the superblock. . . . . sync(1)
sync: update superblock. . . . . sync(2)
document. style: analyze surface characteristics of a . . style(1)
interval. sleep: suspend execution for an . . sleep(1)
sleep: suspend execution for interval. sleep(3C)
pause: suspend process until signal. . pause(2)
swab: swap bytes. . . . . swab(3C)
swap: swap administrative interface. swap(1M)
swab: swap bytes. . . . . swab(3C)
interface. swap: swap administrative . . swap(1M)
sxt: pseudo-device driver. . . sxt(7)
information from an/ strip: strip symbol and line number . . . strip(1)
ldgetname: retrieve symbol name for object file. . ldgetname(3X)
ldtbindex: compute the index of a symbol table entry of a common/ ldtbindex(3X)
object/ ldtbread: read an indexed symbol table entry of a common ldtbread(3X)
syms: common object file symbol table format. . . . . syms(4)
file. ldtbseek: seek to the symbol table of a common object ldtbseek(3X)
sdb: symbolic debugger. . . . . sdb(1)

readlink: read value of a symbolic link. . . . . readlink(2)
symlink: make symbolic link to a file. . . . symlink(2)
file. symlink: make symbolic link to a symlink(2)
table format. syms: common object file symbol syms(4)
sync: update superblock. . . sync(2)
sync: update the superblock. . sync(1)
clock. /correct the time to allow synchronization of the system adjtime(2)
state with that on disk. fsync: synchronize a file's in-core . . fsync(2)
select: synchronous I/O multiplexing. select(2N)
a command interpreter with C-like syntax. csh: C shell, . . . . csh(1)
error messages. perror, errno, sys_errlist, sys_nerr: system . perror(3C)
status line of a terminal. sysline: display system status on sysline(1)
perror, errno, sys_errlist, sys_nerr: system error messages. perror(3C)
sag: system activity graph. . . . . sag(1G)
sadc, sa1, sa2: system activity report package. sadc(1M)
sar: system activity reporter. . . . sar(1)
command; report process data and system activity. timex: time a timex(1)
stat: data returned by stat system call. . . . . stat(5)
intro: introduction to system calls and error numbers. intro(2)
to allow synchronization of the system clock. /correct the time adjtime(2)
uux: UNIX-to-UNIX system command execution. . uux(1C)
interactive repair. fsck: file system consistency check and fsck(1M)
uuname: UNIX system to UNIX system copy. uucp, uulog, . . uucp(1C)
crash: what to do when the system crashes. . . . . crash(8)
cu: call another system. . . . . cu(1C)
types: primitive system data types. . . . . types(5)
fsdb: file system debugger. . . . . fsdb(1M)
/endmntent, hasmntopt: get file system descriptor file entry. . getmntent(3)
mfs: read a Macintosh flat file system disk. . . . . mfs(1)
or display name of current domain system. domainname: set . . domainname(1)
rdumpfs: file system dump across the network. rdumpfs(1M)
rrestore: restore a file system dump across the network. rrestore(1M)
dumpfs: incremental file system dump. . . . . dumpfs(1M)
errno, sys_errlist, sys_nerr: system error messages. perror, perror(3C)
pstat: print system facts. . . . . pstat(1M)
names and statistics for a file system. ff: list file . . . . . ff(1M)
uuto, uupick: public UNIX-to-UNIX system file copy. . . . . uuto(1C)
fsmount: mount an NFS file system. . . . . fsmount(2)
uptime: show how long system has been up. . . . . uptime(1)
print identifier of current host system. hostid: set or . . . . hostid(1N)
set or print name of current host system. hostname: . . . . . hostname(1N)
/gets directory entries in a file system independent format . . getdirentries(2)
brc, bcheckrc, rc, powerfail: system initialization shell/ . . brc(1M)
from Fortran. system: issue a shell command system(3F)
system: issue a shell command. system(3S)
configure the LP spooling system. lpadmin: . . . . . lpadmin(1M)
interactive message processing system. mailx: . . . . . mailx(1)
mkfs1b: construct a file system. . . . . mkfs1b(1M)
mkfs: construct a file system. . . . . mkfs(1M)
mount: mount a file system. . . . . mount(3)
chgnod: change current A/UX system nodename. . . . . chgnod(1M)

pagesize: print system page size. . . . . pagesize(1)
powerdown: power down the system. . . . . powerdown(1M)
reboot: reboot the system. . . . . reboot(1M)
reboot: reboot the system. . . . . reboot(2)
autorecovery: standalone file system repair. . . . . autorecovery(8)
