NAME

intro − introduction to system calls and error numbers

SYNOPSIS

#include <errno.h>

DESCRIPTION

This section describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible return value. This is almost always −1; the individual descriptions specify the details.

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

The following is a complete list of the errors and their names as given in <*errno.h*>.

0 Error 0
: Unused.

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

2 ENOENT No such file or directory
: This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

3 ESRCH No such process
: The process whose number was given to *kill* and *ptrace* does not exist, or is already dead.

4 EINTR Interrupted system call
: An asynchronous signal (such as interrupt or quit), which the user 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 occurred during a *read* or *write*. This error may in some cases occur on a call following the one to which it actually applies.

6 ENXIO No such device or address
: I/O on a special file refers to a subdevice which does not exist, or beyond the limits of the device. It may also occur when, for example, an illegal tape drive unit number is selected or a disk pack is not loaded on a drive.

7 E2BIG Arg list too long
: An argument list longer than 10240 bytes is presented to *execve*.

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*(5).

9 EBADF Bad file number
: Either a file descriptor refers to no open file, or a read (resp. write) request is made to a file which is open only for writing (resp. reading).

10 ECHILD No children
: *Wait* and the process has no living or unwaited-for children.

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

12 ENOMEM Not enough core
: During an *execve* or *break,* a program asks for more core or swap space than the system is able to supply. A lack of swap space is normally a temporary condition, however a lack of core is not a temporary condition; the maximum size of the text, data, and stack segments is a system parameter.

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

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

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

16 EBUSY Mount device busy
: An attempt to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file directory. (open file, current directory, mounted-on file, active text segment).

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

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

19 ENODEV No such device
: An attempt was made to apply an inappropriate system call to a device; e.g. read a write-only device.

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

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

22 EINVAL Invalid argument
: Some invalid argument: dismounting a non-mounted device, mentioning an unknown signal in *signal,* reading or writing a file for which *seek* has generated a negative pointer. Also set by math functions, see *intro*(3).

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

24 EMFILE Too many open files
: Customary configuration limit is 20 per process.

25 ENOTTY Not a typewriter
: The file mentioned in an *ioctl* is not a terminal or one of the other devices to which these calls apply.

26 ETXTBSY Text file busy
: An attempt to execute a pure-procedure program which is currently open for writing (or reading!). Also an attempt to open for writing a pure-procedure program that is being executed.

27 EFBIG File too large
The size of a file exceeded the maximum (about 10^9 bytes).

28 ENOSPC No space left on device
During a *write* to an ordinary file, there is no free space left on the device.

29 ESPIPE Illegal seek
An *lseek* was issued to a pipe. This error may also be issued for other non-seekable devices.

30 EROFS Read-only file system
An attempt to modify a file or directory was made on a device mounted read-only.

31 EMLINK Too many links
An attempt to make more than 32767 hard links to a file.

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

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

34 ERANGE Result too large
The value of a function in the math package (3M) is unrepresentable within machine precision.

35 EWOULDBLOCK Operation would block
An operation which would cause a process to block was attempted on a object in non-blocking mode (see *ioctl* (2)).

36 EINPROGRESS Operation now in progress
An operation which takes a long time to complete (such as a *connect* (2)) was attempted on a non-blocking object (see *ioctl* (2)).

37 EALREADY Operation already in progress
An operation was attempted on a non-blocking object which already had an operation in progress.

38 ENOTSOCK Socket operation on non-socket
Self-explanatory.

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

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

41 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 ARPA Internet UDP protocol with type SOCK_STREAM.

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

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

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

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

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

47 EAFNOSUPPORT Address family not supported by protocol family
An address incompatible with the requested protocol was used. For example, you shouldn't necessarily expect to be able to use PUP Internet addresses with ARPA Internet protocols.

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

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

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

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

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

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

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

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

56 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.

57 ENOTCONN Socket is not connected
An request to send or receive data was disallowed because the socket is not connected.

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

59 *unused*

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

61 ECONNREFUSED Connection refused
No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service which is inactive on the foreign host.

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

63 ENAMETOOLONG File name too long
: A component of a path name exceeded 255 characters, or an entire path name exceeded 1023 characters.

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

DEFINITIONS

Process ID

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

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

Tty Group ID

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

Real User ID and Real Group ID

Each user on the system is identified by a positive integer termed the real user ID.

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

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

Effective User Id, Effective Group Id, and Access Groups

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

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

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

Super-user

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

Special Processes

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

Descriptor

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

File Name

Names consisting of up to {FILENAME_MAX} characters may be used to name an ordinary file, special file, or directory.

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

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

Path Name

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

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

Directory

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

Root Directory and Current Working Directory

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

File Access Permissions

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

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

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

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

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

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

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

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

Otherwise, permission is denied.

Sockets and Address Families

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

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

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

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

SEE ALSO

intro(3), perror(3)

NAME

accept − accept a connection on a socket

SYNOPSIS

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

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

DESCRIPTION

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

The argument *addr* is a result parameter 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*(2) a socket for the purposes of doing an *accept* by selecting it for read.

RETURN VALUE

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

ERRORS

The *accept* will fail if:

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

SEE ALSO

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

NAME

access − determine accessibility of file

SYNOPSIS

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

#define R_OK    4    /* test for read permission */
#define W_OK    2    /* test for write permission */
#define X_OK    1    /* test for execute (search) permission */
#define F_OK    0    /* test for presence of file */

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

DESCRIPTION

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

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

Notice that only access bits are checked. A directory may be indicated as writable by *access*, but an attempt to open it for writing will fail (although files may be created there); a file may look executable, but *execve* will fail unless it is in proper format.

RETURN VALUE

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

ERRORS

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

[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	The argument path name was too long.
[ENOENT]	Read, write, or execute (search) permission is requested for a null path name or the named file does not exist.
[EPERM]	The argument contains a byte with the high-order bit set.
[ELOOP]	Too many symbolic links were encountered in translating the pathname.
[EROFS]	Write access is requested for a file on a read-only file system.
[ETXTBSY]	Write access is requested for a pure procedure (shared text) file that is being executed.
[EACCES]	Permission bits of the file mode do not permit the requested access; or search permission is denied on a component of the path prefix. The owner of a file has permission checked with respect to the "owner" read, write, and execute mode bits, members of the file's group other than the owner have permission checked with respect to the "group" mode bits, and all others have permissions checked with respect to the "other" mode bits.
[EFAULT]	*Path* points outside the process's allocated address space.

SEE ALSO

chmod(2), stat(2)

NAME

acct − turn accounting on or off

SYNOPSIS

acct(file)
char *file;

DESCRIPTION

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

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

This call is permitted only to the super-user.

NOTES

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

RETURN VALUE

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

ERRORS

Acct will fail if one of the following is true:

[EPERM]	The caller is not the super-user.
[EPERM]	The pathname contains a character with the high-order bit set.
[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	The named file does not exist.
[EISDIR]	The named file is a directory.
[EROFS]	The named file resides on a read-only file system.
[EFAULT]	*File* points outside the process's allocated address space.
[ELOOP]	Too many symbolic links were encountered in translating the pathname.
[EACCES]	The file is a character or block special file.

SEE ALSO

acct(5), sa(8)

BUGS

No accounting is produced for programs running when a crash occurs. In particular nonterminating programs are never accounted for.

NAME

bind − bind a name to a socket

SYNOPSIS

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

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

DESCRIPTION

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

NOTES

Binding a name in the UNIX domain creates a socket in the file system which must be deleted by the caller when it is no longer needed (using *unlink*(2)). The file created is a side-effect of the current implementation, and will not be created in future versions of the UNIX ipc domain.

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

RETURN VALUE

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

ERRORS

The *bind* call will fail if:

[EBADF]	*S* is not a valid descriptor.
[ENOTSOCK]	*S* is not a socket.
[EADDRNOTAVAIL]	The specified address is not available from the local machine.
[EADDRINUSE]	The specified address is already in use.
[EINVAL]	The socket is already bound to an address.
[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(2), listen(2), socket(2), getsockname(2)

NAME

brk, sbrk – change data segment size

SYNOPSIS

caddr_t brk(addr)
caddr_t addr;

caddr_t sbrk(incr)
int incr;

DESCRIPTION

Brk sets the system's idea of the lowest data segment location not used by the program (called the break) to *addr* (rounded up to the next multiple of the system's page size). Locations greater than *addr* and below the stack pointer are not in the address space and will thus cause a memory violation if accessed.

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

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

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

RETURN VALUE

Zero is returned if the *brk* could be set; −1 if the program requests more memory than the system limit. *Sbrk* returns −1 if the break could not be set.

ERRORS

Sbrk will fail and no additional memory will be allocated if one of the following are true:

[ENOMEM]	The limit, as set by *setrlimit*(2), was exceeded.
[ENOMEM]	The maximum possible size of a data segment (compiled into the system) was exceeded.
[ENOMEM]	Insufficient space existed in the swap area to support the expansion.

SEE ALSO

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

BUGS

Setting the break may fail due to a temporary lack of swap space. It is not possible to distinguish this from a failure caused by exceeding the maximum size of the data segment without consulting *getrlimit.*

NAME

chdir − change current working directory

SYNOPSIS

chdir(path)
char *path;

DESCRIPTION

Path is the pathname of a directory. *Chdir* causes this directory to become the current working directory, the starting point for path names not beginning with "/".

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

RETURN VALUE

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

ERRORS

Chdir will fail and the current working directory will be unchanged if one or more of the following are true:

[ENOTDIR]	A component of the pathname is not a directory.
[ENOENT]	The named directory does not exist.
[ENOENT]	The argument path name was too long.
[EPERM]	The argument contains a byte with the high-order bit set.
[EACCES]	Search permission is denied for any component of the path name.
[EFAULT]	*Path* points outside the process's allocated address space.
[ELOOP]	Too many symbolic links were encountered in translating the pathname.

SEE ALSO

chroot(2)

NAME

chmod − change mode of file

SYNOPSIS

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

fchmod(fd, mode)
int fd, mode;
```

DESCRIPTION

The file whose name is given by *path* or referenced by the descriptor *fd* has its mode changed to *mode*. Modes are constructed by *or*'ing together some combination of the following:

```
04000 set user ID on execution
02000 set group ID on execution
01000 save text image after execution
00400 read by owner
00200 write by owner
00100 execute (search on directory) by owner
00070 read, write, execute (search) by group
00007 read, write, execute (search) by others
```

If an executable file is set up for sharing (this is the default) then mode 1000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Ability to set this bit is restricted to the super-user.

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

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

RETURN VALUE

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

ERRORS

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

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

Fchmod will fail if:

[EBADF]	The descriptor is not valid.
[EINVAL]	*Fd* refers to a socket, not to a file.

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

SEE ALSO

open(2), chown(2)

NAME

chown − change owner and group of a file

SYNOPSIS

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

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

DESCRIPTION

The file which is named by *path* or referenced by *fd* has its *owner* and *group* changed as specified. Only the super-user may execute this call, because if users were able to give files away, they could defeat the file-space accounting procedures.

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

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

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]	The effective user ID does not match the owner of the file and the effective user ID is not the super-user.
[EROFS]	The named file resides on a read-only file system.
[EFAULT]	*Path* points outside the process's allocated address space.
[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

chmod(2), flock(2)

NAME

chroot − change root directory

SYNOPSIS

chroot(dirname)
char *dirname;

DESCRIPTION

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

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

This call is restricted to the super-user.

RETURN VALUE

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

ERRORS

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

[ENOTDIR]	A component of the path name is not a directory.
[ENOENT]	The pathname was too long.
[EPERM]	The argument contains a byte with the high-order bit set.
[ENOENT]	The named directory does not exist.
[EACCES]	Search permission is denied for any component of the path name.
[EFAULT]	*Path* points outside the process's allocated address space.
[ELOOP]	Too many symbolic links were encountered in translating the pathname.

SEE ALSO

chdir(2)

NAME

close − delete a descriptor

SYNOPSIS

close(d)
int d;

DESCRIPTION

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

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

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

RETURN VALUE

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

ERRORS

Close will fail if:

[EBADF]	*D* is not an active descriptor.

SEE ALSO

accept(2), flock(2), open(2), pipe(2), socket(2), socketpair(2), execve(2), fcntl(2)

NAME
connect — initiate a connection on a socket

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

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

DESCRIPTION
The parameter *s* is a socket. If it is of type SOCK_DGRAM, then this call 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
The call fails if:

[EBADF]	*S* is not a valid descriptor.
[ENOTSOCK]	*S* is a descriptor for a file, not a socket.
[EADDRNOTAVAIL]	The specified address is not available on this machine.
[EAFNOSUPPORT]	Addresses in the specified address family cannot be used with this socket.
[EISCONN]	The socket is already connected.
[ETIMEDOUT]	Connection establishment timed out without establishing a connection.
[ECONNREFUSED]	The attempt to connect was forcefully rejected.
[ENETUNREACH]	The network isn't reachable from this host.
[EADDRINUSE]	The address is already in use.
[EFAULT]	The *name* parameter specifies an area outside the process address space.
[EWOULDBLOCK]	The socket is non-blocking and the and the connection cannot be completed immediately. It is possible to *select*(2) the socket while it is connecting by selecting it for writing.

SEE ALSO
accept(2), select(2), socket(2), getsockname(2)

NAME

creat − create a new file

SYNOPSIS

creat(name, mode)
char *name;

DESCRIPTION

This interface is obsoleted by open(2).

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

If the file did exist, its mode and owner remain unchanged but it is truncated to 0 length.

The file is also opened for writing, and its file descriptor is returned.

NOTES

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

RETURN VALUE

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

ERRORS

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

[EPERM]	The argument contains a byte with the high-order bit set.
[ENOTDIR]	A component of the path prefix is not a directory.
[EACCES]	A needed directory does not have search permission.
[EACCES]	The file does not exist and the directory in which it is to be created is not writable.
[EACCES]	The file exists, but it is unwritable.
[EISDIR]	The file is a directory.
[EMFILE]	There are already too many files open.
[EROFS]	The named file resides on a read-only file system.
[ENXIO]	The file is a character special or block special file, and the associated device does not exist.
[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed.
[EFAULT]	*Name* points outside the process's allocated address space.
[ELOOP]	Too many symbolic links were encountered in translating the pathname.
[EOPNOTSUPP]	The file was a socket (not currently implemented).

SEE ALSO

open(2), write(2), close(2), chmod(2), umask(2)

NAME

dup, dup2 — duplicate a descriptor

SYNOPSIS

newd = dup(oldd)
int newd, oldd;

dup2(oldd, newd)
int oldd, newd;

DESCRIPTION

Dup duplicates an existing object descriptor. The argument *oldd* is a small non-negative integer index in the per-process descriptor table. The value must be less than the size of the table, which is returned by *getdtablesize*(2). The new descriptor *newd* 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 *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.

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

RETURN VALUE

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

ERRORS

Dup and *dup2* fail if:

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

SEE ALSO

accept(2), open(2), close(2), pipe(2), socket(2), socketpair(2), getdtablesize(2)

NAME

execve − execute a file

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

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

Ignored signals remain ignored across an *execve*, but signals that are caught are reset to their default values. The signal stack is reset to be undefined; see *sigvec*(2) for more information.

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

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

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

When the executed program begins, it is called as follows:

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

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

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

RETURN VALUE

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

ERRORS

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

[ENOENT]	One or more components of the new process file's path name do not exist.
[ENOTDIR]	A component of the new process file 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 execute permission.
[ENOEXEC]	The new process file has the appropriate access permission, but has an invalid magic number in its header.
[ETXTBSY]	The new process file is a pure procedure (shared text) file that is currently open for writing or reading by some process.
[ENOMEM]	The new process requires more virtual memory than is allowed by the imposed maximum (*getrlimit*(2)).
[E2BIG]	The number of bytes in the new process's argument list is larger than the system-imposed limit of {ARG_MAX} bytes.
[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.

CAVEATS

If a program is *setuid* to a non-super-user, but is executed when the real *uid* is "root", then the program has the powers of a super-user as well.

SEE ALSO

exit(2), fork(2), execl(3), environ(7)

NAME

_exit − terminate a process

SYNOPSIS

_exit(status)
int status;

DESCRIPTION

_exit terminates a process with the following consequences:

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

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

The parent process ID of all of the calling process's existing child processes are also set to 1. This means that the initialization process (see *intro*(2)) inherits each of these processes as well.

Most C programs call the library routine *exit*(3) which performs cleanup actions in the standard i/o library before calling *_exit*.

RETURN VALUE

This call never returns.

SEE ALSO

fork(2), wait(2), exit(3)

NAME

fcntl − file control

SYNOPSIS

#include <fcntl.h>

res = fcntl(fd, cmd, arg)
int res;
int fd, cmd, arg;

DESCRIPTION

Fcntl provides for control over descriptors. The argument *fd* is a descriptor to be operated on by *cmd* as follows:

F_DUPFD	Return a new descriptor as follows:
	Lowest numbered available descriptor greater than or equal to *arg*.
	Same object references as the original descriptor.
	New descriptor shares the same file pointer if the object was a file.
	Same access mode (read, write or read/write).
	Same file status flags (i.e., both file descriptors share the same file status flags).
	The close-on-exec flag associated with the new file descriptor is set to remain open across *execv*(2) system calls.
F_GETFD	Get the close-on-exec flag associated with the file descriptor *fd*. If the low-order bit is 0, the file will remain open across *exec*, otherwise the file will be closed upon execution of *exec*.
F_SETFD	Set the close-on-exec flag associated with *fd* to the low order bit of *arg* (0 or 1 as above).
F_GETFL	Get descriptor status flags, as described below.
F_SETFL	Set descriptor status flags.
F_GETOWN	Get the process ID or process group currently receiving SIGIO and SIGURG signals; process groups are returned as negative values.
F_SETOWN	Set the process or process group to receive SIGIO and SIGURG signals; process groups are specified by supplying *arg* as negative, otherwise *arg* is interpreted as a process ID.

The flags for the F_GETFL and F_SETFL flags are as follows:

FNDELAY	Non-blocking I/O; if no data is available to a *read* call, or if a write operation would block, the call returns -1 with the error EWOULDBLOCK.
FAPPEND	Force each write to append at the end of file; corresponds to the O_APPEND flag of *open*(2).
FASYNC	Enable the SIGIO signal to be sent to the process group when I/O is possible, e.g. upon availability of data to be read.

RETURN VALUE

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

F_DUPFD	A new file descriptor.
F_GETFD	Value of flag (only the low-order bit is defined).
F_GETFL	Value of flags.
F_GETOWN	Value of file descriptor owner.

other	Value other than −1.

Otherwise, a value of −1 is returned and *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 allowed number of file descriptors are currently open.
[EINVAL]	*Cmd* is F_DUPFD and *arg* is negative or greater the maximum allowable number (see *getdtablesize*(2)).

SEE ALSO

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

BUGS

The asynchronous I/O facilities of FNDELAY and FASYNC are currently available only for tty operations. No SIGIO signal is sent upon draining of output sufficiently for non-blocking writes to occur.

NAME

flock – apply or remove an advisory lock on an open file

SYNOPSIS

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

#define LOCK_SH    1    /* shared lock */
#define LOCK_EX    2    /* exclusive lock */
#define LOCK_NB    4    /* don't block when locking */
#define LOCK_UN    8    /* unlock */

flock(fd, operation)
int fd, operation;
```

DESCRIPTION

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

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

The locking mechanism allows two types of locks: *shared* locks and *exclusive* locks. At any time multiple shared locks may be applied to a file, but at no time are multiple exclusive, or both shared and exclusive, locks allowed simultaneously on a file.

A shared lock may be *upgraded* to an exclusive lock, and vice versa, simply by specifying the appropriate lock type; this results in the previous lock being released and the new lock applied (possibly after other processes have gained and released the lock).

Requesting a lock on an object which is already locked normally causes the caller to blocked until the lock may be acquired. If LOCK_NB is included in *operation*, then this will not happen; instead the call will fail and the error EWOULDBLOCK will be returned.

NOTES

Locks are on files, not file descriptors. That is, file descriptors duplicated through *dup*(2) or *fork*(2) do not result in multiple instances of a lock, but rather multiple references to a single lock. If a process holding a lock on a file forks and the child explicitly unlocks the file, the parent will lose its lock.

Processes blocked awaiting a lock may be awakened by signals.

RETURN VALUE

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

ERRORS

The *flock* call fails if:

[EWOULDBLOCK]	The file is locked and the LOCK_NB option was specified.
[EBADF]	The argument *fd* is an invalid descriptor.
[EINVAL]	The argument *fd* refers to an object other than a file.

SEE ALSO

open(2), close(2), dup(2), execve(2), fork(2)

NAME

fork — create a new process

SYNOPSIS

pid = fork()
int pid;

DESCRIPTION

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

The child process has a unique process ID.

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

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

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

RETURN VALUE

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

ERRORS

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

[EAGAIN]	The system-imposed limit {PROC_MAX} on the total number of processes under execution would be exceeded.
[EAGAIN]	The system-imposed limit {KID_MAX} on the total number of processes under execution by a single user would be exceeded.

SEE ALSO

execve(2), wait(2)

NAME

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

SYNOPSIS

fsync(fd)
int fd;

DESCRIPTION

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

Fsync should be used by programs 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

The *fsync* fails if:

[EBADF]	*Fd* is not a valid descriptor.
[EINVAL]	*Fd* refers to a socket, not to a file.

SEE ALSO

sync(2), sync(8), update(8)

BUGS

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

NAME

getdtablesize — get descriptor table size

SYNOPSIS

nds = getdtablesize()
int nds;

DESCRIPTION

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

SEE ALSO

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

NAME

getgid, getegid − get group identity

SYNOPSIS

gid = getgid()
int gid;

egid = getegid()
int egid;

DESCRIPTION

Getgid returns the real group ID of the current process, *getegid* the effective group ID.

The real group ID is specified at login time.

The effective group ID is more transient, and determines additional access permission during execution of a "set-group-ID" process, and it is for such processes that *getgid* is most useful.

SEE ALSO

getuid(2), setregid(2), setgid(3)

NAME

getgroups – get group access list

SYNOPSIS

#include <sys/param.h>

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

DESCRIPTION

Getgroups gets the current group access list of the user process and stores it in the array *gidset*. The parameter *ngroups* indicates the number of entries which may be placed in *gidset*. No more than NGROUPS, as defined in <*sys/param.h*>, will ever be returned.

RETURN VALUE

Getgroups returns the number of groups put in *gidset*. A value of 0 or more indicates that the call succeeded. A value of –1 indicates that an error occurred, and the error code is stored in the global variable *errno*. If an error occurs, nothing valid is returned in *gidset*.

ERRORS

The possible errors for *getgroup* are:

[EFAULT]	The argument *gidset* specifies an invalid address.
[EINVAL]	The argument *ngroups* is less than the number of groups that could be returned.

SEE ALSO

setgroups(2), initgroups(3)

NAME

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

SYNOPSIS

hostid = gethostid()
int hostid;

sethostid(hostid)
int hostid;

DESCRIPTION

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

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

SEE ALSO

hostid(1), gethostname(2)

BUGS

32 bits for the identifier is too small.

NAME

gethostname, sethostname — get/set name of current host

SYNOPSIS

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

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

DESCRIPTION

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

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

RETURN VALUE

If the call succeeds a value of 0 is returned. If the call fails, then a value of −1 is returned and an error code is placed int 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 super-user.

SEE ALSO

gethostid(2)

BUGS

Host names are limited to 255 characters.

NAME

getitimer, setitimer − get/set value of interval timer

SYNOPSIS

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

#define ITIMER_REAL      0     /* real time intervals */
#define ITIMER_VIRTUAL   1     /* virtual time intervals */
#define ITIMER_PROF      2     /* user and system virtual time */

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

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

DESCRIPTION

The system provides each process with three interval timers, defined in <*sys/time.h*>. The *getitimer* call returns the current value for the timer specified in *which*, while the *setitimer* call sets the value of a timer (optionally returning the previous value of the timer).

A timer value is defined by the *itimerval* structure:

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

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

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

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

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

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

NOTES

Three macros for manipulating time values are defined in <*sys/time.h*>. *Timerclear* sets a time value to zero, *timerisset* tests if a time value is non-zero, and *timercmp* compares two time values (beware that >= and <= do not work with this macro).

RETURN VALUE

If the calls succeed, a value of 0 is returned. If an error occurs, the value −1 is returned, and a more precise error code is placed in the global variable *errno*.

ERRORS

The possible errors are:

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

SEE ALSO

sigvec(2), gettimeofday(2)

NAME

getpagesize — get system page size

SYNOPSIS

pagesize = getpagesize()
int pagesize;

DESCRIPTION

Getpagesize returns the number of bytes in a page. Page granularity is the granularity of many of the memory management calls.

The page size is a *system* page size and may not be the same as the underlying hardware page size.

SEE ALSO

sbrk(2), pagesize(1)

NAME

getpeername − get name of connected peer

SYNOPSIS

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

DESCRIPTION

Getpeername returns the name of the peer connected to socket *s*. The *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes).

DIAGNOSTICS

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

ERRORS

The call succeeds unless:

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

SEE ALSO

bind(2), socket(2), getsockname(2)

BUGS

Names bound to sockets in the UNIX domain are inaccessible; *getpeername* returns a zero length name.

NAME

getpgrp − get process group

SYNOPSIS

pgrp = getpgrp(pid)
int prgp;
int pid;

DESCRIPTION

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

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

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

SEE ALSO

setpgrp(2), getuid(2), tty(4)

NAME

getpid, getppid — get process identification

SYNOPSIS

```
pid = getpid()
long pid;

ppid = getppid()
long ppid;
```

DESCRIPTION

Getpid returns the process ID of the current process. Most often it is used with the host identifier *gethostid*(2) to generate uniquely-named temporary files.

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

SEE ALSO

gethostid(2)

NAME

getpriority, setpriority − get/set program scheduling priority

SYNOPSIS

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

#define PRIO_PROCESS     0        /* process */
#define PRIO_PGRP        1        /* process group */
#define PRIO_USER        2        /* user id */

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

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

DESCRIPTION

The scheduling priority of the process, process group, or user, as indicated by *which* and *who* is obtained with the *getpriority* call and set with the *setpriority* call. *Which* is one of PRIO_PROCESS, PRIO_PGRP, or PRIO_USER, and *who* is interpreted relative to *which* (a process identifier for PRIO_PROCESS, process group identifier for PRIO_PGRP, and a user ID for PRIO_USER). *Prio* is a value in the range −20 to 20. The default priority is 0; lower priorities cause more favorable scheduling.

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

RETURN VALUE

Since *getpriority* can legitimately return the value −1, it is necessary to clear the external variable *errno* prior to the call, then check it afterward to determine if a −1 is an error or a legitimate value. The *setpriority* call returns 0 if there is no error, or −1 if there is.

ERRORS

Getpriority and *setpriority* may return one of the following errors:

[ESRCH]	No process(es) were located using the *which* and *who* values specified.
[EINVAL]	*Which* was not one of PRIO_PROCESS, PRIO_PGRP, or PRIO_USER.

In addition to the errors indicated above, *setpriority* may fail with one of the following errors returned:

[EACCES]	A process was located, but neither its effective nor real user ID matched the effective user ID of the caller.
[EACCES]	A non super-user attempted to change a process priority to a negative value.

SEE ALSO

nice(1), fork(2), renice(8)

NAME

getrlimit, setrlimit − control maximum system resource consumption

SYNOPSIS

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

getrlimit(resource, rlp)
int resource;
struct rlimit *rlp;

setrlimit(resource, rlp)
int resource;
struct rlimit *rlp;
```

DESCRIPTION

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

The *resource* parameter is one of the following:

RLIMIT_CPU	the maximum amount of cpu time (in milliseconds) to be used by each process.
RLIMIT_FSIZE	the largest size, in bytes, of any single file which may be created.
RLIMIT_DATA	the maximum size, in bytes, of the data segment for a process; this defines how far a program may extend its break with the *sbrk*(2) system call.
RLIMIT_STACK	the maximum size, in bytes, of the stack segment for a process; this defines how far a program's stack segment may be extended, either automatically by the system, or explicitly by a user with the *sbrk*(2) system call.
RLIMIT_CORE	the largest size, in bytes, of a *core* file which may be created.
RLIMIT_RSS	the maximum size, in bytes, a process's resident set size may grow to. This imposes a limit on the amount of physical memory to be given to a process; if memory is tight, the system will prefer to take memory from processes which are exceeding their declared resident set size.

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

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

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

An "infinite" value for a limit is defined as RLIMIT_INFINITY (0x7fffffff).

Because this information is stored in the per-process information, this system call must be executed directly by the shell if it is to affect all future processes created by the shell; *limit* is thus a built-in command to *csh*(1).

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way: a *break* call fails if the data space limit is reached, or the process is killed when the stack limit is reached (since the stack cannot be extended, there is no way to send a signal!).

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

RETURN VALUE

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

ERRORS

The possible errors are:

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

SEE ALSO

csh(1), quota(2)

BUGS

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

NAME

getrusage — get information about resource utilization

SYNOPSIS

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

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

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

DESCRIPTION

Getrusage returns information describing the resources utilized by the current process, or all its terminated child processes. The *who* parameter is one of RUSAGE_SELF and RUSAGE_CHILDREN. If *rusage* is non-zero, the buffer it points to will be filled in with the following structure:

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

The fields are interpreted as follows:

ru_utime	the total amount of time spent executing in user mode.
ru_stime	the total amount of time spent in the system executing on behalf of the process(es).
ru_maxrss	the maximum resident set size utilized (in kilobytes).
ru_ixrss	an "integral" value indicating the amount of memory used which was also shared among other processes. This value is expressed in units of kilobytes * seconds-of-execution and is calculated by summing the number of shared memory pages in use each time the internal system clock ticks and then averaging over 1 second intervals.
ru_idrss	an integral value of the amount of unshared memory residing in the data segment of a process (expressed in units of kilobytes * seconds-of-execution).
ru_isrss	an integral value of the amount of unshared memory residing in the stack segment of a process (expressed in units of kilobytes * seconds-of-execution).
ru_minflt	the number of page faults serviced without any i/o activity; here i/o activity is

	avoided by "reclaiming" a page frame from the list of pages awaiting reallocation.
ru_majflt	the number of page faults serviced which required i/o activity.
ru_nswap	the number of times a process was "swapped" out of main memory.
ru_inblock	the number of times the file system had to perform input.
ru_outblock	the number of times the file system had to perform output.
ru_msgsnd	the number of ipc messages sent.
ru_msgrcv	the number of ipc messages received.
ru_nsignals	the number of signals delivered.
ru_nvcsw	the number of times a context switch resulted due to a process voluntarily giving up the processor before its time slice was completed (usually to await availability of a resource).
ru_nivcsw	the number of times a context switch resulted due to a higher priority process becoming runnable or because the current process exceeded its time slice.

NOTES

The numbers *ru_inblock* and *ru_outblock* account only for real i/o; data supplied by the cacheing mechanism is charged only to the first process to read or write the data.

SEE ALSO

gettimeofday(2), wait(2)

BUGS

There is no way to obtain information about a child process which has not yet terminated.

NAME

getsockname − get socket name

SYNOPSIS

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

DESCRIPTION

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

DIAGNOSTICS

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

ERRORS

The call succeeds unless:

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

SEE ALSO

bind(2), socket(2)

BUGS

Names bound to sockets in the UNIX domain are inaccessible; *getsockname* returns a zero length name.

NAME

getsockopt, setsockopt − get and set options on sockets

SYNOPSIS

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

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

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

DESCRIPTION

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

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

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

Optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file *<sys/socket.h>* contains definitions for "socket" level options; see *socket*(2). Options at other protocol levels vary in format and name, consult the appropriate entries in (4P).

RETURN VALUE

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

ERRORS

The call succeeds unless:

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

SEE ALSO

socket(2), getprotoent(3N)

NAME

gettimeofday, settimeofday − get/set date and time

SYNOPSIS

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

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

settimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;
```

DESCRIPTION

Gettimeofday returns the system's notion of the current Greenwich time and the current time zone. Time returned is expressed relative in seconds and microseconds since midnight January 1, 1970.

The structures pointed to by *tp* and *tzp* are defined in <*sys/time.h*> as:

```
struct timeval {
        u_long  tv_sec;         /* seconds since Jan. 1, 1970 */
        long    tv_usec;                /* and microseconds */
};

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

The *timezone* structure indicates the local time zone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time applies locally during the appropriate part of the year.

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

RETURN

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

ERRORS

The following error codes may be set in *errno*:

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

SEE ALSO

date(1), ctime(3)

BUGS

Time is never correct enough to believe the microsecond values. There should a mechanism by which, at least, local clusters of systems might synchronize their clocks to millisecond granularity.

NAME

getuid, geteuid — get user identity

SYNOPSIS

uid = getuid()
int uid;

euid = geteuid()
int euid;

DESCRIPTION

Getuid returns the real user ID of the current process, *geteuid* the effective user ID.

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

SEE ALSO

getgid(2), setreuid(2)

NAME

ioctl — control device

SYNOPSIS

#include <sys/ioctl.h>

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

DESCRIPTION

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

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

RETURN VALUE

If an error has occurred, a value of −1 is returned and *errno* is set to indicate the error.

ERRORS

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

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

SEE ALSO

execve(2), fcntl(2), mt(4), tty(4), intro(4N)

NAME

kill − send signal to a process

SYNOPSIS

kill(pid, sig)
int pid, sig;

DESCRIPTION

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

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

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

If the process number is −1, and the user is the super-user, the signal is broadcast universally except to system processes and the process sending the signal.

Processes may send signals to themselves.

RETURN VALUE

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

ERRORS

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

[EINVAL]	*Sig* is not a valid signal number.
[ESRCH]	No process can be found corresponding to that specified by *pid*.
[EPERM]	The sending process is not the super-user and its effective user id does not match the effective user-id of the receiving process.

SEE ALSO

getpid(2), getpgrp(2), killpg(2), sigvec(2)

NAME

killpg − send signal to a process group

SYNOPSIS

killpg(pgrp, sig)
int pgrp, sig;

DESCRIPTION

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

The sending process and members of the process group must have the same effective user ID, otherwise this call is restricted to the super-user. 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 *pid.*
[EPERM]	The sending process is not the super-user and one or more of the target processes has an effective user ID different from that of the sending process.

SEE ALSO

kill(2), getpgrp(2), sigvec(2)

NAME

link − make a hard link to a file

SYNOPSIS

link(name1, name2)
char *name1, *name2;

DESCRIPTION

A hard link to *name1* is created; the link has the name *name2*. *Name1* must exist.

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

RETURN VALUE

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

ERRORS

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

[EPERM]	Either pathname contains a byte with the high-order bit set.
[ENOENT]	Either pathname was too long.
[ENOTDIR]	A component of either path prefix is not a directory.
[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 *name1* does not exist.
[EEXIST]	The link named by *name2* does exist.
[EPERM]	The file named by *name1* is a directory and the effective user ID is not super-user.
[EXDEV]	The link named by *name2* and the file named by *name1* are on different file systems.
[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]	One of the pathnames specified is outside the process's allocated address space.
[ELOOP]	Too many symbolic links were encountered in translating the pathname.

SEE ALSO

symlink(2), unlink(2)

NAME

listen — listen for connections on a socket

SYNOPSIS

listen(s, backlog)
int s, backlog;

DESCRIPTION

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

The *backlog* parameter defines the maximum length the queue of pending connections may grow to. If a connection request arrives with the queue full the client will receive an error with an indication of ECONNREFUSED.

RETURN VALUE

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

ERRORS

The call fails if:

[EBADF]	The argument *s* is not a valid descriptor.
[ENOTSOCK]	The argument *s* is not a socket.
[EOPNOTSUPP]	The socket is not of a type that supports the operation *listen.*

SEE ALSO

accept(2), connect(2), socket(2)

BUGS

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

NAME

lseek − move read/write pointer

SYNOPSIS

```
#define L_SET    0    /* set the seek pointer */
#define L_INCR   1    /* increment the seek pointer */
#define L_XTND   2    /* extend the file size */

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

DESCRIPTION

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

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

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

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

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

NOTES

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

RETURN VALUE

Upon successful completion, a non-negative integer, the current file pointer value, is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

ERRORS

Lseek will fail and the file pointer will remain unchanged if:

[EBADF]	*Fildes* is not an open file descriptor.
[ESPIPE]	*Fildes* is associated with a pipe or a socket.
[EINVAL]	*Whence* is not a proper value.
[EINVAL]	The resulting file pointer would be negative.