restore: incremental file system restore. . . . . restore(1M)
Fortran action on receipt of a system signal. signal: specify . signal(3F)
nfsstat: Network File System statistics. . . . . nfsstat(1M)
statfs: get file system statistics. . . . . statfs(2)
ustat: get file system statistics. . . . . ustat(2)
terminal. sysline: display system status on status line of a sysline(1)
rwhod: system status server. . . . . rwhod(1M)
mtab: mounted file system table. . . . . mtab(4)
rmtab: remotely mounted file system table. . . . . rmtab(4)
mactime: set the system time/real time chip. . . mactime(1M)
tip: connect to a remote system. . . . . tip(1C)
uucp, uulog, uuname: UNIX system to UNIX system copy. uucp(1C)
umount: unmount a file system. . . . . umount(2)
umount: unmount a file system. . . . . umount(3)
uname: print name of current system. . . . . uname(1)
uname: get name of current system. . . . . uname(2)
unmount: remove a file system. . . . . unmount(2)
list of users who are on the system. users: compact . . . users(1)
dir: format of System V directories. . . . . dir(4)
inode: format of a System V inode. . . . . inode(4)
fs: format of a System V system volume. . . fs(4)
fs: format of a System V system volume. . . . . fs(4)
who: who is on the system. . . . . who(1)
exports: NFS file systems being exported. . . . exports(4)
dcopy: copy file systems for optimal access time. dcopy(1M)
static information about file systems. fstab: . . . . . fstab(4)
umount: mount and dismount file systems. mount, . . . . . mount(1M)
volcopy, labelit: copy file systems with label checking. . volcopy(1M)
information. uvar: returns system-specific configuration . uvar(2)
bsearch: binary search a sorted table. . . . . bsearch(3C)
/compute the index of a symbol table entry of a common object/ ldtbindex(3X)
ldtbread: read an indexed symbol table entry of a common object/ ldtbread(3X)
numbptabent: get partition table file entry. /setptabent, . getptabent(3)
ptab: partition table file. . . . . ptab(4)
syms: common object file symbol table format. . . . . syms(4)
mtab: mounted file system table. . . . . mtab(4)
ldtbseek: seek to the symbol table of a common object file. ldtbseek(3X)
remotely mounted file system table. rmtab: . . . . . rmtab(4)
getdtablesize: get descriptor table size . . . . . getdtablesize(2)
nterm: terminal driving tables for nroff. . . . . nterm(5)
tbl: format tables for nroff or troff. . . . tbl(1)
hdestroy: manage hash search tables. hsearch, hcreate, . . . hsearch(3C)
manually manipulate the routing tables. route: . . . . . route(1M)
tabs: set tabs on a terminal. . . . . tabs(1)
tabs: set tabs on a terminal. . . tabs(1)
expand, unexpand: expand tabs to spaces, and vice versa. expand(1)

ctags: maintain a tags file for a C program. . . ctags(1)
file. tail: deliver the last part of a . tail(1)
machine. take: takes a file from a remote take(1C)
machine. take: takes a file from a remote . . take(1C)
talk: talk to another user. . . talk(1N)
talk: talk to another user. . . . . talk(1N)
server. talkd: remote user communication talkd(1M)
trigonometric/ sin, cos, tan, asin, acos, atan, atan2: . . trig(3M)
intrinsic function. tan, dtan: Fortran tangent . . tan(3F)
tan, dtan: Fortran tangent intrinsic function. . . tan(3F)
tanh, dtanh: Fortran hyperbolic tangent intrinsic function. . . tanh(3F)
tangent intrinsic function. tanh, dtanh: Fortran hyperbolic tanh(3F)
sinh, cosh, tanh: hyperbolic functions. . . sinh(3M)
tp: manipulate tape archive. . . . . tp(1)
tar: tape file archiver. . . . . tar(1)
frec: recover files from a backup tape. . . . . frec(1M)
tar: tape file archiver. . . . . tar(1)
programs for simple lexical tasks. lex: generate . . . . . lex(1)
deroff: remove nroff/troff, tbl, and eqn constructs. . . . deroff(1)
troff. tbl: format tables for nroff or . tbl(1)
tc: troff output interpreter. . . tc(1)
Control Protocol. tcp: Internet Transmission . . tcp(5P)
search trees. tsearch, tfind, tdelete, twalk: manage binary . tsearch(3C)
tee: pipe fitting. . . . . tee(1)
4014: paginator for the Tektronix 4014 terminal. . . 4014(1)
tset, reset: set or reset the teletype bits to a sensible/ . . tset(1)
indicate last logins of users and teletypes. last: . . . . . last(1)
initialization. init, telinit: process control . . . init(1M)
closedir:/ opendir, readdir, telldir, seekdir, rewinddir, . . directory(3)
telnetd: DARPA TELNET protocol server. . . telnetd(1M)
telnet: user interface to the TELNET protocol. . . . . telnet(1C)
TELNET protocol. telnet: user interface to the . . telnet(1C)
server. telnetd: DARPA TELNET protocol telnetd(1M)
temporary file. tmpnam, tempnam: create a name for a . tmpnam(3S)
tmpfile: create a temporary file. . . . . tmpfile(3S)
tempnam: create a name for a temporary file. tmpnam, . . . tmpnam(3S)
terminals. term: conventional names for . term(5)
term: format of compiled term file.. . . . . term(4)
file.. term: format of compiled term term(4)
base. termcap: terminal capability data termcap(4)
paginator for the Tektronix 4014 terminal. 4014: . . . . . 4014(1)
special functions of the DASI 450 terminal. 450: handle . . . . 450(1)
termcap: terminal capability data base. . termcap(4)
terminfo: terminal capability data base. . terminfo(4)
ct: spawn getty to a remote terminal. . . . . ct(1C)
ctermid: generate filename for terminal. . . . . ctermid(3S)
pty: pseudo terminal driver. . . . . pty(7)
nroff. nterm: terminal driving tables for . . nterm(5)
greek: select terminal filter. . . . . greek(1)
/tgetflag, tgetstr, tgoto, tputs: terminal independent operation/ termcap(3X)
termio: general terminal interface. . . . . termio(7)

tty: controlling terminal interface. . . . . tty(7)
dial: establish an out-going terminal line connection. . . dial(3C)
colcrt: filter nroff output for terminal previewing. . . . . colcrt(1)
clear: clear terminal screen. . . . . clear(1)
script: make typescript of terminal session. . . . . script(1)
gettydefs: speed and terminal settings used by getty. gettydefs(4)
stty: set the options for a terminal. . . . . stty(1)
system status on status line of a terminal. sysline: display . . sysline(1)
tabs: set tabs on a terminal. . . . . tabs(1)
ttyname, isatty: find name of a terminal. . . . . ttyname(3C)
line discipline. getty: set terminal type, modes, speed, and getty(1M)
ttytype: data base of terminal types by port. . . . ttytype(4)
worms: animate worms on a display terminal. . . . . worms(6)
functions of DASI 300 and 300s terminals. /300s: handle special 300(1)
tty: get the terminal's name. . . . . tty(1)
file perusal filter for soft-copy terminals. pg: . . . . . pg(1)
scr_color: program to change the terminal's screen color. . . . scr_color(1)
term: conventional names for terminals. . . . . term(5)
kill: terminate a process. . . . . kill(1)
shutdown: terminate all processing. . . . shutdown(1M)
abort: terminate Fortran program. . . abort(3F)
exit, _exit: terminate process. . . . . exit(2)
daemon. errstop: terminate the error-logging . . errstop(1M)
wait for child process to stop or terminate. wait: . . . . . wait(2)
wait for child process to stop or terminate. wait3: . . . . . wait3(2N)
tic: terminfo compiler. . . . . tic(1M)
tput: query terminfo database. . . . . tput(1)
data base. terminfo: terminal capability . terminfo(4)
interface. termio: general terminal . . . termio(7)
command. test: condition evaluation . . test(1)
quiz: test your knowledge. . . . . quiz(6)
ed, red: text editor. . . . . ed(1)
ex, edit: text editor. . . . . ex(1)
newform: change the format of a text file. . . . . newform(1)
fspec: format specification in text files. . . . . fspec(4)
neqn: format mathematical text for nroff. . . . . neqn(1)
checkcw: prepare constant-width text for otroff. cw, . . . . . cw(1)
eqn: format mathematical text for troff. . . . . eqn(1)
fmt: simple text formatter. . . . . fmt(1)
otroff: text formatting and typesetting. otroff(1)
troff: text formatting and typesetting. troff(1)
nroff: text formatting language. . . nroff(1)
ms: text formatting macros. . . . ms(5)
plock: lock process, text, or data in memory. . . . plock(2)
binary search trees. tsearch, tfind, tdelete, twalk: manage . tsearch(3C)
program. tftp: trivial file transfer . . . tftp(1C)
Transfer Protocol server. tftpd: DARPA Trivial File . . tftpd(1M)
tgetstr, tgoto, tputs: terminal/ tgetent, tgetnum, tgetflag, . . termcap(3X)
terminal/ tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs: . termcap(3X)
tgoto, tputs: terminal/ tgetent, tgetnum, tgetflag, tgetstr, . . termcap(3X)
tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs: terminal/ . termcap(3X)