SEE ALSO

dup(2), open(2)

BUGS

This document's use of *whence* is incorrect English, but maintained for historical reasons.

NAME

mkdir − make a directory file

SYNOPSIS

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

DESCRIPTION

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

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

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

RETURN VALUE

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

ERRORS

Mkdir will fail and no directory will be created if:

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

chmod(2), stat(2), umask(2)

NAME

mknod − make a special file

SYNOPSIS

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

DESCRIPTION

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

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

Mknod may be invoked only by the super-user.

RETURN VALUE

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

ERRORS

Mknod will fail and the file mode will be unchanged if:

[EPERM]	The process's effective user ID is not super-user.
[EPERM]	The pathname contains a character 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.

SEE ALSO

chmod(2), stat(2), umask(2)

NAME

mount, umount — mount or remove file system

SYNOPSIS

mount(special, name, rwflag)
char *special, *name;
int rwflag;

umount(special)
char *special;

DESCRIPTION

Mount announces to the system that a removable file system has been mounted on the block-structured special file *special;* from now on, references to file *name* will refer to the root file on the newly mounted file system. *Special* and *name* are pointers to null-terminated strings containing the appropriate path names.

Name must exist already. *Name* must be a directory. Its old contents are inaccessible while the file system is mounted.

The *rwflag* argument determines whether the file system can be written on; if it is 0 writing is allowed, if non-zero no writing is done. 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.

Umount announces to the system that the *special* file is no longer to contain a removable file system. The associated file reverts to its ordinary interpretation.

RETURN VALUE

Mount returns 0 if the action occurred, −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.

Umount returns 0 if the action occurred; −1 if if the special file is inaccessible or does not have a mounted file system, or if there are active files in the mounted file system.

ERRORS

Mount will fail when one of the following occurs:

[NODEV]	The caller is not the super-user.
[NODEV]	*Special* does not exist.
[ENOTBLK]	*Special* is not a block device.
[ENXIO]	The major device number of *special* is out of range (this indicates no device driver exists for the associated hardware).
[EPERM]	The pathname contains a character with the high-order bit set.
[ENOTDIR]	A component of the path prefix in *name* is not a directory.
[EROFS]	*Name* resides on a read-only file system.
[EBUSY]	*Name* 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.
[EBUSY]	An i/o error occurred while reading the super block or cylinder group information.

Umount may fail with one of the following errors:

[NODEV]	The caller is not the super-user.
[NODEV]	*Special* does not exist.
[ENOTBLK]	*Special* is not a block device.
[ENXIO]	The major device number of *special* is out of range (this indicates no device driver exists for the associated hardware).
[EINVAL]	The requested device is not in the mount table.
[EBUSY]	A process is holding a reference to a file located on the file system.

SEE ALSO

mount(8), umount(8)

BUGS

The error codes are in a state of disarray; too many errors appear to the caller as one value.

NAME

open − open a file for reading or writing, or create a new file

SYNOPSIS

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

open(path, flags, mode)
char *path;
int flags, mode;
```

DESCRIPTION

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

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

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

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

Upon successful completion a non-negative integer termed a file descriptor is returned. The file pointer used to mark the current position within the file is set to the beginning of the file.

The new descriptor is set to remain open across *execve* system calls; see *close*(2).

No process may have more than {OPEN_MAX} file descriptors open simultaneously.

ERRORS

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

[EPERM]	The pathname contains a character with the high-order bit set.
[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	O_CREAT is not set and the named file does not exist.
[EACCES]	A component of the path prefix denies search permission.
[EACCES]	The required permissions (for reading and/or writing) are denied for the named flag.
[EISDIR]	The named file is a directory, and the arguments specify it is to be opened for writting.
[EROFS]	The named file resides on a read-only file system, and the file is to be modified.

[EMFILE]	{OPEN_MAX} file descriptors are currently open.
[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 the *open* call requests write access.
[EFAULT]	*Path* points outside the process's allocated address space.
[ELOOP]	Too many symbolic links were encountered in translating the pathname.
[EEXIST]	O_EXCL was specified and the file exists.
[ENXIO]	The O_NDELAY flag is given, and the file is a communications device on which their is no carrier present.
[EOPNOTSUPP]	An attempt was made to open a socket (not currently implemented).

SEE ALSO

chmod(2), close(2), dup(2), lseek(2), read(2), write(2), umask(2)

NAME

pipe − create an interprocess communication channel

SYNOPSIS

pipe(fildes)
int fildes[2];

DESCRIPTION

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

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

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

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

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

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

RETURN VALUE

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

ERRORS

The *pipe* call will fail if:

[EMFILE]	Too many descriptors are active.
[EFAULT]	The *fildes* buffer is in an invalid area of the process's address space.

SEE ALSO

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

BUGS

Should more than 4096 bytes be necessary in any pipe among a loop of processes, deadlock will occur.

NAME

profil — execution time profile

SYNOPSIS

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

DESCRIPTION

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

The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0x10000 gives a 1-1 mapping of pc's to words in *buff;* 0x8000 maps each pair of instruction words together. 0x2 maps all instructions onto the beginning of *buff* (producing a non-interrupting 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 *execve* is executed, but remains on in child and parent both after a *fork*. Profiling is turned off if an update in *buff* would cause a memory fault.

RETURN VALUE

A 0, indicating success, is always returned.

SEE ALSO

gprof(1), setitimer(2), monitor(3)

NAME

ptrace − process trace

SYNOPSIS

#include <signal.h>

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

DESCRIPTION

Ptrace provides a means by which a parent process may control the execution of a child process, and examine and change its core image. Its primary use is for the implementation of breakpoint debugging. There are four arguments whose interpretation depends on a *request* argument. Generally, *pid* is the process ID of the traced process, which must be a child (no more distant descendant) of the tracing process. A process being traced behaves normally until it encounters some signal whether internally generated like "illegal instruction" or externally generated like "interrupt". See *sigvec*(2) for the list. Then the traced process enters a stopped state and its parent is notified via *wait*(2). When the child is in the stopped state, its core image can be examined and modified using *ptrace.* If desired, another *ptrace* request can then cause the child either to terminate or to continue, possibly ignoring the signal.

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

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

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

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

4,5 The given *data* is written at the word in the process's address space corresponding to *addr,* which must be even. No useful value is returned. If I and D space are separated, request 4 indicates I space, 5 D space. Attempts to write in pure procedure fail if another process is executing the same file.

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

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

8 The traced process terminates.

9 Execution continues as in request 7; however, as soon as possible after execution of at least one instruction, execution stops again. The signal number from the stop is SIGTRAP. (On the VAX-11 the T-bit is used and just one instruction is executed.) This is part of the mechanism for implementing breakpoints.

As indicated, these calls (except for request 0) can be used only when the subject process has stopped. The *wait* call is used to determine when a process stops; in such a case the "termination" status returned by *wait* has the value 0177 to indicate stoppage rather than genuine termination.

To forestall possible fraud, *ptrace* inhibits the set-user-id and set-group-id facilities on subsequent *execve*(2) calls. If a traced process calls *execve*, it will stop before executing the first instruction of the new image showing signal SIGTRAP.

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

RETURN VALUE

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

ERRORS

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

SEE ALSO

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

BUGS

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

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

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

It should be possible to stop a process on occurrence of a system call; in this way a completely controlled environment could be provided.

NAME

quota − manipulate disk quotas

SYNOPSIS

#include <sys/quota.h>

quota(cmd, uid, arg, addr)
int cmd, uid, arg;
caddr_t addr;

DESCRIPTION

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

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

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

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

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

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

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

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

RETURN VALUE

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

ERRORS

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

[EINVAL]	*Cmd* is invalid.

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

SEE ALSO

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

BUGS

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

The Australian spelling of *disk* is used throughout the quota facilities in honor of the implementors.

NAME

read, readv − read input

SYNOPSIS

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

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

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

DESCRIPTION

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

For *readv*, the *iovec* structure is defined as

```
struct iovec {
        caddr_t iov_base;
        int     iov_len;
};
```

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

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

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

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

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

RETURN VALUE

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

ERRORS

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

[EBADF]	*Fildes* is not a valid file descriptor open for reading.
[EFAULT]	*Buf* points outside the allocated address space.
[EINTR]	A read from a slow device was interrupted before any data arrived by the delivery of a signal.

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

dup(2), open(2), pipe(2), socket(2), socketpair(2)

NAME
readlink − read value of a symbolic link

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

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

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

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

[EPERM]	The *path* argument contained a byte with the high-order bit set.
[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 super-user.
[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 system or halt processor

SYNOPSIS

#include <sys/reboot.h>

reboot(howto)
int howto;

DESCRIPTION

Reboot reboots the system, and is invoked automatically in the event of unrecoverable system failures. *Howto* is a mask of options passed to the bootstrap program. The system call interface permits only RB_HALT or RB_AUTOBOOT to be passed to the reboot program; the other flags are used in scripts stored on the console storage media, or used in manual bootstrap procedures. When none of these options (e.g. RB_AUTOBOOT) is given, the system is rebooted from file "vmunix" in the root file system of unit 0 of a disk chosen in a processor specific way. An automatic consistency check of the disks is then normally performed.

The bits of *howto* are:

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

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

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

Only the super-user may *reboot* a machine.

RETURN VALUES

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

ERRORS

[EPERM]	The caller is not the super-user.

SEE ALSO

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

BUGS

The notion of "console medium", among other things, is specific to the VAX.

NAME

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

SYNOPSIS

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

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

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

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

DESCRIPTION

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

The *recv* call may be used only on a *connected* socket (see *connect*(2)), while *recvfrom* and *recvmsg* may be used to receive data on a socket whether it is in a connected state or not.

If *from* is non-zero, the source address of the message is filled in. *Fromlen* is a value-result parameter, initialized to the size of the buffer associated with *from*, and modified on return to indicate the actual size of the address stored there. The length of the message is returned in *cc*. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from; see *socket*(2).

If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is nonblocking (see *ioctl*(2)) in which case a *cc* of −1 is returned with the external variable errno set to EWOULDBLOCK.

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

The *flags* argument to a 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, as described in *read*(2). 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 non-blocking and the receive operation would block.
[EINTR]	The receive was interrupted by delivery of a signal before any data was available for the receive.
[EFAULT]	The data was specified to be received into a non-existent or protected part of the process address space.

SEE ALSO

read(2), send(2), socket(2)

NAME

rename − change the name of a file

SYNOPSIS

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

DESCRIPTION

Rename causes the link named *from* to be renamed as *to*. If *to* exists, then it is first removed. Both *from* and *to* must be of the same type (that is, both directories or both non-directories), and must reside on the same file system.

Rename guarantees that an instance of *to* will always exist, even if the system should crash in the middle of the operation.

CAVEAT

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

RETURN VALUE

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

ERRORS

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

[ENOTDIR]	A component of either path prefix is not a directory.
[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 super-user.
[EXDEV]	The link named by *to* and the file named by *from* are on different logical devices (file systems). Note that this error code will not be returned if the implementation permits cross-device links.
[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

open(2)

NAME

rmdir − remove a directory file

SYNOPSIS

rmdir(path)
char *path;

DESCRIPTION

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

RETURN VALUE

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

ERRORS

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

[ENOTEMPTY]	The named directory contains files other than "." and ".." in it.
[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 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.
[ELOOP]	Too many symbolic links were encountered in translating the pathname.

SEE ALSO

mkdir(2), unlink(2)

NAME

select − synchronous i/o multiplexing

SYNOPSIS

#include <sys/time.h>

nfound = select(nfds, readfds, writefds, execptfds, timeout)
int nfound, 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* desciptors 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 in *nfound*.

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

Any of *readfds*, *writefds*, and *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]	An signal was delivered before any of the selected for events occurred or the time limit expired.

SEE ALSO

accept(2), connect(2), read(2), write(2), recv(2), send(2)

BUGS

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

NAME

send, sendto, sendmsg − send a message from a socket

SYNOPSIS

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

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

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

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

DESCRIPTION

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

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

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

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

The *flags* parameter may be set to SOF_OOB to send "out-of-band" data on sockets which support this notion (e.g. SOCK_STREAM).

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

RETURN VALUE

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

ERRORS

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

SEE ALSO

recv(2), socket(2)

NAME

setgroups – set group access list

SYNOPSIS

#include <sys/param.h>

setgroups(ngroups, gidset)
int ngroups, *gidset;

DESCRIPTION

Setgroups sets the group access list of the current user process according to the array *gidset.* The parameter *ngroups* indicates the number of entries in the array and must be no more than NGRPS, as defined in <*sys/param.h*>.

Only the super-user may set new groups.

RETURN VALUE

A 0 value is returned on success, –1 on error, with a error code stored in *errno.*

ERRORS

The *setgroups* call will fail if:

[EPERM]	The caller is not the super-user.
[EFAULT]	The address specified for *gidset* is outside the process address space.

SEE ALSO

getgroups(2), initgroups(3X)

NAME

setpgrp − set process group

SYNOPSIS

setpgrp(pid, pgrp)
int pid, pgrp;

DESCRIPTION

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

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

RETURN VALUE

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

ERRORS

Setpgrp will fail and the process group will not be altered if one of the following occur:

[ESRCH]	The requested process does not exist.
[EPERM]	The effective user ID of the requested process is different from that of the caller and the process is not a descendent of the calling process.

SEE ALSO

getpgrp(2)

NAME

setquota − enable/disable quotas on a file system

SYNOPSIS

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

DESCRIPTION

Disc quotas are enabled or disabled with the *setquota* call. *Special* indicates a block special device on which a mounted file system exists. If *file* is nonzero, it specifies a file in that file system from which to take the quotas. If *file* is 0, then quotas are disabled on the file system. The quota file must exist; it is normally created with the *checkquota*(8) program.

Only the super-user may turn quotas on or off.

SEE ALSO

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

RETURN VALUE

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

ERRORS

Setquota will fail when one of the following occurs:

[NODEV]	The caller is not the super-user.
[NODEV]	*Special* does not exist.
[ENOTBLK]	*Special* is not a block device.
[ENXIO]	The major device number of *special* is out of range (this indicates no device driver exists for the associated hardware).
[EPERM]	The pathname contains a character with the high-order bit set.
[ENOTDIR]	A component of the path prefix in *file* is not a directory.
[EROFS]	*File* resides on a read-only file system.
[EACCES]	*File* resides on a file system different from *special.*
[EACCES]	*File* is not a plain file.

BUGS

The error codes are in a state of disarray; too many errors appear to the caller as one value.

NAME

setregid − set real and effective group ID

SYNOPSIS

setregid(rgid, egid)
int rgid, egid;

DESCRIPTION

The real and effective group ID's of the current process are set to the arguments. Only the super-user may change the real group ID of a process. Unpriviledged 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 super-user and a change other than changing the effective group-id to the real group-id was specified.

SEE ALSO

getgid(2), setreuid(2), setgid(3)

NAME

setreuid − set real and effective user ID's

SYNOPSIS

setreuid(ruid, euid)
int ruid, euid;

DESCRIPTION

The real and effective user ID's of the current process are set according to the arguments. If *ruid* or *euid* is −1, the current uid is filled in by the system. Only the super-user may modify the real uid of a process. Users other than the super-user may change the effective uid of a process only to the real uid.

RETURN VALUE

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

ERRORS

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

SEE ALSO

getuid(2), setregid(2), setuid(3)

NAME

shutdown − shut down part of a full-duplex connection

SYNOPSIS

shutdown(s, how)
int s, how;

DESCRIPTION

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

DIAGNOSTICS

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

ERRORS

The call succeeds unless:

[EBADF]	*S* is not a valid descriptor.
[ENOTSOCK]	*S* is a file, not a socket.
[ENOTCONN]	The specified socket is not connected.

SEE ALSO

connect(2), socket(2)

NAME

sigblock -- block signals

SYNOPSIS

sigblock(mask);
int mask;

DESCRIPTION

Sigblock causes the signals specified in *mask* to be added to the set of signals currently being blocked from delivery. Signal *i* is blocked if the *i*-th bit in *mask* is a 1. Bits are numbered starting at one; for example, to block SIGALRM use