tgetnum, tgetflag, tgetstr, tgoto, tputs: terminal/ tgetent, termcap(3X)
and uncompress files, and cat them. /uncompact, ccat: compress compact(1)
explain: print wordy sentences; thesaurus for diction. diction, diction(1)
tic: terminfo compiler. . . . tic(1M)
ttt, cubic: tic-tac-toe. . . . . ttt(6)
execute commands at a later time. at, batch: . . . . . at(1)
file systems for optimal access time. dcopy: copy . . . . . dcopy(1M)
time: get time. . . . . time(2)
settimeofday: get/set date and time. gettimeofday, . . . . gettimeofday(2)
up an environment at login time. profile: setting . . . . profile(4)
stime: set time. . . . . stime(2)
time: time a command. . . . time(1)
time: get time. . . . . time(2)
get/set value of interval timer. getitimer, setitimer: . . getitimer(2)
mactime: set the system time/real time chip. . . . . mactime(1M)
process times. times: get process and child . times(2)
update access and modification times of a file. touch: . . . . touch(1)
get process and child process times. times: . . . . . times(2)
set file access and modification times. utime: . . . . . utime(2)
process data and system/ timex: time a command; report timex(1)
tip: connect to a remote system. tip(1C)
tmpfile: create a temporary file. tmpfile(3S)
for a temporary file. tmpnam, tempnam: create a name tmpnam(3S)
/tolower, _toupper, _tolower, toascii: translate characters. . conv(3C)
popen, pclose: initiate pipe to/from a process. . . . . popen(3S)
toupper, tolower, _toupper, _tolower, toascii: translate/ . . conv(3C)
toascii: translate/ toupper, tolower, _toupper, _tolower, . conv(3C)
up for and clean up after A/UX Toolbox programs. /set . . . toolboxd(1M)
clean up after A/UX Toolbox/ toolboxdaemon: set up for and toolboxd(1M)
tsort: topological sort. . . . . tsort(1)
acctmerg: merge or add total accounting files. . . . . acctmerg(1M)
modification times of a file. touch: update access and . . touch(1)
translate/ toupper, tolower, _toupper, _tolower, toascii: . conv(3C)
_tolower, toascii: translate/ toupper, tolower, _toupper, . conv(3C)
tp: manipulate tape archive. . tp(1)
tplot: graphics filters. . . . . tplot(1G)
tput: query terminfo database. tput(1)
/tgetflag, tgetstr, tgoto, tputs: terminal independent/ . termcap(3X)
tr: translate characters. . . . tr(1)
ptrace: process trace. . . . . ptrace(2)
trpt: transliterate protocol trace. . . . . trpt(1M)
/pstext, psbanner, psinterface: TranScript spooler filters for/ . transcript(1M)
server to here. ypxfr: transfer a YP map from some YP ypxfr(1M)
blt, blt512: block transfer data. . . . . blt(3C)
kermit: kermit file transfer. . . . . kermit(1C)
ftp: ARPANET file transfer program. . . . . ftp(1N)
tftp: trivial file transfer program. . . . . tftp(1C)
ftpd: DARPA Internet File Transfer Protocol server. . . ftpd(1M)
tftpd: DARPA Trivial File Transfer Protocol server. . . tftpd(1M)
sign, isign, dsign: Fortran transfer-of-sign intrinsic/ . . sign(3F)
_toupper, _tolower, toascii: translate characters. /tolower, conv(3C)

tr: translate characters. . . . . tr(1)
from downloading into a/ rcvhex: translates Motorola S-records . rcvhex(1)
hex: translates object files. . . . . hex(1)
trpt: transliterate protocol trace. . . trpt(1M)
tcp: Internet Transmission Control Protocol. tcp(5P)
encode/decode a binary file for transmission via mail. /uudecode: uuencode(1C)
ftw: walk a file tree. . . . . ftw(3C)
twalk: manage binary search trees. tsearch, tfind, tdelete, . tsearch(3C)
trek: trekkie game. . . . . trek(6)
trek: trekkie game. . . . . trek(6)
tan, asin, acos, atan, atan2: trigonometric functions. /cos, trig(3M)
tftp: trivial file transfer program. . tftp(1C)
server. tftpd: DARPA Trivial File Transfer Protocol . tftpd(1M)
iwprep: prepare troff description files. . . . . iwprep(1)
language. troff: description of output . . troff(5)
eqn: format mathematical text for troff. . . . . eqn(1)
files for device-independent troff. font: description . . . font(5)
psdit: convert troff intermediate format to/ . psdit(1)
typesetting viewgraphs and/ mv: a troff macro package for . . . mv(5)
tc: troff output interpreter. . . . tc(1)
daiw: Apple ImageWriter II troff postprocessor filter. . . daiw(1)
pictures. pic: troff preprocessor for drawing pic(1)
tbl: format tables for nroff or troff. . . . . tbl(1)
typesetting. troff: text formatting and . . troff(1)