```
oldmask = sigblock(1 << (SIGALRM - 1));
```

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

RETURN VALUE

The previous set of masked signals is returned.

SEE ALSO

kill(2), sigvec(2), sigsetmask(2),

NAME

sigpause — atomically release blocked signals and wait for interrupt

SYNOPSIS

sigpause(sigmask)
int sigmask;

DESCRIPTION

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

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

SEE ALSO

sigblock(2), sigvec(2)

NAME

sigsetmask − set current signal mask

SYNOPSIS

sigsetmask(mask);
int mask;

DESCRIPTION

Sigsetmask sets the current signal mask (those signals which are blocked from delivery). Signal *i* is blocked if the *i*-th bit in *mask* is a 1.

The system quietly disallows SIGKILL, SIGSTOP, or SIGCONT to be blocked.

RETURN VALUE

The previous set of masked signals is returned.

SEE ALSO

kill(2), sigvec(2), sigblock(2), sigpause(2)

NAME

sigstack − set and/or get signal stack context

SYNOPSIS

```
#include <signal.h>

struct sigstack {
      caddr_t   ss_sp;
      int       ss_onstack;
};

sigstack(ss, oss);
struct sigstack *ss, *oss;
```

DESCRIPTION

Sigstack allows users to define an alternate stack on which signals are to be processed. If *ss* is non-zero, it specifies a *signal stack* on which to deliver signals and tells the system if the process is currently executing on that stack. When a signal's action indicates its handler should execute on the signal stack (specified with a *sigvec*(2) call), the system checks to see if the process is currently executing on that stack. If the process is not currently executing on the signal stack, the system arranges a switch to the signal stack for the duration of the signal handler's execution. If *oss* is non-zero, the current signal stack state is returned.

NOTES

Signal stacks are not "grown" automatically, as is done for the normal stack. If the stack overflows unpredictable results may occur.

RETURN VALUE

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

ERRORS

Sigstack will fail and the signal stack context will remain unchanged if one of the following occurs.

[EFAULT]	Either *ss* or *oss* points to memory which is not a valid part of the process address space.

SEE ALSO

sigvec(2), setjmp(3)

NAME

sigvec – software signal facilities

SYNOPSIS

```
#include <signal.h>

struct sigvec {
        int         (*sv_handler)();
        int         sv_mask;
        int         sv_onstack;
};

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 occurence 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 initilized from that of its parent (normally 0). It may be changed with a *sigblock*(2) or *sigsetmask*(2) call, or when a signal is delivered to the process.

When a signal condition arises for a process, the signal is added to a set of signals pending for the process. If the signal is not currently *blocked* by the process then it is delivered to the process. When a signal is delivered, the current state of the process is saved, a new signal mask is calculated (as described below), and the signal handler is invoked. The call to the handler is arranged so that if the signal handling routine returns normally the process will resume execution in the context from before the signal's delivery. If the process wishes to resume in a different context, then it must arrange to restore the previous context itself.

When a signal is delivered to a process a new signal mask is installed for the duration of the process' signal handler (or until a *sigblock* or *sigsetmask* call is made). This mask is formed by taking the current signal mask, adding the signal to be delivered, and *or*'ing in the signal mask associated with the handler to be invoked.

Sigvec assigns a handler for a specific signal. If *vec* is non-zero, it specifies a handler routine and mask to be used when delivering the specified signal. Further, if *sv_onstack* is 1, the system will deliver the signal to the process on a *signal stack*, specified with *sigstack*(2). If *ovec* is non-zero, the previous handling information for the signal is returned to the user.

The following is a list of all signals with names as in the include file <*signal.h*>:

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction
SIGTRAP	5*	trace trap
SIGIOT	6*	IOT instruction
SIGEMT	7*	EMT instruction
SIGFPE	8*	floating point exception

SIGKILL	9	kill (cannot be caught, blocked, or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGURG	16●	urgent condition present on socket
SIGSTOP	17†	stop (cannot be caught, blocked, or ignored)
SIGTSTP	18†	stop signal generated from keyboard
SIGCONT	19●	continue after stop (cannot be blocked)
SIGCHLD	20●	child status has changed
SIGTTIN	21†	background read attempted from control terminal
SIGTTOU	22†	background write attempted to control terminal
SIGIO	23●	i/o is possible on a descriptor (see *fcntl*(2))
SIGXCPU	24	cpu time limit exceeded (see *setrlimit*(2))
SIGXFSZ	25	file size limit exceeded (see *setrlimit*(2))
SIGVTALRM	26	virtual time alarm (see *setitimer*(2))
SIGPROF	27	profiling timer alarm (see *setitimer*(2))

The starred signals in the list above cause a core image if not caught or ignored.

Once a signal handler is installed, it remains installed until another *sigvec* call is made, or an *execve*(2) is performed. The default action for a signal may be reinstated by setting *sv_handler* to SIG_DFL; this default is termination (with a core image for starred signals) except for signals marked with ● or †. Signals marked with ● are discarded if the action is SIG_DFL; signals marked with † cause the process to stop. If *sv_handler* is SIG_IGN the signal is subsequently ignored, and pending instances of the signal are discarded.

If a caught signal occurs during certain system calls, causing the call to terminate prematurely, the call is automatically restarted. In particular this can occur during a *read* or *write*(2) on a slow device (such as a terminal; but not a file) and during a *wait*(2).

After a *fork*(2) or *vfork*(2) the child inherits all signals, the signal mask, and the signal stack.

Execve(2) resets all caught signals to default action; ignored signals remain ignored; the signal mask remains the same; the signal stack state is reset.

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 occured 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 which is not a valid part of the process address space.
[EINVAL]	*Sig* is not a valid signal number.
[EINVAL]	An attempt is made to ignore or supply a handler for SIGKILL or SIGSTOP.
[EINVAL]	An attempt is made to ignore SIGCONT (by default SIGCONT is ignored).

SEE ALSO

kill(1), ptrace(2), kill(2), sigblock(2), sigsetmask(2), sigpause(2) sigstack(2), sigvec(2), setjmp(3), tty(4)

NOTES (VAX-11)

The handler routine can be declared:

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

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

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

Hardware condition	Signal	Code
Arithmetic traps:		
Integer overflow	SIGFPE	FPE_INTOVF_TRAP
Integer division by zero	SIGFPE	FPE_INTDIV_TRAP
Floating overflow trap	SIGFPE	FPE_FLTOVF_TRAP
Floating/decimal division by zero	SIGFPE	FPE_FLTDIV_TRAP
Floating underflow trap	SIGFPE	FPE_FLTUND_TRAP
Decimal overflow trap	SIGFPE	FPE_DECOVF_TRAP
Subscript-range	SIGFPE	FPE_SUBRNG_TRAP
Floating overflow fault	SIGFPE	FPE_FLTOVF_FAULT
Floating divide by zero fault	SIGFPE	FPE_FLTDIV_FAULT
Floating underflow fault	SIGFPE	FPE_FLTUND_FAULT
Length access control	SIGSEGV	
Protection violation	SIGBUS	
Reserved instruction	SIGILL	ILL_RESAD_FAULT
Customer-reserved instr.	SIGEMT	
Reserved operand	SIGILL	ILL_PRIVIN_FAULT
Reserved addressing	SIGILL	ILL_RESOP_FAULT
Trace pending	SIGTRAP	
Bpt instruction	SIGTRAP	
Compatibility-mode	SIGILL	hardware supplied code
Chme	SIGSEGV	
Chms	SIGSEGV	
Chmu	SIGSEGV	

BUGS

This manual page is confusing.

NAME

socket — create an endpoint for communication

SYNOPSIS

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

s = socket(af, type, protocol)
int s, 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), and
AF_IMPLINK	(IMP "host at IMP" addresses).

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 super-user, 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*(3N) and *protocols*(3N).

Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. A stream socket must be in a *connected* state before any data may be sent or received on it. A connection to another socket is created with a *connect*(2) call. Once connected, data may be transferred using *read*(2) and *write*(2) calls or some variant of the *send*(2) and *recv*(2) calls. When a session has been completed a *close*(2) may be performed. Out-of-band data may also be transmitted as described in *send*(2) and received as described in *recv*(2).

The communications protocols used to implement a SOCK_STREAM insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with −1 returns and with ETIMEDOUT as the specific code in the global variable errno. The protocols optionally keep sockets "warm" by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for a extended period (e.g. 5 minutes). A SIGPIPE signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

SOCK_DGRAM and SOCK_RAW sockets allow sending of datagrams to correspondents named in *send*(2) calls. It is also possible to receive datagrams at such a socket with *recv*(2).

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*(2) 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 the rules used in validating addresses supplied in a *bind*(2) call should allow reuse of local addresses. SO_KEEPALIVE enables the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified via a SIGPIPE signal. SO_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address. SO_LINGER and SO_DONTLINGER control the actions taken when unsent messags 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(2), bind(2), connect(2), getsockname(2), getsockopt(2), ioctl(2), listen(2), recv(2), select(2), send(2), shutdown(2), socketpair(2)
"A 4.2BSD Interprocess Communication Primer".

BUGS

The use of keepalives is a questionable feature for this layer.

NAME

socketpair − create a pair of connected sockets

SYNOPSIS

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

socketpair(d, type, protocol, sv)
int d, type, protocol;
int sv[2];

DESCRIPTION

The *socketpair* call creates an unnamed pair of connected sockets in the specified domain *d*, of the specified *type*, and using the optionally specified *protocol*. The descriptors used in referencing the new sockets are returned in *sv*[0] and *sv*[1]. The two sockets are indistinguishable.

DIAGNOSTICS

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

ERRORS

The call succeeds unless:

[EMFILE]	Too many descriptors are in use by this process.
[EAFNOSUPPORT]	The specified address family is not supported on this machine.
[EPROTONOSUPPORT]	The specified protocol is not supported on this machine.
[EOPNOSUPPORT]	The specified protocol does not support creation of socket pairs.
[EFAULT]	The address *sv* does not specify a valid part of the process address space.

SEE ALSO

read(2), write(2), pipe(2)

BUGS

This call is currently implemented only for the UNIX domain.

NAME

stat, lstat, fstat − get file status

SYNOPSIS

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

stat(path, buf)
char *path;
struct stat *buf;

lstat(path, buf)
char *path;
struct stat *buf;

fstat(fd, buf)
int fd;
struct stat *buf;
```

DESCRIPTION

Stat obtains information about the file *path*. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be reachable.

Lstat is like *stat* except in the case where the named file is a symbolic link, in which case *lstat* returns information about the link, while *stat* returns information about the file the link references.

Fstat obtains the same information about an open file referenced by the argument descriptor, such as would be obtained by an *open* call.

Buf is a pointer to a *stat* structure into which information is placed concerning the file. The contents of the structure pointed to by *buf*

```
struct stat {
            dev_t       st_dev;      /* device inode resides on */
            ino_t       st_ino;      /* this inode's number */
            u_short     st_mode;     /* protection */
            short       st_nlink;    /* number or hard links to the file */
            short       st_uid;      /* user-id of owner */
            short       st_gid;      /* group-id of owner */
            dev_t       st_rdev;     /* the device type, for inode that is device */
            off_t       st_size;     /* total size of file */
            time_t      st_atime;    /* file last access time */
            int         st_spare1;
            time_t      st_mtime;    /* file last modify time */
            int         st_spare2;
            time_t      st_ctime;    /* file last status change time */
            int         st_spare3;
            long        st_blksize;  /* optimal blocksize for file system i/o ops */
            long        st_blocks;   /* actual number of blocks allocated */
            long        st_spare4[2];
};
```

st_atime	Time when file data was last read or modified. Changed by the following system calls: *mknod*(2), *utimes*(2), *read*(2), and *write*(2). For reasons of efficiency, st_atime is not set when a directory is searched, although this would be more logical.

st_mtime	Time when data was last modified. It is not set by changes of owner, group, link count, or mode. Changed by the following system calls: *mknod*(2), *utimes*(2), *write*(2).
st_ctime	Time when file status was last changed. It is set both both by writing and changing the i-node. Changed by the following system calls: *chmod*(2) *chown*(2), *link*(2), *mknod*(2), *unlink*(2), *utimes*(2), *write*(2).

The status information word *st_mode* has bits:

```
#define S_IFMT     0170000   /* type of file */
#define   S_IFDIR  0040000   /* directory */
#define   S_IFCHR  0020000   /* character special */
#define   S_IFBLK  0060000   /* block special */
#define   S_IFREG  0100000   /* regular */
#define   S_IFLNK  0120000   /* symbolic link */
#define   S_IFSOCK 0140000   /* socket */
#define S_ISUID    0004000   /* set user id on execution */
#define S_ISGID    0002000   /* set group id on execution */
#define S_ISVTX    0001000   /* save swapped text even after use */
#define S_IREAD    0000400   /* read permission, owner */
#define S_IWRITE   0000200   /* write permission, owner */
#define S_IEXEC    0000100   /* execute/search permission, owner */
```

The mode bits 0000070 and 0000007 encode group and others permissions (see *chmod*(2)).

When *fd* is associated with a pipe, *fstat* reports an ordinary file with an i-node number, restricted permissions, and a not necessarily meaningful length.

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]	The pathname contains a character with the high-order bit set.
[ENOENT]	The pathname was too long.
[ENOENT]	The named file does not exist.
[EACCES]	Search permission is denied for a component of the path prefix.
[EFAULT]	*Buf* or *name* points to an invalid address.

Fstat will fail if one or both of the following are true:

[EBADF]	*Fildes* is not a valid open file descriptor.
[EFAULT]	*Buf* points to an invalid address.
[ELOOP]	Too many symbolic links were encountered in translating the pathname.

CAVEAT

The fields in the stat structure currently marked *st_spare1*, *st_spare2*, and *st_spare3* are present in preparation for inode time stamps expanding to 64 bits. This, however, can break certain programs which depend on the time stamps being contiguous (in calls to *utimes*(2)).

SEE ALSO

chmod(2), chown(2), utimes(2)

BUGS

Applying *fstat* to a socket returns a zero'd buffer.

The list of calls which modify the various fields should be carefully checked with reality.

NAME

swapon — add a swap device for interleaved paging/swapping

SYNOPSIS

swapon (special)
char *special;

DESCRIPTION

Swapon makes the block device *special* available to the system for allocation for paging and swapping. The names of potentially available devices are known to the system and defined at system configuration time. The size of the swap area on *special* is calculated at the time the device is first made available for swapping.

SEE ALSO

swapon(8), config(8)

BUGS

There is no way to stop swapping on a disk so that the pack may be dismounted.

This call will be upgraded in future versions of the system.

NAME

symlink − make symbolic link to a file

SYNOPSIS

symlink(name1, name2)
char *name1, *name2;

DESCRIPTION

A symbolic link *name2* is created to *name1* (*name2* is the name of the file created, *name1* is the string used in creating the symbolic link). Either name may be an arbitrary path name; the files need not be on the same file system.

RETURN VALUE

Upon successful completion, a zero value is returned. If an error occurs, the error code is stored in *errno* and a −1 value is returned.

ERRORS

The symbolic link is made unless on or more of the following are true:

[EPERM]	Either *name1* or *name2* contains a character with the high-order bit set.
[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.
[ELOOP]	Too may symbolic links were encountered in translating the pathname.

SEE ALSO

link(2), ln(1), unlink(2)

NAME

sync − update super-block

SYNOPSIS

sync()

DESCRIPTION

Sync causes all information in core memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

Sync should be used by programs which examine a file system, for example *fsck, df,* etc. *Sync* is mandatory before a boot.

SEE ALSO

fsync(2), sync(8), update(8)

BUGS

The writing, although scheduled, is not necessarily complete upon return from *sync*.

NAME
syscall − indirect system call

SYNOPSIS
syscall(number, arg, ...) (VAX-11)

DESCRIPTION
Syscall performs the system call whose assembly language interface has the specified *number,* register arguments *r0* and *r1* and further arguments *arg.*

The r0 value of the system call is returned.

DIAGNOSTICS
When the C-bit is set, *syscall* returns −1 and sets the external variable *errno* (see *intro*(2)).

BUGS
There is no way to simulate system calls such as *pipe*(2), which return values in register r1.

NAME

truncate — truncate a file to a specified length

SYNOPSIS

truncate(path, length)
char *path;
int length;

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 succeeds unless:

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

Ftruncate succeeds unless:

[EBADF]	The *fd* is not a valid descriptor.
[EINVAL]	The *fd* references a socket, not a file.

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

umask — set file creation mode mask

SYNOPSIS

oumask = umask(numask)
int oumask, numask;

DESCRIPTION

Umask sets the process's file mode creation mask to *numask* and returns the previous value of the mask. The low-order 9 bits of *numask* are used whenever a file is created, clearing corresponding bits in the file mode (see *chmod*(2)). This clearing allows each user to restrict the default access to his files.

The value is initially 022 (write access for owner only). The mask is inherited by child processes.

RETURN VALUE

The previous value of the file mode mask is returned by the call.

SEE ALSO

chmod(2), mknod(2), open(2)

NAME

unlink − remove directory entry

SYNOPSIS

unlink(path)
char *path;

DESCRIPTION

Unlink removes the entry for the file *path* from its directory. If this entry was the last link to the file, and no process has the file open, then all resources associated with the file are reclaimed. If, however, the file was open in any process, the actual resource reclamation is delayed until it is closed, even though the directory entry has disappeared.

RETURN VALUE

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

ERRORS

The *unlink* succeeds unless:

[EPERM]	The path contains a character with the high-order bit set.
[ENOENT]	The path name is too long.
[ENOTDIR]	A component of the path prefix is not a directory.
[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.
[EPERM]	The named file is a directory and the effective user ID of the process is not the super-user.
[EBUSY]	The entry to be unlinked is the mount point for a mounted file system.
[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.

SEE ALSO

close(2), link(2), rmdir(2)

NAME

utimes — set file times

SYNOPSIS

#include <sys/time.h>

utimes(file, tvp)
char *file;
struct timeval tvp[2];

DESCRIPTION

The *utimes* call uses the "accessed" and "updated" times in that order from the *tvp* vector to set the corresponding recorded times for *file.*

The caller must be the owner of the file or the super-user. The "inode-changed" time of the file is set to the current time.

RETURN VALUE

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

ERRORS

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

[EPERM]	The pathname contained a character with the high-order bit set.
[ENOENT]	The pathname was too long.
[ENOENT]	The named file does not exist.
[ENOTDIR]	A component of the path prefix is not a directory.
[EACCES]	A component of the path prefix denies search permission.
[EPERM]	The process is not super-user and not the owner of the file.
[EACCES]	The effective user ID is not super-user 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]	*Tvp* points outside the process's allocated address space.
[ELOOP]	Too many symbolic links were encountered in translating the pathname.

SEE ALSO

stat(2)

NAME

vfork — spawn new process in a virtual memory efficient way

SYNOPSIS

pid = vfork()
int pid;

DESCRIPTION

Vfork can be used to create new processes without fully copying the address space of the old process, which is horrendously inefficient in a paged environment. It is useful when the purpose of *fork*(2) would have been to create a new system context for an *execve*. *Vfork* differs from *fork* in that the child borrows the parent's memory and thread of control until a call to *execve*(2) or an exit (either by a call to *exit*(2) or abnormally.) The parent process is suspended while the child is using its resources.

Vfork returns 0 in the child's context and (later) the pid of the child in the parent's context.

Vfork can normally be used just like *fork*. It does not work, however, to return while running in the childs context from the procedure which called *vfork* since the eventual return from *vfork* would then return to a no longer existent stack frame. Be careful, also, to call *_exit* rather than *exit* if you can't *execve*, since *exit* will flush and close standard I/O channels, and thereby mess up the parent processes standard I/O data structures. (Even with *fork* it is wrong to call *exit* since buffered data would then be flushed twice.)

SEE ALSO

fork(2), execve(2), sigvec(2), wait(2),

DIAGNOSTICS

Same as for *fork*.

BUGS

This system call will be eliminated when proper system sharing mechanisms are implemented. Users should not depend on the memory sharing semantics of *vfork* as it will, in that case, be made synonymous to *fork*.

To avoid a possible deadlock situation, processes which are children in the middle of a *vfork* are never sent SIGTTOU or SIGTTIN signals; rather, output or *ioctl*s are allowed and input attempts result in an end-of-file indication.

NAME

vhangup — virtually "hangup" the current control terminal

SYNOPSIS

vhangup()

DESCRIPTION

Vhangup is used by the initialization process *init*(8) (among others) to arrange that users are given "clean"' terminals at login, by revoking access of the previous users' processes to the terminal. To effect this, *vhangup* searches the system tables for references to the control terminal of the invoking process, revoking access permissions on each instance of the terminal which it finds. Further attempts to access the terminal by the affected processes will yield i/o errors (EBADF). Finally, a hangup signal (SIGHUP) is sent to the process group of the control terminal.

SEE ALSO

init (8)

BUGS

Access to the control terminal via **/dev/tty** is still possible.

This call should be replaced by an automatic mechanism which takes place on process exit.

NAME

wait, wait3 — wait for process to terminate

SYNOPSIS

#include <sys/wait.h>

pid = wait(status)
int pid;
union wait *status;

pid = wait(0)
int pid;

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

pid = wait3(status, options, rusage)
int pid;
union wait *status;
int options;
struct rusage *rusage;

DESCRIPTION

Wait causes its caller to delay until a signal is received or one of its child processes terminates. If any child has died since the last *wait*, return is immediate, returning the process id and exit status of one of the terminated children. If there are no children, return is immediate with the value −1 returned.

On return from a successful *wait* call, *status* is nonzero, and the high byte of *status* contains the low byte of the argument to *exit* supplied by the child process; the low byte of *status* contains the termination status of the process. A more precise definition of the *status* word is given in *<sys/wait.h>*.

Wait3 provides an alternate interface for programs 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 only children of the current process which are stopped due to a SIGTTIN, SIGTTOU, SIGTSTP, or SIGSTOP signal should have their status reported (WUNTRACED). If *rusage* is non-zero, a summary of the resources used by the terminated process and all its children is returned (this information is currently not available for stopped processes).

When the WNOHANG option is specified and no processes wish to report status, *wait3* returns a *pid* of 0. The WNOHANG and WUNTRACED options may be combined by *or*'ing the two values.

NOTES

See *sigvec*(2) for a list of termination statuses (signals); 0 status indicates normal termination. A special status (0177) is returned for a stopped process which has not terminated and can be restarted; see *ptrace*(2). If the 0200 bit of the termination status is set, a core image of the process was produced by the system.

If the parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

Wait and *wait3* are automatically restarted when a process receives a signal while awaiting termination of a child process.

RETURN VALUE

If *wait* returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

Wait3 returns −1 if there are no children not previously waited for; 0 is returned if WNOHANG is specified and there are no stopped or exited chiidren.

ERRORS

Wait will fail and return immediately if one or more of the following are true:

[ECHILD]	The calling process has no existing unwaited-for child processes.
[EFAULT]	The *status* or *rusage* arguments point to an illegal address.

SEE ALSO

exit(2)

NAME

write, writev − write on a file

SYNOPSIS

write(d, buf, nbytes)
int d;
char *buf;
int nbytes;

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

writev(d, iov, ioveclen)
int d;
struct iovec *iov;
int ioveclen;

DESCRIPTION

Write attempts to write *nbytes* of data to the object referenced by the descriptor *d* from the buffer pointed to by *buf*. *Writev* performs the same action, but gathers the output data from the *iovlen* buffers specified by the members of the *iovec* array: iov[0], iov[1], etc.

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

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

If the real user is not the super-user, then *write* clears the set-user-id bit on a file. This prevents penetration of system security by a user who "captures" a writable set-user-id file owned by the super-user.

RETURN VALUE

Upon successful completion the number of bytes actually writen is returned. Otherwise a −1 is returned and *errno* is set to indicate the error.

ERRORS

Write will fail and the file pointer will remain unchanged if one or more of the following are true:

[EBADF]	*D* is not a valid descriptor open for writing.
[EPIPE]	An attempt is made to write to a pipe that is not open for reading by any process.
[EPIPE]	An attempt is made to write to a socket of type SOCK_STREAM which is not connected to a peer socket.
[EFBIG]	An attempt was made to write a file that exceeds the process's file size limit or the maximum file size.
[EFAULT]	Part of *iov* or data to be written to the file points outside the process's allocated address space.

SEE ALSO

lseek(2), open(2), pipe(2)

NAME

intro — introduction to library functions

DESCRIPTION

This section describes functions that may be found in various libraries. The library functions are those other than the functions which directly invoke UNIX system primitives, described in section 2. This section has the libraries physically grouped together. This is a departure from older versions of the UNIX Programmer's Reference Manual, which did not group functions by library. The functions described in this section are grouped into various libraries:

(3) and (3S)
The straight "3" functions are the standard C library functions. The C library also includes all the functions described in section 2. The 3S functions comprise the standard I/O library. Together with the (3N), (3X), and (3C) routines, these functions constitute library *libc*, which is automatically loaded by the C compiler *cc*(1), the Pascal compiler *pc*(1), and the Fortran compiler *f77*(1). The link editor *ld*(1) searches this library under the '−lc' option. Declarations for some of these functions may be obtained from include files indicated on the appropriate pages.

(3F) The 3F functions are all functions callable from FORTRAN. These functions perform the same jobs as do the straight "3" functions.

(3M) These functions constitute the math library, *libm*. They are automatically loaded as needed by the Pascal compiler *pc*(1) and the Fortran compiler *f77*(1). The link editor searches this library under the '−lm' option. Declarations for these functions may be obtained from the include file <*math.h*>.

(3N) These functions constitute the internet network library,

(3S) These functions constitute the 'standard I/O package', see *intro*(3S). These functions are in the library *libc* already mentioned. Declarations for these functions may be obtained from the include file <*stdio.h*>.

(3X) Various specialized libraries have not been given distinctive captions. Files in which such libraries are found are named on appropriate pages.

(3C) Routines included for compatibility with other systems. In particular, a number of system call interfaces provided in previous releases of 4BSD have been included for source code compatibility. The manual page entry for each compatibility routine indicates the proper interface to use.

FILES

/lib/libc.a
/usr/lib/libm.a
/usr/lib/libc_p.a
/usr/lib/libm_p.a

SEE ALSO

intro(3C), intro(3S), intro(3F), intro(3M), intro(3N), nm(1), ld(1), cc(1), f77(1), intro(2)

DIAGNOSTICS

Functions in the math library (3M) may return conventional values 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 (domain error) or ERANGE (range error). The values of EDOM and ERANGE are defined in the include file <*math.h*>.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
abort	abort.3	generate a fault
abort	abort.3f	terminate abruptly with memory image

abs	abs.3	integer absolute value
access	access.3f	determine accessability of a file
acos	sin.3m	trigonometric functions
alarm	alarm.3c	schedule signal after specified time
alarm	alarm.3f	execute a subroutine after a specified time
alloca	malloc.3	memory allocator
arc	plot.3x	graphics interface
asctime	ctime.3	convert date and time to ASCII
asin	sin.3m	trigonometric functions
assert	assert.3x	program verification
atan	sin.3m	trigonometric functions
atan2	sin.3m	trigonometric functions
atof	atof.3	convert ASCII to numbers
atoi	atof.3	convert ASCII to numbers
atol	atof.3	convert ASCII to numbers
bcmp	bstring.3	bit and byte string operations
bcopy	bstring.3	bit and byte string operations
bessel	bessel.3f	of two kinds for integer orders
bit	bit.3f	and, or, xor, not, rshift, lshift bitwise functions
bzero	bstring.3	bit and byte string operations
cabs	hypot.3m	Euclidean distance
calloc	malloc.3	memory allocator
ceil	floor.3m	absolute value, floor, ceiling functions
chdir	chdir.3f	change default directory
chmod	chmod.3f	change mode of a file
circle	plot.3x	graphics interface
clearerr	ferror.3s	stream status inquiries
closedir	directory.3	directory operations
closelog	syslog.3	control system log
closepl	plot.3x	graphics interface
cont	plot.3x	graphics interface
cos	sin.3m	trigonometric functions
cosh	sinh.3m	hyperbolic functions
crypt	crypt.3	DES encryption
ctime	ctime.3	convert date and time to ASCII
ctime	time.3f	return system time
curses	curses.3x	screen functions with "optimal" cursor motion
dbminit	dbm.3x	data base subroutines
delete	dbm.3x	data base subroutines
dffrac	flmin.3f	return extreme values
dflmax	flmin.3f	return extreme values
dflmax	range.3f	return extreme values
dflmin	flmin.3f	return extreme values
dflmin	range.3f	return extreme values
drand	rand.3f	return random values
dtime	etime.3f	return elapsed execution time
ecvt	ecvt.3	output conversion
edata	end.3	last locations in program
encrypt	crypt.3	DES encryption
end	end.3	last locations in program
endfsent	getfsent.3x	get file system descriptor file entry
endgrent	getgrent.3	get group file entry

endhostent	gethostent.3n	get network host entry
endnetent	getnetent.3n	get network entry
endprotoent	getprotoent.3n	get protocol entry
endpwent	getpwent.3	get password file entry
endservent	getservent.3n	get service entry
environ	execl.3	execute a file
erase	plot.3x	graphics interface
etext	end.3	last locations in program
etime	etime.3f	return elapsed execution time
exec	execl.3	execute a file
exece	execl.3	execute a file
execl	execl.3	execute a file
execle	execl.3	execute a file
execlp	execl.3	execute a file
exect	execl.3	execute a file
execv	execl.3	execute a file
execvp	execl.3	execute a file
exit	exit.3	terminate a process after flushing any pending output
exit	exit.3f	terminate process with status
exp	exp.3m	exponential, logarithm, power, square root
fabs	floor.3m	absolute value, floor, ceiling functions
fclose	fclose.3s	close or flush a stream
fcvt	ecvt.3	output conversion
fdate	fdate.3f	return date and time in an ASCII string
feof	ferror.3s	stream status inquiries
ferror	ferror.3s	stream status inquiries
fetch	dbm.3x	data base subroutines
fflush	fclose.3s	close or flush a stream
ffrac	flmin.3f	return extreme values
ffs	bstring.3	bit and byte string operations
fgetc	getc.3f	get a character from a logical unit
fgetc	getc.3s	get character or word from stream
fgets	gets.3s	get a string from a stream
fileno	ferror.3s	stream status inquiries
firstkey	dbm.3x	data base subroutines
flmax	flmin.3f	return extreme values
flmax	range.3f	return extreme values
flmin	flmin.3f	return extreme values
flmin	range.3f	return extreme values
floor	floor.3m	absolute value, floor, ceiling functions
flush	flush.3f	flush output to a logical unit
fork	fork.3f	create a copy of this process
fpecnt	trpfpe.3f	trap and repair floating point faults
fprintf	printf.3s	formatted output conversion
fputc	putc.3f	write a character to a fortran logical unit
fputc	putc.3s	put character or word on a stream
fputs	puts.3s	put a string on a stream
fread	fread.3s	buffered binary input/output
free	malloc.3	memory allocator
frexp	frexp.3	split into mantissa and exponent
fscanf	scanf.3s	formatted input conversion
fseek	fseek.3f	reposition a file on a logical unit

fseek	fseek.3s	reposition a stream
fstat	stat.3f	get file status
ftell	fseek.3f	reposition a file on a logical unit
ftell	fseek.3s	reposition a stream
ftime	time.3c	get date and time
fwrite	fread.3s	buffered binary input/output
gamma	gamma.3m	log gamma function
gcvt	ecvt.3	output conversion
gerror	perror.3f	get system error messages
getarg	getarg.3f	return command line arguments
getc	getc.3f	get a character from a logical unit
getc	getc.3s	get character or word from stream
getchar	getc.3s	get character or word from stream
getcwd	getcwd.3f	get pathname of current working directory
getdiskbyname	getdisk.3x	get disk description by its name
getenv	getenv.3	value for environment name
getenv	getenv.3f	get value of environment variables
getfsent	getfsent.3x	get file system descriptor file entry
getfsfile	getfsent.3x	get file system descriptor file entry
getfsspec	getfsent.3x	get file system descriptor file entry
getfstype	getfsent.3x	get file system descriptor file entry
getgid	getuid.3f	get user or group ID of the caller
getgrent	getgrent.3	get group file entry
getgrgid	getgrent.3	get group file entry
getgrnam	getgrent.3	get group file entry
gethostbyaddr	gethostent.3n	get network host entry
gethostbyname	gethostent.3n	get network host entry
gethostent	gethostent.3n	get network host entry
getlog	getlog.3f	get user's login name
getlogin	getlogin.3	get login name
getnetbyaddr	getnetent.3n	get network entry
getnetbyname	getnetent.3n	get network entry
getnetent	getnetent.3n	get network entry
getpass	getpass.3	read a password
getpid	getpid.3f	get process id
getprotobyname	getprotoent.3n	get protocol entry
getprotobynumber	getprotoent.3n	get protocol entry
getprotoent	getprotoent.3n	get protocol entry
getpw	getpw.3	get name from uid
getpwent	getpwent.3	get password file entry
getpwnam	getpwent.3	get password file entry
getpwuid	getpwent.3	get password file entry
gets	gets.3s	get a string from a stream
getservbyname	getservent.3n	get service entry
getservbyport	getservent.3n	get service entry
getservent	getservent.3n	get service entry
getuid	getuid.3f	get user or group ID of the caller
getw	getc.3s	get character or word from stream
getwd	getwd.3	get current working directory pathname
gmtime	ctime.3	convert date and time to ASCII
gmtime	time.3f	return system time
gtty	stty.3c	set and get terminal state (defunct)

hostnm	hostnm.3f	get name of current host
htonl	byteorder.3n	convert values between host and network byte order
htons	byteorder.3n	convert values between host and network byte order
hypot	hypot.3m	Euclidean distance
iargc	getarg.3f	return command line arguments
idate	idate.3f	return date or time in numerical form
ierrno	perror.3f	get system error messages
index	index.3f	tell about character objects
index	string.3	string operations
inet_addr	inet.3n	Internet address manipulation routines
inet_lnaof	inet.3n	Internet address manipulation routines
inet_makeaddr	inet.3n	Internet address manipulation routines
inet_netof	inet.3n	Internet address manipulation routines
inet_network	inet.3n	Internet address manipulation routines
initgroups	initgroups.3x	initialize group access list
initstate	random.3	better random number generator
inmax	flmin.3f	return extreme values
inmax	range.3f	return extreme values
insque	insque.3	insert/remove element from a queue
ioinit	ioinit.3f	change f77 I/O initialization
irand	rand.3f	return random values
isalnum	ctype.3	character classification macros
isalpha	ctype.3	character classification macros
isascii	ctype.3	character classification macros
isatty	ttynam.3f	find name of a terminal port
isatty	ttyname.3	find name of a terminal
iscntrl	ctype.3	character classification macros
isdigit	ctype.3	character classification macros
islower	ctype.3	character classification macros
isprint	ctype.3	character classification macros
ispunct	ctype.3	character classification macros
isspace	ctype.3	character classification macros
isupper	ctype.3	character classification macros
itime	idate.3f	return date or time in numerical form
j0	j0.3m	bessel functions
j1	j0.3m	bessel functions
jn	j0.3m	bessel functions
kill	kill.3f	send a signal to a process
label	plot.3x	graphics interface
ldexp	frexp.3	split into mantissa and exponent
len	index.3f	tell about character objects
lib2648	lib2648.3x	subroutines for the HP 2648 graphics terminal
line	plot.3x	graphics interface
linemod	plot.3x	graphics interface
link	link.3f	make a link to an existing file
lnblnk	index.3f	tell about character objects
loc	loc.3f	return the address of an object
localtime	ctime.3	convert date and time to ASCII
log	exp.3m	exponential, logarithm, power, square root
log10	exp.3m	exponential, logarithm, power, square root
long	long.3f	integer object conversion
longjmp	setjmp.3	non-local goto

lstat	stat.3f	get file status
ltime	time.3f	return system time
malloc	malloc.3	memory allocator
mktemp	mktemp.3	make a unique file name
modf	frexp.3	split into mantissa and exponent
moncontrol	monitor.3	prepare execution profile
monitor	monitor.3	prepare execution profile
monstartup	monitor.3	prepare execution profile
move	plot.3x	graphics interface
nextkey	dbm.3x	data base subroutines
nice	nice.3c	set program priority
nlist	nlist.3	get entries from name list
ntohl	byteorder.3n	convert values between host and network byte order
ntohs	byteorder.3n	convert values between host and network byte order
opendir	directory.3	directory operations
openlog	syslog.3	control system log
pause	pause.3c	stop until signal
pclose	popen.3	initiate I/O to/from a process
perror	perror.3	system error messages
perror	perror.3f	get system error messages
plot: openpl	plot.3x	graphics interface
point	plot.3x	graphics interface
popen	popen.3	initiate I/O to/from a process
pow	exp.3m	exponential, logarithm, power, square root
printf	printf.3s	formatted output conversion
psignal	psignal.3	system signal messages
putc	putc.3f	write a character to a fortran logical unit
putc	putc.3s	put character or word on a stream
putchar	putc.3s	put character or word on a stream
puts	puts.3s	put a string on a stream
putw	putc.3s	put character or word on a stream
qsort	qsort.3	quicker sort
qsort	qsort.3f	quick sort
rand	rand.3c	random number generator
rand	rand.3f	return random values
random	random.3	better random number generator
rcmd	rcmd.3x	routines for returning a stream to a remote command
re_comp	regex.3	regular expression handler
re_exec	regex.3	regular expression handler
readdir	directory.3	directory operations
realloc	malloc.3	memory allocator
remque	insque.3	insert/remove element from a queue
rename	rename.3f	rename a file
rewind	fseek.3s	reposition a stream
rewinddir	directory.3	directory operations
rexec	rexec.3x	return stream to a remote command
rindex	index.3f	tell about character objects
rindex	string.3	string operations
rresvport	rcmd.3x	routines for returning a stream to a remote command
ruserok	rcmd.3x	routines for returning a stream to a remote command
scandir	scandir.3	scan a directory
scanf	scanf.3s	formatted input conversion

seekdir	directory.3	directory operations
setbuf	setbuf.3s	assign buffering to a stream
setbuffer	setbuf.3s	assign buffering to a stream
setegid	setuid.3	set user and group ID
seteuid	setuid.3	set user and group ID
setfsent	getfsent.3x	get file system descriptor file entry
setgid	setuid.3	set user and group ID
setgrent	getgrent.3	get group file entry
sethostent	gethostent.3n	get network host entry
setjmp	setjmp.3	non-local goto
setkey	crypt.3	DES encryption
setlinebuf	setbuf.3s	assign buffering to a stream
setnetent	getnetent.3n	get network entry
setprotoent	getprotoent.3n	get protocol entry
setpwent	getpwent.3	get password file entry
setrgid	setuid.3	set user and group ID
setruid	setuid.3	set user and group ID
setservent	getservent.3n	get service entry
setstate	random.3	better random number generator
setuid	setuid.3	set user and group ID
short	long.3f	integer object conversion
signal	signal.3	simplified software signal facilities
signal	signal.3f	change the action for a signal
sin	sin.3m	trigonometric functions
sinh	sinh.3m	hyperbolic functions
~~sleep~~	~~sleep.3~~	~~suspend~~ execution for interval
sleep	sleep.3f	suspend execution for an interval
space	plot.3x	graphics interface
sprintf	printf.3s	formatted output conversion
sqrt	exp.3m	exponential, logarithm, power, square root
srand	rand.3c	random number generator
srandom	random.3	better random number generator
sscanf	scanf.3s	formatted input conversion
stat	stat.3f	get file status
stdio	intro.3s	standard buffered input/output package
store	dbm.3x	data base subroutines
strcat	string.3	string operations
strcmp	string.3	string operations
strcpy	string.3	string operations
strlen	string.3	string operations
strncat	string.3	string operations
strncmp	string.3	string operations
strncpy	string.3	string operations
stty	stty.3c	set and get terminal state (defunct)
swab	swab.3	swap byt
sys_errlist	perror.3	system e
sys_nerr	perror.3	system e
sys_siglist	psignal.3	system s
syslog	syslog.3	control
system	system.3	issue a
system	system.3f	execute
tan	sin.3m	trigonor

tanh	sinh.3m	hyperbolic functions
tclose	topen.3f	f77 tape I/O
telldir	directory.3	directory operations
tgetent	termcap.3x	terminal independent operation routines
tgetflag	termcap.3x	terminal independent operation routines
tgetnum	termcap.3x	terminal independent operation routines
tgetstr	termcap.3x	terminal independent operation routines
tgoto	termcap.3x	terminal independent operation routines
time	time.3c	get date and time
time	time.3f	return system time
times	times.3c	get process times
timezone	ctime.3	convert date and time to ASCII
topen	topen.3f	f77 tape I/O
tputs	termcap.3x	terminal independent operation routines
traper	traper.3f	trap arithmetic errors
trapov	trapov.3f	trap and repair floating point overflow
tread	topen.3f	f77 tape I/O
trewin	topen.3f	f77 tape I/O
trpfpe	trpfpe.3f	trap and repair floating point faults
tskipf	topen.3f	f77 tape I/O
tstate	topen.3f	f77 tape I/O
ttynam	ttynam.3f	find name of a terminal port
ttyname	ttyname.3	find name of a terminal
ttyslot	ttyname.3	find name of a terminal
twrite	topen.3f	f77 tape I/O
ungetc	ungetc.3s	push character back into input stream
unlink	unlink.3f	remove a directory entry
utime	utime.3c	set file times
valloc	valloc.3	aligned memory allocator
varargs	varargs.3	variable argument list
vlimit	vlimit.3c	control maximum system resource consumption
vtimes	vtimes.3c	get information about resource utilization
wait	wait.3f	wait for a process to terminate
y0	j0.3m	bessel functions
y1	j0.3m	bessel functions
yn	j0.3m	bessel functions

NAME

abort − generate a fault

DESCRIPTION

Abort executes an instruction which is illegal in user mode. This causes a signal that normally terminates the process with a core dump, which may be used for debugging.

SEE ALSO

adb(1), sigvec(2), exit(2)

DIAGNOSTICS

Usually 'IOT trap − core dumped' from the shell.

BUGS

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

NAME

abs — integer absolute value

SYNOPSIS

abs(i)
int i;

DESCRIPTION

Abs returns the absolute value of its integer operand.

SEE ALSO

floor(3M) for *fabs*

BUGS

Applying the *abs* function to the most negative integer generates a result which is the most negative integer. That is,

abs(0x80000000)

returns 0x80000000 as a result.

NAME

atof, atoi, atol − convert ASCII to numbers

SYNOPSIS

double atof(nptr)
char *nptr;

atoi(nptr)
char *nptr;

long atol(nptr)
char *nptr;

DESCRIPTION

These functions convert a string pointed to by *nptr* to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

Atof recognizes an optional string of spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional 'e' or 'E' followed by an optionally signed integer.

Atoi and *atol* recognize an optional string of spaces, then an optional sign, then a string of digits.

SEE ALSO

scanf(3S)

BUGS

There are no provisions for overflow.

NAME

bcopy, bcmp, bzero, ffs − bit and byte string operations

SYNOPSIS