psroff: troff to a/ . . . . . psroff(1)
trace. trpt: transliterate protocol . . trpt(1M)
values. true, false: provide truth . . . true(1)
length. truncate, ftruncate: truncate a file to a specified . truncate(2)
file to a specified length. truncate, ftruncate: truncate a . truncate(2)
hosts.equiv: list of trusted hosts. . . . . hosts.equiv(4)
/u3b2, u3b5, u3b15, vax: provide truth value about processor type. machid(1)
true, false: provide truth values. . . . . true(1)
chase: try to escape the killer robots. . chase(6)
manage binary search trees. tsearch, tfind, tdelete, twalk: . tsearch(3C)
teletype bits to a sensible/ tset, reset: set or reset the . . tset(1)
tsort: topological sort. . . . . tsort(1)
ttt, cubic: tic-tac-toe. . . . . ttt(6)
interface. tty: controlling terminal . . . tty(7)
tty: get the terminal's name. . tty(1)
greek: graphics for the extended TTY-37 type-box. . . . . greek(5)
/etc/inittab file. tty_add, tty_kill: modify the . tty_add(1M)
file. tty_add, tty_kill: modify the /etc/inittab tty_add(1M)
terminal. ttyname, isatty: find name of a ttyname(3C)
utmp file of the current user. ttyslot: find the slot in the . . ttyslot(3C)
types by port. ttytype: data base of terminal . ttytype(4)
change a kernel's parameters for tuning. kconfig: . . . . . kconfig(1M)
extended errors in the/ exterr: turn on/off the reporting of . . exterr(1M)
/prtacct, shutacct, startup, turnacct: shell procedures for/ acctsh(1M)
trees. tsearch, tfind, tdelete, twalk: manage binary search . tsearch(3C)
twinkle: twinkle stars on the screen. . . twinkle(6)
screen. twinkle: twinkle stars on the . twinkle(6)

resource file. settc: set the type and creator of a Macintosh settc(1)
ichar, char: explicit Fortran type conversion. /cmplx, dcmplx, ftype(3F)
file: determine file type. . . . . file(1)
truth value about processor type. /u3b5, u3b15, vax: provide machid(1)
discipline. getty: set terminal type, modes, speed, and line . getty(1M)
graphics for the extended TTY-37 type-box. greek: . . . . . greek(5)
ttytype: data base of terminal types by port. . . . . ttytype(4)
types. types: primitive system data . types(5)
types: primitive system data types. . . . . types(5)
script: make typescript of terminal session. script(1)
mmt: typeset documents. . . . . mmt(1)
mvt: typeset view graphs and slides. mvt(1)
otroff: text formatting and typesetting. . . . . otroff(1)
troff: text formatting and typesetting. . . . . troff(1)
mv: a troff macro package for typesetting viewgraphs and/ . mv(5)
tzdump: time zone dumper. . tzdump(1M)
tzfile: time zone information. . tzfile(4)
tzic: time zone compiler. . . tzic(1M)
and/ /localtime, gmtime, asctime, tzset, tzsetwall: convert date . ctime(3)
to/ /gmtime, asctime, tzset, tzsetwall: convert date and time ctime(3)
provide truth value/ m68k, pdp11, u3b, u3b2, u3b5, u3b15, vax: . machid(1)
m68k, pdp11, u3b, u3b2, u3b5, u3b15, vax: provide truth value/ machid(1)
truth value/ m68k, pdp11, u3b, u3b2, u3b5, u3b15, vax: provide machid(1)
value/ m68k, pdp11, u3b, u3b2, u3b5, u3b15, vax: provide truth machid(1)
Protocol. udp: Internet User Datagram . udp(5P)
getpw: get name from UID. . . . . getpw(3C)
ul: do underlining. . . . . ul(1)
ulimit: get and set user limits. . ulimit(2)
mask. umask: set and get file creation umask(2)
systems. mount, umount: mount and dismount file mount(1M)
umount: unmount a file system. umount(2)
umount: unmount a file system. umount(3)
system. uname: get name of current . uname(2)
system. uname: print name of current . uname(1)
uncompress files, and/ compact, uncompact, ccat: compress and compact(1)
/uncompact, ccat: compress and uncompress files, and cat them. compact(1)
ul: do underlining. . . . . ul(1)
file. unget: undo a previous get of an SCCS unget(1)
and vice versa. expand, unexpand: expand tabs to spaces, expand(1)
SCCS file. unget: undo a previous get of an unget(1)
input stream. ungetc: push character back into ungetc(3S)
irand, srand, rand: Fortran uniform random-number generator. rand(3F)
seed48, lcong48: generate uniformly distributed/ /srand48, drand48(3C)
file. uniq: report repeated lines in a uniq(1)
mktemp: make a unique filename. . . . . mktemp(3C)
gethostid, sethostid: get/set unique identifier of current/ . gethostid(2N)
units: conversion program. . . units(1)
uulog, uuname: UNIX system to UNIX system copy. uucp, . . uucp(1C)
uucp, uulog, uuname: UNIX system to UNIX system copy. uucp(1C)
execution. uux: UNIX-to-UNIX system command uux(1C)
uuto, uupick: public UNIX-to-UNIX system file copy. uuto(1C)

unlink: remove directory entry. unlink(2)
umount: unmount a file system. . . . umount(2)
umount: unmount a file system. . . . umount(3)
unmount: remove a file system. unmount(2)
files. pack, pcat, unpack: compress and expand pack(1)
pause: suspend process until signal. . . . . pause(2)
show how long system has been up. uptime: . . . . . uptime(1)
times of a file. touch: update access and modification touch(1)
programs. make: maintain, update, and regenerate groups of make(1)
badblk: set or update bad block information. badblk(1M)
machines. updater: update files between two . . updater(1)
lsearch, lfind: linear search and update. . . . . lsearch(3C)
sync: update superblock. . . . . sync(2)
sync: update the superblock. . . . sync(1)
machines. updater: update files between two updater(1)
signal: specify what to do upon receipt of a signal. . . . signal(3)
been up. uptime: show how long system has uptime(1)
autoconfig: build a new up-to-date kernel. . . . . autoconfig(1M)
du: summarize disk usage. . . . . du(1)
clock: report CPU time used. . . . . clock(3C)
id: print user and group IDs and names. id(1)
setuid, setgid: set user and group IDs. . . . . setuid(2)
setuid, setgid: set user and group IDs. . . . . setuid(3)
talkd: remote user communication server. . talkd(1M)
crontab: user crontab utility. . . . . crontab(1)
get character login name of the user. cuserid: . . . . . cuserid(3S)
udp: Internet User Datagram Protocol. . . udp(5P)
and/ /getgid, getegid: get real user, effective user, real group, getuid(2)
environ: user environment. . . . . environ(5)
generate disk accounting data by user ID. diskusg: . . . . . diskusg(1M)
su: substitute user ID. . . . . su(1)
whoami: print effective current user ID. . . . . whoami(1)
setreuid: set real and effective user ID's. . . . . setreuid(2)
finger: user information lookup program. finger(1)
protocol. telnet: user interface to the TELNET telnet(1C)
ulimit: get and set user limits. . . . . ulimit(2)
logname: return login name of user. . . . . logname(3X)
/getegid: get real user, effective user, real group, and effective/ getuid(2)
talk: talk to another user. . . . . talk(1N)
in the utmp file of the current user. ttyslot: find the slot . . ttyslot(3C)
write: write to another user. . . . . write(1)
last: indicate last logins of users and teletypes. . . . . last(1)
are on the system. users: compact list of users who users(1)
mail, rmail: send mail to users or read mail. . . . . mail(1)
B-NET/ rwall: write to all users over a network running . rwall(1M)
wall: write to all users. . . . . wall(1M)
users: compact list of users who are on the system. . users(1)
fuser: identify processes using a file or file structure. . fuser(1M)
help: ask for help in using SCCS. . . . . help(1)
statistics. ustat: get file system . . . . ustat(2)
crontab: user crontab utility. . . . . crontab(1)

modification times. utime: set file access and . . utime(2)
utmp, wtmp: utmp and wtmp entry formats. utmp(4)
endutent, utmpname: access utmp file entry. /setutent, . . getut(3C)
ttyslot: find the slot in the utmp file of the current user. . ttyslot(3C)
formats. utmp, wtmp: utmp and wtmp entry utmp(4)
/pututline, setutent, endutent, utmpname: access utmp file entry. getut(3C)
clean-up. uuclean: uucp spool directory . uuclean(1M)
uusub: monitor uucp network. . . . . uusub(1M)
uuclean: uucp spool directory clean-up. uuclean(1M)
control. uustat: uucp status inquiry and job . . uustat(1C)
to UNIX system copy. uucp, uulog, uuname: UNIX system uucp(1C)
file for transmission/ uuencode, uudecode: encode/decode a binary uuencode(1C)
a binary file for transmission/ uuencode, uudecode: encode/decode uuencode(1C)
UNIX system copy. uucp, uulog, uuname: UNIX system to uucp(1C)
system copy. uucp, uulog, uuname: UNIX system to UNIX uucp(1C)
system file copy. uuto, uupick: public UNIX-to-UNIX uuto(1C)
host. uusend: send a file to a remote uusend(1C)
job control. uustat: uucp status inquiry and uustat(1C)
uusub: monitor uucp network. uusub(1M)
system file copy. uuto, uupick: public UNIX-to-UNIX uuto(1C)