```
bcopy(b1, b2, length)
char *b1, *b2;
int length;

bcmp(b1, b2, length)
char *b1, *b2;
int length;

bzero(b, length)
char *b;
int length;

ffs(i)
int i;
```

DESCRIPTION

The functions *bcopy*, *bcmp*, and *bzero* operate on variable length strings of bytes. They do not check for null bytes as the routines in *string*(3) do.

Bcopy copies *length* bytes from string *b1* to the string *b2*.

Bcmp compares byte string *b1* against byte string *b2*, returning zero if they are identical, non-zero otherwise. Both strings are assumed to be *length* bytes long.

Bzero places *length* 0 bytes in the string *b1*.

Ffs find the first bit set in the argument passed it and returns the index of that bit. Bits are numbered starting at 1. A return value of −1 indicates the value passed is zero.

BUGS

The *bcmp* and *bcopy* routines take parameters backwards from *strcmp* and *strcpy*.

NAME

crypt, setkey, encrypt — DES encryption

SYNOPSIS

char *crypt(key, salt)
char *key, *salt;

setkey(key)
char *key;

encrypt(block, edflag)
char *block;

DESCRIPTION

Crypt is the password encryption routine. It is based on the NBS Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to *crypt* is normally a user's typed password. The second is a 2-character string chosen from the set [a-zA-Z0-9./]. The *salt* string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The other entries provide (rather primitive) access to the actual DES algorithm. The argument of *setkey* is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine.

The argument to the *encrypt* entry is likewise a character array of length 64 containing 0's and 1's. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by *setkey*. If *edflag* is 0, the argument is encrypted; if non-zero, it is decrypted.

SEE ALSO

passwd(1), passwd(5), login(1), getpass(3)

BUGS

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

NAME

ctime, localtime, gmtime, asctime, timezone – convert date and time to ASCII

SYNOPSIS

```
char *ctime(clock)
long *clock;

#include <sys/time.h>

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *timezone(zone, dst)
```

DESCRIPTION

Ctime converts a time pointed to by *clock* such as returned by *time*(3) into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
Sun Sep 16 01:03:52 1973\n\0
```

Localtime and *gmtime* return pointers to structures containing the broken-down time. *Localtime* corrects for the time zone and possible daylight savings time; *gmtime* converts directly to GMT, which is the time UNIX uses. *Asctime* converts a broken-down time to ASCII and returns a pointer to a 26-character string.

The structure declaration from the include file is:

```
struct tm {
        int     tm_sec;
        int     tm_min;
        int     tm_hour;
        int     tm_mday;
        int     tm_mon;
        int     tm_year;
        int     tm_wday;
        int     tm_yday;
        int     tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (0-11), day of week (Sunday = 0), year – 1900, day of year (0-365), and a flag that is nonzero if daylight saving time is in effect.

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

Timezone returns the name of the time zone associated with its first argument, which is measured in minutes westward from Greenwich. If the second argument is 0, the standard name is used, otherwise the Daylight Saving version. If the required name does not appear in a table built into the routine, the difference from GMT is produced; e.g. in Afghanistan *timezone(-(60*4+30), 0)* is appropriate because it is 4:30 ahead of GMT and the string **GMT+4:30** is produced.

SEE ALSO

gettimeofday(2), time(3)

BUGS

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

NAME

isalpha, isupper, islower, isdigit, isalnum, isspace, ispunct, isprint, iscntrl, isascii — character classification macros

SYNOPSIS

#include <ctype.h>

isalpha(c)

. . .

DESCRIPTION

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *Isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value EOF (see *stdio*(3S)).

isalpha	*c* is a letter
isupper	*c* is an upper case letter
islower	*c* is a lower case letter
isdigit	*c* is a digit
isalnum	*c* is an alphanumeric character
isspace	*c* is a space, tab, carriage return, newline, or formfeed
ispunct	*c* is a punctuation character (neither control nor alphanumeric)
isprint	*c* is a printing character, code 040(8) (space) through 0176 (tilde)
iscntrl	*c* is a delete character (0177) or ordinary control character (less than 040).
isascii	*c* is an ASCII character, code less than 0200

SEE ALSO

ascii(7)

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(dirp, loc)
DIR *dirp;
long loc;

rewinddir(dirp)
DIR *dirp;

closedir(dirp)
DIR *dirp;
```

DESCRIPTION

Opendir opens the directory named by *filename* and associates a *directory stream* with it. *Opendir* returns a pointer to be used to identify the *directory stream* in subsequent operations. The pointer NULL is returned if *filename* cannot be accessed, or if it cannot *malloc*(3) enough memory to hold the whole thing.

Readdir returns a pointer to the next directory entry. It returns NULL upon reaching the end of the directory or detecting an invalid *seekdir* operation.

Telldir returns the current location associated with the named *directory stream.*

Seekdir sets the position of the next *readdir* operation on the *directory stream.* The new position reverts to the one associated with the *directory stream* when the *telldir* operation was performed. Values returned by *telldir* are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the *telldir* value may be invalidated due to undetected directory compaction. It is safe to use a previous *telldir* value immediately after a call to *opendir* and before any calls to *readdir.*

Rewinddir resets the position of the named *directory stream* to the beginning of the directory.

Closedir closes the named *directory stream* and frees the structure associated with the DIR pointer.

Sample code which 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

open(2), close(2), read(2), lseek(2), dir(5)

NAME

ecvt, fcvt, gcvt — output conversion

SYNOPSIS

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

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

char *gcvt(value, ndigit, buf)
double value;
char *buf;

DESCRIPTION

Ecvt converts the *value* to a null-terminated string of *ndigit* ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt* (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero. The low-order digit is rounded.

Fcvt is identical to *ecvt*, except that the correct digit has been rounded for Fortran F-format output of the number of digits specified by *ndigits*.

Gcvt converts the *value* to a null-terminated ASCII string in *buf* and returns a pointer to *buf*. It attempts to produce *ndigit* significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

SEE ALSO

printf(3)

BUGS

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

NAME

end, etext, edata − last locations in program

SYNOPSIS

extern end;
extern etext;
extern edata;

DESCRIPTION

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

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

SEE ALSO

brk(2), malloc(3)

NAME

(except) raise, raise_sys() – C exception handling

SYNOPSIS

#include <except.h>

raise(code, msg)
int code;
char *msg;

raise_sys()

cc ... –lexcept

EXTENDED C SYNTAX

DURING statement1 **HANDLER** statement2 **END_HANDLER**

E_RETURN(expression **)**

E_RETURN_VOID

DESCRIPTION

The macros and functions in this package provide a limited amount of exception handling for programming in C. They provide the ability to associate an exception handler to be invoked if an exception is raised during the execution of a statement.

C syntax is extended by several macros that allow the programmer to associate an exception handler with a statement. The "syntax" for this is:

DURING statement1 HANDLER statement2 END_HANDLER

Either or both statement may be a compound statement. If an exception is raised using the *raise*() function during *statement1* (or during any functions called by *statement1*), the stack will be unwound and *statement2* will be invoked in the current context. However, if the exception handler is redeclared in a *dynamically* enclosed statement, the current exception handler will be inactive during the execution of the enclosed statement.

During the execution of *statement2*, two predefined values may be used: *Exception.Code*, an integer, is the value of *code* passed to the *raise*() call which invoked the handler, and *Exception.Message* is the value of *msg*. It is up to the user to define the values used for the exception codes; by convention, small positive integers are interpreted as Unix error codes.

As an example of the use of this package, the following "toy" code computes the quotient of variables f1 and f2, unless f2 is 0.0:

```
DURING {
   if (f2 == 0.0)
        raise(DIVIDE_BY_ZERO, "Division by zero attempted");
   quotient = f1/ f2;
} HANDLER
   switch (Exception.Code) {
   case DIVIDE_BY_ZERO:
        return(HUGE);
        break;
   default:
        printf("Unexpected error %s\n", Exception.Message);
```

```
    }
  END_HANDLER
```

If a handler does not want to take responsibility for an exception, it can "pass the buck" to the dynamically enclosing exception handler by use of the *RERAISE* macro, which simply raises the exception that invoked the handler. Of course, it is possible that there is no higher-level handler. The programmer can control the action in this case by setting the external int *ExceptMode* to some (bit-wise OR'd) combination of the following constants:

EX_MODE_REPORT	Print a message on stderr if an exception is not caught. If this is not set, no message is printed.
EX_MODE_ABORT	Calls the *abort*(3) routine if an exception is not caught. If this is not set, *exit*(3) is called, with the exception code as an argument.

The default value for *ExceptMode* is zero.

RESTRICTIONS

THESE RESTRICTIONS ARE IMPORTANT; YOU WILL SUFFER IF YOU DISOBEY THEM.