execution. uux: UNIX-to-UNIX system command uux(1C)
configuration information. uvar: returns system-specific . uvar(2)
dir: format of System V directories. . . . . dir(4)
inode: format of a System V inode. . . . . inode(4)
fs: format of a System V system volume. . . . . fs(4)
val: validate SCCS file. . . . val(1)
val: validate SCCS file. . . . . val(1)
/u3b5, u3b15, vax: provide truth value about processor type. . machid(1)
abs: return integer absolute value. . . . . abs(3C)
cabs, zabs: Fortran absolute value. abs, iabs, dabs, . . . . abs(3F)
getenv: return value for environment name. . getenv(3C)
ceiling, remainder, absolute value functions. /fabs: floor, . floor(3M)
readlink: read value of a symbolic link. . . readlink(2)
getitimer, setitimer: get/set value of interval timer. . . . getitimer(2)
YP map. ypmatch: print the value of one or more keys from a ypmatch(1)
putenv: change or add value to environment. . . . . putenv(3C)
/htons, ntohl, ntohs: convert values between host and network/ byteorder(3N)
ypcat: print values in a YP data base. . . ypcat(1)
values: machine-dependent values. values(5)
true, false: provide truth values. . . . . true(1)
values: machine-dependent values. . . . . values(5)
print formatted output of a varargs argument list. /vsprintf: vprintf(3S)
list. varargs: handle variable argument varargs(3X)
varargs: handle variable argument list. . . . varargs(3X)
return Fortran environment variable. getenv: . . . . . getenv(3F)
/pdp11, u3b, u3b2, u3b5, u3b15, vax: provide truth value about/ machid(1)
vc: version control. . . . . vc(1)
get option letter from argument vector. getopt: . . . . . getopt(3C)
display editor. vi, view, vedit: screen-oriented (visual) vi(1)
assert: verify program assertion. . . assert(3X)
expand tabs to spaces, and vice versa. expand, unexpand: . . expand(1)

vc: version control. . . . . vc(1)
version: reports version number of files. . . . version(1)
server host. yppoll: what version of a YP map is at a YP yppoll(1M)
get: get a version of an SCCS file. . . . get(1)
of files. version: reports version number version(1)
status of local machines (RPC version). rup: show host . . . rup(1C)
logged in on local machines (RPC version). rusers: who's . . . rusers(1N)
sccsdiff: compare two versions of an SCCS file. . . sccsdiff(1)
formatted output of a/ vprintf, vfprintf, vsprintf: print . . . vprintf(3S)
(visual) display editor. vi, view, vedit: screen-oriented vi(1)
a binary file for transmission via mail. /encode/decode . . uuencode(1C)
expand tabs to spaces, and vice versa. expand, unexpand: expand(1)
mvt: typeset view graphs and slides. . . . mvt(1)
(visual) display editor. vi, view, vedit: screen-oriented . vi(1)
macro package for typesetting viewgraphs and slides. /a troff mv(5)
more: file perusal filter for CRT viewing. . . . . more(1)
vipw: edit the password file. . vipw(1M)
vi, view, vedit: screen-oriented (visual) display editor. . . . vi(1)
systems with label checking. volcopy, labelit: copy file . . volcopy(1M)
fs: format of a System V system volume. . . . . fs(4)
print formatted output of a/ vprintf, vfprintf, vsprintf: . . vprintf(3S)
of a varargs/ vprintf, vfprintf, vsprintf: print formatted output vprintf(3S)
doing. w: who is on and what they are w(1)
terminate. wait: wait for child process to stop or wait(2)
terminate. wait3: wait for child process to stop or wait3(2N)
release blocked signals and wait for interrupt. /atomically sigpause(2)
stop or terminate. wait: wait for child process to wait(2)
stop or terminate. wait3: wait for child process to wait3(2N)
ftw: walk a file tree. . . . . ftw(3C)
wall: write to all users. . . . wall(1M)
wc: word count. . . . . wc(1)
what: identify SCCS files. . . what(1)
whodo: who is doing what. . . . . whodo(1M)
is. whatis: describe what a command whatis(1)
and/or manual for program. whereis: locate source, binary, whereis(1)
including aliases and paths. which: locate a program file . which(1)
who: who is on the system. . . who(1)
user ID. whoami: print effective current whoami(1)
whodo: who is doing what. . . whodo(1M)
(RPC version). rusers: who's logged in on local machines rusers(1N)
machines. rwho: who's logged in on local . . . rwho(1N)
prof: profile within a function. . . . . prof(5)
wc: word count. . . . . wc(1)
fgetc, getw: get character or word from a stream. /getchar, getc(3S)