During the execution of *statement1*, no transfers out of the statement are allowed, except as noted here. Execution of a compound *statement1* must "run off the end" of the block. This means that *statement1* may not include a **return** or **goto**, or a **break** or **continue** that would affect a loop enclosing the *DURING ... END_HANDLER* block. The *statement1* may include a call to *raise*() (but not *RERAISE*), *exit*(3), and any statement at all may be used in a function called.

If you wish to use a **return** within *statement1*, you must instead use *E_RETURN()* to return a value, or *E_RETURN_VOID* if the enclosing function is declared **void**. These two macros may be used *only* in the (lexically) outermost *statement1* of a function, and nowhere else.

There are no restrictions on what may be done inside the *statement2* part of a handler block, except that it is subject to the above constraints if it is lexically enclosed in the *statement1* part of another handler.

As an aid to Unix programmers, the *raise_sys*() function is provided. It is used exactly as *raise*() is, except that it uses the global *errno*(3) to produce the exception code and message.

SEE ALSO

errno(3), setjmp(3)

AUTHOR

Jeffrey Mogul (Stanford)

BUGS

Due to a limitation of the *setjmp*(3) implementation, **register** variables which are actually stored in registers (and this is not always easy to determine, and especially is not portable) are restored to the values they had upon entering *statement1* when the handler (*statement2*) is invoked. All other data keeps whatever values they were assigned during the (interrupted) execution of *statement1*. A good rule to follow is that you should not rely on the values of variables declared **register** (in the current block) after an exception has been caught.

NAME

execl, execv, execle, execlp, execvp, exec, exece, exect, environ — execute a file

SYNOPSIS

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

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

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

exect(name, argv, envp)
char *name, *argv[], *envp[];

extern char **environ;

DESCRIPTION

These routines provide various interfaces to the *execve* system call. Refer to *execve*(2) for a description of their properties; only brief descriptions are provided here.

Exec in all its forms overlays the calling process with the named file, then transfers to the entry point of the core image of the file. There can be no return from a successful exec; the calling core image is lost.

The *name* argument is a pointer to the name of the file to be executed. The pointers *arg*[*0*], *arg*[*1*] ... address null-terminated strings. Conventionally *arg*[*0*] is the name of the file.

Two interfaces are available. *execl* is useful when a known file with known arguments is being called; the arguments to *execl* are the character strings constituting the file and the arguments; the first argument is conventionally the same as the file name (or its last component). A 0 argument must end the argument list.

The *execv* version is useful when the number of arguments is unknown in advance; the arguments to *execv* are the name of the file to be executed and a vector of strings containing the arguments. The last argument string must be followed by a 0 pointer.

The *exect* version is used when the executed file is to be manipulated with *ptrace*(2). The program is forced to single step a single instruction giving the parent an opportunity to manipulate its state. On the VAX-11 this is done by setting the trace bit in the process status longword.

When a C program is executed, it is called as follows:

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

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. As indicated, *argc* is conventionally at least one and the first member of the array points to a string containing the name of the file.

Argv is directly usable in another *execv* because *argv*[*argc*] is 0.

Envp is a pointer to an array of strings that constitute the *environment* of the process. Each string consists of a name, an "=", and a null-terminated value. The array of pointers is terminated by a null pointer. The shell *sh*(1) passes an environment entry for each global shell variable defined when the program is called. See *environ*(7) for some conventionally used names. The C run-time start-off routine places a copy of *envp* in the global cell *environ*, which is used by *execv* and *execl* to pass the environment to any subprograms executed by the current program.

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.

FILES

/bin/sh shell, invoked if command file found by *execlp* or *execvp*

SEE ALSO

execve(2), fork(2), environ(7), csh(1)

DIAGNOSTICS

If the file cannot be found, if it is not executable, if it does not start with a valid magic number (see *a.out*(5)), if maximum memory is exceeded, or if the arguments require too much space, a return constitutes the diagnostic; the return value is −1. Even for the super-user, at least one of the execute-permission bits must be set for a file to be executed.

BUGS

If *execvp* is called to execute a file that turns out to be a shell command file, and if it is impossible to execute the shell, the values of *argv[0]* and *argv[−1]* will be modified before return.

NAME
exit — terminate a process after flushing any pending output

SYNOPSIS
exit(status)
int status;

DESCRIPTION
Exit terminates a process after calling the Standard I/O library function *_cleanup* to flush any buffered output. *Exit* never returns.

SEE ALSO
exit(2), intro(3S)

NAME

frexp, ldexp, modf − split into mantissa and exponent

SYNOPSIS

double frexp(value, eptr)
double value;
int *eptr;

double ldexp(value, exp)
double value;

double modf(value, iptr)
double value, *iptr;

DESCRIPTION

Frexp returns the mantissa of a double *value* as a double quantity, *x,* of magnitude less than 1 and stores an integer *n* such that $value = x*2^n$ indirectly through *eptr.*

Ldexp returns the quantity $value*2^{exp}$.

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

NAME

getbanner – get system login banner string

SYNOPSIS

char *getbanner();
cc files ... –lgetty.

DESCRIPTION

Getbanner tries to extract the *im* (initial message) field from the *default* entry in /etc/gettytab (see *gettytab*(5)). It then decodes the initial message according to the *gettytab* specifications (including possible hostname substitution and alteration via the *hn* and *he* fields), and returns a pointer to the static string. If /etc/gettytab does not exist or the default entry can not be found, a default banner is substituted.

FILES

/usr/stanford/lib/libgetty.a
/etc/gettytab

SEE ALSO

gettytab(5)

AUTHOR

Bill Burgess

NAME

getenv − value for environment name

SYNOPSIS

char *getenv(name)
char *name;

DESCRIPTION

Getenv searches the environment list (see *environ*(7)) for a string of the form *name*=*value* and returns a pointer to the string *value* if such a string is present, otherwise *getenv* returns the value 0 (NULL).

SEE ALSO

environ(7), execve(2)

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent − get group file entry

SYNOPSIS

```
#include <grp.h>

struct group *getgrent()

struct group *getgrgid(gid)
int gid;

struct group *getgrnam(name)
char *name;

setgrent()

endgrent()
```

DESCRIPTION

Getgrent, getgrgid and *getgrnam* each return pointers to an object with the following structure containing the broken-out fields of a line in the group file.

```
struct  group { /* see getgrent(3) */
        char    *gr_name;
        char    *gr_passwd;
        int     gr_gid;
        char    **gr_mem;
};

struct group *getgrent(), *getgrgid(), *getgrnam();
```

The members of this structure are:

gr_name	The name of the group.
gr_passwd	The encrypted password of the group.
gr_gid	The numerical group-ID.
gr_mem	Null-terminated vector of pointers to the individual member names.

Getgrent simply reads the next line while *getgrgid* and *getgrnam* search until a matching *gid* or *name* is found (or until EOF is encountered). Each routine picks up where the others leave off so successive calls may be used to search the entire file.

A call to *setgrent* has the effect of rewinding the group file to allow repeated searches. *Endgrent* may be called to close the group file when processing is complete.

FILES

/etc/group

SEE ALSO

getlogin(3), getpwent(3), group(5)

DIAGNOSTICS

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

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

getlogin − get login name

SYNOPSIS

char *getlogin()

DESCRIPTION

Getlogin returns a pointer to the login name as found in */etc/utmp*. It may be used in conjunction with *getpwnam* to locate the correct password file entry when the same userid is shared by several login names.

If *getlogin* is called within a process that is not attached to a typewriter, it returns NULL. The correct procedure for determining the login name is to first call *getlogin* and if it fails, to call *getpw*(*getuid*()).

FILES

/etc/utmp

SEE ALSO

getpwent(3), getgrent(3), utmp(5), getpw(3)

DIAGNOSTICS

Returns NULL (0) if name not found.

BUGS

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

NAME

getpass − read a password

SYNOPSIS

char *getpass(prompt)
char *prompt;

DESCRIPTION

Getpass reads a password from the file */dev/tty*, or if that cannot be opened, from the standard input, after prompting with the null-terminated string *prompt* and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters.

FILES

/dev/tty

SEE ALSO

crypt(3)

BUGS

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

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent — get password file entry

SYNOPSIS

#include <pwd.h>

struct passwd *getpwent()

struct passwd *getpwuid(uid)
int uid;

struct passwd *getpwnam(name)
char *name;

int setpwent()

int endpwent()

DESCRIPTION

Getpwent, getpwuid and *getpwnam* each return a pointer to an object with the following structure containing the broken-out fields of a line in the password file.

```
struct  passwd { /* see getpwent(3) */
        char    *pw_name;
        char    *pw_passwd;
        int     pw_uid;
        int     pw_gid;
        int     pw_quota;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};

struct passwd *getpwent(), *getpwuid(), *getpwnam();
```

The fields *pw_quota* and *pw_comment* are unused; the others have meanings described in *passwd*(5).

Getpwent reads the next line (opening the file if necessary); *setpwent* rewinds the file; *endpwent* closes it.

Getpwuid and *getpwnam* search from the beginning until a matching *uid* or *name* is found (or until EOF is encountered).

FILES

/etc/passwd

SEE ALSO

getlogin(3), getgrent(3), passwd(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

getwd — get current working directory pathname

SYNOPSIS

char *getwd(pathname)
char *pathname;

DESCRIPTION

Getwd copies the absolute pathname of the current working directory to *pathname* and returns a pointer to the result.

LIMITATIONS

Maximum pathname length is MAXPATHLEN characters (1024).

DIAGNOSTICS

Getwd returns zero and places a message in *pathname* if an error occurs.

BUGS

Getwd may fail to return to the current directory if an error occurs.

NAME

insque, remque — insert/remove element from a queue

SYNOPSIS

```
struct qelem {
        struct  qelem *q_forw;
        struct  qelem *q_back;
        char    q_data[];
};

insque(elem, pred)
struct qelem *elem, *pred;

remque(elem)
struct qelem *elem;
```

DESCRIPTION

Insque and *remque* manipulate queues built from doubly linked lists. Each element in the queue must in the form of "struct qelem". *Insque* inserts *elem* in a queue imediately after *pred*; *remque* removes an entry *elem* from a queue.

SEE ALSO

"VAX Architecture Handbook", pp. 228-235.

NAME

malloc, free, realloc, calloc, alloca — memory allocator

SYNOPSIS

```
char *malloc(size)
unsigned size;

free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

char *alloca(size)
int size;
```

DESCRIPTION

Malloc and *free* provide a simple general-purpose memory allocation package. *Malloc* returns a pointer to a block of at least *size* bytes beginning on a word boundary.

The argument to *free* is a pointer to a block previously allocated by *malloc*; this space is made available for further allocation, but its contents are left undisturbed.

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

Malloc maintains multiple lists of free blocks according to size, allocating space from the appropriate list. It calls *sbrk* (see *brk*(2)) to get more memory from the system when there is no suitable space already free.

Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

In order to be compatible with older versions, *realloc* also works if *ptr* points to a block freed since the last call of *malloc, realloc* or *calloc*; sequences of *free, malloc* and *realloc* were previously used to attempt storage compaction. This procedure is no longer recommended.

Calloc allocates space for an array of *nelem* elements of size *elsize.* The space is initialized to zeros.

Alloca allocates *size* bytes of space in the stack frame of the caller. This temporary space is automatically freed on return.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

DIAGNOSTICS

Malloc, realloc and *calloc* return a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. *Malloc* may be recompiled to check the arena very stringently on every transaction; those sites with a source code license may check the source code to see how this can be done.

BUGS

When *realloc* returns 0, the block pointed to by *ptr* may be destroyed.

Alloca is machine dependent; it's use is discouraged.

NAME

mktemp — make a unique file name

SYNOPSIS

char *mktemp(template)
char *template;

DESCRIPTION

Mktemp replaces *template* by a unique file name, and returns the address of the template. The template should look like a file name with six trailing X's, which will be replaced with the current process id and a unique letter.

SEE ALSO

getpid(2)

NAME

monitor, monstartup, moncontrol — prepare execution profile

SYNOPSIS

monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc) (), (*highpc) ();
short buffer[];

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

moncontrol(mode)

DESCRIPTION

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

```
cc −p . . .
```

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

```
cc −pg . . .
```

automatically includes calls for the *gprof*(1) monitor.

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

To profile the entire program, it is sufficient to use

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

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

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

```
monitor(0);
```

then *prof*(1) can be used to examine the results.

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

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

To profile the entire program, it is sufficient to use

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

FILES

mon.out

SEE ALSO

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

NAME

nlist − get entries from name list

SYNOPSIS

#include <nlist.h>

nlist(filename, nl)
char *filename;
struct nlist nl[];

DESCRIPTION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of an array of structures containing names, types and values. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to 0. See *a.out*(5) for the structure declaration.

This subroutine is useful for examining the system name list kept in the file **/vmunix**. In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out(5)

DIAGNOSTICS

All type entries are set to 0 if the file cannot be found or if it is not a valid namelist.

NAME

perror, sys_errlist, sys_nerr − system error messages

SYNOPSIS

perror(s)
char *s;

int sys_nerr;
char *sys_errlist[];

DESCRIPTION

Perror produces a short error message on the standard error file describing the last error encountered during a call to the system from a C program. First the argument string *s* is printed, then a colon, then the message and a new-line. Most usefully, the argument string is the name of the program which incurred the error. The error number is taken from the external variable *errno* (see *intro*(2)), which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings *sys_errlist* is provided; *errno* can be used as an index in this table to get the message string without the newline. *Sys_nerr* is the number of messages provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO

intro(2), psignal(3)

NAME

psignal, sys_siglist — system signal messages

SYNOPSIS

```
psignal(sig, s)
unsigned sig;
char *s;

char *sys_siglist[];
```

DESCRIPTION

Psignal produces a short message on the standard error file describing the indicated signal. First the argument string *s* is printed, then a colon, then the name of the signal and a new-line. Most usefully, the argument string is the name of the program which incurred the signal. The signal number should be from among those found in <*signal.h*>.

To simplify variant formatting of signal names, the vector of message strings *sys_siglist* is provided; the signal number can be used as an index in this table to get the signal name without the newline. The define NSIG defined in <*signal.h*> is the number of messages provided for in the table; it should be checked because new signals may be added to the system before they are added to the table.

SEE ALSO

sigvec(2), perror(3)

NAME

qsort — quicker sort

SYNOPSIS

qsort(base, nel, width, compar)
char *base;
int (*compar)();

DESCRIPTION

Qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine to be called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

SEE ALSO

sort(1)

NAME

random, srandom, initstate, setstate — better random number generator; routines for changing generators

SYNOPSIS

long random()

srandom(seed)
int seed;

char *initstate(seed, state, n)
unsigned seed;
char *state;
int n;

char *setstate(state)
char *state;

DESCRIPTION

Random uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$. The period of this random number generator is very large, approximately $16*(2^{31}-1)$.

Random/srandom have (almost) the same calling sequence and initialization properties as *rand/srand.* The difference is that *rand*(3) produces a much less random sequence -- in fact, the low dozen bits generated by rand go through a cyclic pattern. All the bits generated by *random* are usable. For example, "random()&01" will produce a random binary value.

Unlike *srand*, *srandom* does not return the old seed; the reason for this is that the amount of state information used is much more than a single word. (Two other routines are provided to deal with restarting/changing random number generators). Like *rand*(3), however, *random* will by default produce a sequence of numbers that can be duplicated by calling *srandom* with *1* as the seed.

The *initstate* routine allows a state array, passed in as an argument, to be initialized for future use. The size of the state array (in bytes) is used by *initstate* to decide how sophisticated a random number generator it should use -- the more state, the better the random numbers will be. (Current "optimal" values for the amount of state information are 8, 32, 64, 128, and 256 bytes; other amounts will be rounded down to the nearest known amount. Using less than 8 bytes will cause an error). The seed for the initialization (which specifies a starting point for the random number sequence, and provides for restarting at the same point) is also an argument. *Initstate* returns a pointer to the previous state information array.

Once a state has been initialized, the *setstate* routine provides for rapid switching between states. *Setstate returns a pointer to the* argument state array is used for further random number generation until the next call to *initstate* or *setstate.*

Once a state array has been initialized, it may be restarted at a different point either by calling *initstate* (with the desired seed, the state array, and its size) or by calling both *setstate* (with the state array) and *srandom* (with the desired seed). The advantage of calling both *setstate* and *srandom* is that the size of the state array does not have to be remembered after it is initialized.

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

AUTHOR

Earl T. Cohen

DIAGNOSTICS

If *initstate* is called with less than 8 bytes of state information, or if *setstate* detects that the state information has been garbled, error messages are printed on the standard error output.

SEE ALSO

rand(3)

BUGS

About 2/3 the speed of *rand*(3C).

NAME

re_comp, re_exec — regular expression handler

SYNOPSIS

```
char *re_comp(s)
char *s;

re_exec(s)
char *s;
```

DESCRIPTION

Re_comp compiles a string into an internal form suitable for pattern matching. *Re_exec* checks the argument string against the last string passed to *re_comp.*

Re_comp returns 0 if the string *s* was compiled successfully; otherwise a string containing an error message is returned. If *re_comp* is passed 0 or a null string, it returns without changing the currently compiled regular expression.

Re_exec returns 1 if the string *s* matches the last compiled regular expression, 0 if the string *s* failed to match the last compiled regular expression, and −1 if the compiled regular expression was invalid (indicating an internal error).

The strings passed to both *re_comp* and *re_exec* may have trailing or embedded newline characters; they are terminated by nulls. The regular expressions recognized are described in the manual entry for *ed*(1), given the above difference.

SEE ALSO

ed(1), ex(1), egrep(1), fgrep(1), grep(1)

DIAGNOSTICS

Re_exec returns −1 for an internal error.