hangman: guess the word. . . . . hangman(6)
fputc, putw: put character or word on a stream. putc, putchar, putc(3S)
hyphen: find hyphenated words. . . . . hyphen(1)
diction. diction, explain: print wordy sentences; thesaurus for diction(1)
chdir: change working directory. . . . . chdir(2)
getcwd: get pathname of current working directory. . . . . getcwd(3C)
pwd: print working directory name. . . . pwd(1)

getwd: get current working directory pathname. . getwd(3)
worm: play the growing worm game. . . . . worm(6)
worm: play the growing worm game. worm(6)
terminal. worms: animate worms on a display worms(6)
worms: animate worms on a display terminal. . worms(6)
write, writev: write on a file. . . . . write(2)
putpwent: write password file entry. . . putpwent(3C)
running B-NET software. rwall: write to all users over a network rwall(1M)
wall: write to all users. . . . . wall(1M)
write: write to another user. . . . . write(1)
write: write to another user. . write(1)
write, writev: write on a file. . write(2)
write, writev: write on a file. . . . . write(2)
file regions for reading or writing. /provide exclusive . locking(2)
open: open for reading or writing. . . . . open(2)
utmp, wtmp: utmp and wtmp entry formats. . . . . utmp(4)
formats. utmp, wtmp: utmp and wtmp entry . utmp(4)
accounting records. fwtmp, wtmpfix: manipulate connect . fwtmp(1M)
hunt-the-wumpus. wump: the game of . . . . . wump(6)
and execute command. xargs: construct argument list(s) xargs(1)
external data representation. xdr: library routines for . . . xdr(3N)
bitwise boolean/ and, or, xor, not, lshift, rshift: Fortran . bool(3F)
programs to implement shared/ xstr: extract strings from C . . xstr(1)
j0, j1, jn, y0, y1, yn: Bessel functions. . bessel(3M)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . bessel(3M)
compiler-compiler. yacc: yet another . . . . . yacc(1)
ypclnt: yellow pages client interface. . ypclnt(3N)
ypinit: build and install yellow pages database. . . . ypinit(1M)
ypmake: rebuild yellow pages database. . . . ypmake(1M)
makedbm: make a yellow pages dbm file. . . . makedbm(1M)
yppasswdd: server for modifying yellow pages password file. . yppasswdd(1M)
processes. ypserv, ypbind: yellow pages server and binder ypserv(1M)
change login password in yellow pages. yppasswd: . . yppasswd(1)
directory/ ypfiles: the yellowpages database and . . ypfiles(4)
yes: be repetitively affirmative. yes(1)
yacc: yet another compiler-compiler. yacc(1)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)
ypcat: print values in a YP data base. . . . . ypcat(1)
here. ypxfr: transfer a YP map from some YP server to ypxfr(1M)
yppoll: what version of a YP map is at a YP server host. yppoll(1M)
value of one or more keys from a YP map. ypmatch: print the . ypmatch(1)
force propagation of a changed YP map. yppush: . . . . . yppush(1M)
what version of a YP map is at a YP server host. yppoll: . . . yppoll(1M)
ypwhich: which host is the YP server or map master?. . . ypwhich(1)
transfer a YP map from some YP server to here. ypxfr: . . ypxfr(1M)
ypset: point ypbind at a particular server. . ypset(1M)
binder processes. ypserv, ypbind: yellow pages server and ypserv(1M)
base. ypcat: print values in a YP data ypcat(1)
interface. ypclnt: yellow pages client . . ypclnt(3N)
and directory structure. ypfiles: the yellowpages database ypfiles(4)
pages database. ypinit: build and install yellow ypinit(1M)

database. ypmake: rebuild yellow pages ypmake(1M)
or more keys from a YP map. ypmatch: print the value of one ypmatch(1)
in yellow pages. yppasswd: change login password yppasswd(1)
yellow pages password file. yppasswdd: server for modifying yppasswdd(1M)
is at a YP server host. yppoll: what version of a YP map yppoll(1M)
changed YP map. yppush: force propagation of a yppush(1M)
server and binder processes. ypserv, ypbind: yellow pages . ypserv(1M)
particular server. ypset: point ypbind at a . . . ypset(1M)
server or map master?. ypwhich: which host is the YP ypwhich(1)
some YP server to here. ypxfr: transfer a YP map from ypxfr(1M)
abs, iabs, dabs, cabs, zabs: Fortran absolute value. . abs(3F)
bzb: format of Block Zero Blocks. . . . . bzb(4)
tzic: time zone compiler. . . . . tzic(1M)
tzdump: time zone dumper. . . . . tzdump(1M)
tzfile: time zone information. . . . . tzfile(4)