Re_comp returns one of the following strings if an error occurs:

No previous regular expression,
Regular expression too long,
unmatched \(,
missing],
too many \(\) pairs,
unmatched \).

NAME

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.

SEE ALSO

directory(3), malloc(3), qsort(3), dir(5)

DIAGNOSTICS

Returns −1 if the directory cannot be opened for reading or if *malloc*(3) cannot allocate enough memory to hold all the data structures.

NAME

setjmp, longjmp — non-local goto

SYNOPSIS

#include <setjmp.h>

setjmp(env)
jmp_buf env;

longjmp(env, val)
jmp_buf env;

_setjmp(env)
jmp_buf env;

_longjmp(env, val)
jmp_buf env;

DESCRIPTION

These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

Setjmp saves its stack environment in *env* for later use by *longjmp.* It returns value 0.

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

Setjmp and *longjmp* save and restore the signal mask *sigmask*(2), while *_setjmp* and *_longjmp* manipulate only the C stack and registers.

SEE ALSO

sigvec(2), sigstack(2), signal(3)

BUGS

Setjmp does not save current notion of whether the process is executing on the signal stack. The result is that a longjmp to some place on the signal stack leaves the signal stack state incorrect.

NAME

setuid, seteuid, setruid, setgid, setegid, setrgid — set user and group ID

SYNOPSIS

setuid (uid)
seteuid (euid)
setruid (ruid)

setgid (gid)
setegid (egid)
setrgid (rgid)

DESCRIPTION

Setuid (*setgid*) sets both the real and effective user ID (group ID) of the current process to as specified.

Seteuid (*setegid*) sets the effective user ID (group ID) of the current process.

Setruid (*setruid*) sets the real user ID (group ID) of the current process.

These calls are only permitted to the super-user or if the argument is the real or effective ID.

SEE ALSO

setreuid(2), setregid(2), getuid(2), getgid(2)

DIAGNOSTICS

Zero is returned if the user (group) ID is set; −1 is returned otherwise.

NAME

sleep — suspend execution for interval

SYNOPSIS

sleep(seconds)
unsigned seconds;

DESCRIPTION

The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an interval timer and pausing until it occurs. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous timer, the process sleeps only until the signal would have occurred, and the signal is sent 1 second later.

SEE ALSO

setitimer(2), sigpause(2)

BUGS

An interface with finer resolution is needed.

NAME

strcmpfold, strncmpfold – case-folded string comparison operations

SYNOPSIS

strcmpfold(s1, s2)
char *s1, *s2;

strncmpfold(s1, s2, n)
char *s1, *s2;

DESCRIPTION

These functions operate on null-terminated strings. They ignore case in comparisons; e.g., "CAT" and "Cat" compare equal, and "cat" collates before "DOG". Otherwise, they are the same as the *strcmp* and *strncmp* functions described in *string*(3).

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

SEE ALSO

string(3)

BUGS

The name of *strncmpfold* is not unique in its first seven characters, and thus cannot be ported to some implementations of C.

These functions are currently unique to Stanford, and so programs using them may not be portable.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index, rindex — string operations

SYNOPSIS

```
#include <strings.h>

char *strcat(s1, s2)
char *s1, *s2;

char *strncat(s1, s2, n)
char *s1, *s2;

strcmp(s1, s2)
char *s1, *s2;

strncmp(s1, s2, n)
char *s1, *s2;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;

strlen(s)
char *s;

char *index(s, c)
char *s, c;

char *rindex(s, c)
char *s, c;
```

DESCRIPTION

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

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

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

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

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

Index (*rindex*) returns a pointer to the first (last) occurrence of character *c* in string *s*, or zero if *c* does not occur in the string.

NAME

swab − swap bytes

SYNOPSIS

swab(from, to, nbytes)
char *from, *to;

DESCRIPTION

Swab copies *nbytes* bytes pointed to by *from* to the position pointed to by *to,* exchanging adjacent even and odd bytes. It is useful for carrying binary data between PDP11's and other machines. *Nbytes* should be even.

NAME

syslog, openlog, closelog -- control system log

SYNOPSIS

```
# include <syslog.h>

openlog(ident, logstat)
char *ident;

syslog(priority, message, parameters ... )
char *message;

closelog()
```

DESCRIPTION

Syslog arranges to write the *message* onto the system log maintained by *syslog*(8). The message is tagged with *priority.* The message looks like a *printf(3)* string except that **%m** is replaced by the current error message (collected from *errno*). A trailing newline is added if needed. This message will be read by *syslog(8)* and output to the system console or files as appropriate.

If special processing is needed, *openlog* can be called to initialize the log file. Parameters are *ident* which is prepended to every message, and *logstat* which is a bit field indicating special status; current values are:

LOG_PID	log the process id with each message: useful for identifying instantiations of daemons.

Openlog returns zero on success. If it cannot open the file */dev/log,* it writes on */dev/console* instead and returns -1.

Closelog can be used to close the log file.

EXAMPLES

```
syslog(LOG_SALERT, "who: internal error 23");

openlog("serverftp", LOG_PID);
syslog(LOG_INFO, "Connection from host %d", CallingHost);
```

SEE ALSO

syslog(8)

NAME

system − issue a shell command

SYNOPSIS

system(string)
char *string;

DESCRIPTION

System causes the *string* to be given to *sh*(1) as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

SEE ALSO

popen(3S), execve(2), wait(2)

DIAGNOSTICS

Exit status 127 indicates the shell couldn't be executed.

NAME

ttyname, isatty, ttyslot — find name of a terminal

SYNOPSIS

char *ttyname (filedes)

isatty (filedes)

ttyslot ()

DESCRIPTION

Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor *filedes* (this is a system file descriptor and has nothing to do with the standard I/O FILE typedef).

Isatty returns 1 if *filedes* is associated with a terminal device, 0 otherwise.

Ttyslot returns the number of the entry in the *ttys*(5) file for the control terminal of the current process.

FILES

/dev/*
/etc/ttys

SEE ALSO

ioctl(2), ttys(5)

DIAGNOSTICS

Ttyname returns a null pointer (0) if *filedes* does not describe a terminal device in directory '/dev'.

Ttyslot returns 0 if '/etc/ttys' is inaccessible or if it cannot determine the control terminal.

BUGS

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

NAME

valloc — aligned memory allocator

SYNOPSIS

char *valloc(size)
unsigned size;

DESCRIPTION

Valloc allocates *size* bytes aligned on a page boundary. It is implemented by calling *malloc*(3) with a slightly larger request, saving the true beginning of the block allocated, and returning a properly aligned pointer.

DIAGNOSTICS

Valloc returns a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block.

BUGS

Vfree isn't implemented.

NAME

varargs — variable argument list

SYNOPSIS

#include <varargs.h>

function(**va_alist**)
va_dcl
va_list *pvar*;
va_start(*pvar*);
f = **va_arg**(*pvar*, *type*);
va_end(*pvar*);

DESCRIPTION

This set of macros provides a means of writing portable procedures that accept variable argument lists. Routines having variable argument lists (such as *printf*(3)) that do not use varargs are inherently nonportable, since different machines use different argument passing conventions.

va_alist is used in a function header to declare a variable argument list.

va_dcl is a declaration for **va_alist**. Note that there is no semicolon after **va_dcl.**

va_list is a type which can be used for the variable *pvar*, which is used to traverse the list. One such variable must always be declared.

va_start(pvar) is called to initialize *pvar* to the beginning of the list.

va_arg(*pvar*, *type*) will return the next argument in the list pointed to by *pvar*. *Type* is the type 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, since it cannot be determined at runtime.

va_end(*pvar*) is used to finish up.

Multiple traversals, each bracketed by **va_start** ... **va_end,** are possible.

EXAMPLE

```
#include <varargs.h>
execl(va_alist)
va_dcl
{
        va_list ap;
        char *file;
        char *args[100];
        int argno = 0;

        va_start(ap);
        file = va_arg(ap, char *);
        while (args[argno++] = va_arg(ap, char *))
                ;
        va_end(ap);
        return execv(file, args);
}
```

BUGS

It is up to the calling routine to determine how many arguments there are, since it is not possible to determine this from the stack frame. For example, *execl* passes a 0 to signal the end of the list. *Printf* can tell how many arguments are supposed to be there by the format.

NAME

intro − introduction to FORTRAN library functions

DESCRIPTION

This section describes those functions that are in the FORTRAN run time library. The functions listed here provide an interface from *f77* programs to the system in the same manner as the C library does for C programs. They are automatically loaded as needed by the Fortran compiler *f77*(1).

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

For efficiency, the SCCS ID strings are not normally included in the *a.out* file. To include them, simply declare

```
        external f77lid
```

in any *f77* module.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
abort	abort.3f	terminate abruptly with memory image
access	access.3f	determine accessability of a file
alarm	alarm.3f	execute a subroutine after a specified time
bessel	bessel.3f	of two kinds for integer orders
bit	bit.3f	and, or, xor, not, rshift, lshift bitwise functions
chdir	chdir.3f	change default directory
chmod	chmod.3f	change mode of a file
ctime	time.3f	return system time
dffrac	flmin.3f	return extreme values
dflmax	flmin.3f	return extreme values
dflmin	flmin.3f	return extreme values
drand	rand.3f	return random values
dtime	etime.3f	return elapsed execution time
etime	etime.3f	return elapsed execution time
exit	exit.3f	terminate process with status
fdate	fdate.3f	return date and time in an ASCII string
ffrac	flmin.3f	return extreme values
fgetc	getc.3f	get a character from a logical unit
flmax	flmin.3f	return extreme values
flmin	flmin.3f	return extreme values
flush	flush.3f	flush output to a logical unit
fork	fork.3f	create a copy of this process
fpecnt	trpfpe.3f	trap and repair floating point faults
fputc	putc.3f	write a character to a fortran logical unit
fseek	fseek.3f	reposition a file on a logical unit
fstat	stat.3f	get file status
ftell	fseek.3f	reposition a file on a logical unit
gerror	perror.3f	get system error messages
getarg	getarg.3f	return command line arguments
getc	getc.3f	get a character from a logical unit
getcwd	getcwd.3f	get pathname of current working directory
getenv	getenv.3f	get value of environment variables
getgid	getuid.3f	get user or group ID of the caller
getlog	getlog.3f	get user's login name

getpid	getpid.3f	get process id
getuid	getuid.3f	get user or group ID of the caller
gmtime	time.3f	return system time
hostnm	hostnm.3f	get name of current host
iargc	getarg.3f	return command line arguments
idate	idate.3f	return date or time in numerical form
ierrno	perror.3f	get system error messages
index	index.3f	tell about character objects
inmax	flmin.3f	return extreme values
intro	intro.3f	introduction to FORTRAN library functions
ioinit	ioinit.3f	change f77 I/O initialization
irand	rand.3f	return random values
isatty	ttynam.3f	find name of a terminal port
itime	idate.3f	return date or time in numerical form
kill	kill.3f	send a signal to a process
len	index.3f	tell about character objects
link	link.3f	make a link to an existing file
lnblnk	index.3f	tell about character objects
loc	loc.3f	return the address of an object
long	long.3f	integer object conversion
lstat	stat.3f	get file status
ltime	time.3f	return system time
perror	perror.3f	get system error messages
putc	putc.3f	write a character to a fortran logical unit
qsort	qsort.3f	quick sort
rand	rand.3f	return random values
rename	rename.3f	rename a file
rindex	index.3f	tell about character objects
short	long.3f	integer object conversion
signal	signal.3f	change the action for a signal
sleep	sleep.3f	suspend execution for an interval
stat	stat.3f	get file status
system	system.3f	execute a UNIX command
tclose	topen.3f	f77 tape I/O
time	time.3f	return system time
topen	topen.3f	f77 tape I/O
traper	traper.3f	trap arithmetic errors
trapov	trapov.3f	trap and repair floating point overflow
tread	topen.3f	f77 tape I/O
trewin	topen.3f	f77 tape I/O
trpfpe	trpfpe.3f	trap and repair floating point faults
tskipf	topen.3f	f77 tape I/O
tstate	topen.3f	f77 tape I/O
ttynam	ttynam.3f	find name of a terminal port
twrite	topen.3f	f77 tape I/O
unlink	unlink.3f	remove a directory entry
wait	wait.3f	wait for a process to terminate

NAME

abort – terminate abruptly with memory image

SYNOPSIS

subroutine abort (string)
character*(*) string

DESCRIPTION

Abort cleans up the I/O buffers and then aborts producing a *core* file in the current directory. If *string* is given, it is written to logical unit 0 preceeded by "abort:".

FILES

/usr/lib/libF77.a

SEE ALSO

abort(3)

BUGS

String is ignored on the PDP11.

NAME

access — determine accessability of a file

SYNOPSIS

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

DESCRIPTION

Access checks the given file, *name,* for accessability with respect to the caller according to *mode.* *Mode* may include in any order and in any combination one or more of:

r	test for read permission
w	test for write permission
x	test for execute permission
(blank)	test for existence

An error code is returned if either argument is illegal, or if the file can not be accessed in all of the specified modes. 0 is returned if the specified access would be successful.

FILES

/usr/lib/libU77.a

SEE ALSO

access(2), perror(3F)

BUGS

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

NAME

alarm — execute a subroutine after a specified time

SYNOPSIS

integer function alarm (time, proc)
integer time
external proc

DESCRIPTION

This routine arranges for subroutine *proc* to be called after *time* seconds. If *time* is "0", the alarm is turned off and no routine will be called. The returned value will be the time remaining on the last alarm.

FILES

/usr/lib/libU77.a

SEE ALSO

alarm(3C), sleep(3F), signal(3F)

BUGS

Alarm and *sleep* interact. If *sleep* is called after *alarm*, the *alarm* process will never be called. SIGALRM will occur at the lesser of the remaining *alarm* time or the *sleep* time.

NAME

bessel functions — of two kinds for integer orders

SYNOPSIS

function besj0 (x)

function besj1 (x)

function besjn (n, x)

function besy0 (x)

function besy1 (x)

function besyn (n, x)

double precision function dbesj0 (x)
double precision x

double precision function dbesj1 (x)
double precision x

double precision function dbesjn (n, x)
double precision x

double precision function dbesy0 (x)
double precision x

double precision function dbesy1 (x)
double precision x

double precision function dbesyn (n, x)
double precision x

DESCRIPTION

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

DIAGNOSTICS

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

FILES

/usr/lib/libF77.a

SEE ALSO

j0(3M), perror(3F)

NAME

bit – and, or, xor, not, rshift, lshift bitwise functions

SYNOFSIS

(intrinsic) function and (word1, word2)

(intrinsic) function or (word1, word2)

(intrinsic) function xor (word1, word2)

(intrinsic) function not (word)

(intrinsic) function rshift (word, nbits)

(intrinsic) function lshift (word, nbits)

DESCRIPTION

These bitwise functions are built into the compiler and return the data type of their argument(s). It is recommended that their arguments be **integer** values; inappropriate manipulation of **real** objects may cause unexpected results.

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

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

FILES

These functions are generated in-line by the f77 compiler.

NAME
chdir — change default directory

SYNOPSIS
integer function chdir (dirname)
character*(*) dirname

DESCRIPTION
The default directory for creating and locating files will be changed to *dirname*. Zero is returned if successful; an error code otherwise.

FILES
/usr/lib/libU77.a

SEE ALSO
chdir(2), cd(1), perror(3F)

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

Use of this function may cause **inquire** by unit to fail.

NAME

chmod − change mode of a file

SYNOPSIS

integer function chmod (name, mode)
character*(*) name, mode

DESCRIPTION

This function changes the filesystem *mode* of file *name*. *Mode* can be any specification recognized by *chmod*(1). *Name* must be a single pathname.

The normal returned value is 0. Any other value will be a system error number.

FILES

/usr/lib/libU77.a	
/bin/chmod	exec'ed to change the mode.

SEE ALSO

chmod(1)

BUGS

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

NAME

etime, dtime — return elapsed execution time

SYNOPSIS

function etime (tarray)
real tarray (2)

function dtime (tarray)
real tarray (2)

DESCRIPTION

These two routines return elapsed runtime in seconds for the calling process. *Dtime* returns the elapsed time since the last call to *dtime,* or the start of execution on the first call.

The argument array returns user time in the first element and system time in the second element. The function value is the sum of user and system time.

The resolution of all timing is 1/HZ sec. where HZ is currently 60.

FILES

/usr/lib/libU77.a

SEE ALSO

times (2)

NAME

exit — terminate process with status

SYNOPSIS

subroutine exit (status)
integer status

DESCRIPTION

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

This call will never return.

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

FILES

/usr/lib/libF77.a

SEE ALSO

exit(2), fork(2), fork(3F), wait(2), wait(3F)

NAME

fdate − return date and time in an ASCII string

SYNOPSIS

subroutine fdate (string)
character*(*) string

character*(*) function fdate()

DESCRIPTION

Fdate returns the current date and time as a 24 character string in the format described under *ctime*(3). Neither 'newline' nor NULL will be included.

Fdate can be called either as a function or as a subroutine. If called as a function, the calling routine must define its type and length. For example:

```
character*24  fdate
external      fdate

write(*,*) fdate()
```

FILES

/usr/lib/libU77.a

SEE ALSO

ctime(3), time(3F), itime(3F), idate(3F), ltime(3F)

NAME

flmin, flmax, ffrac, dflmin, dflmax, dffrac, inmax — return extreme values

SYNOPSIS

function flmin ()

function flmax ()

function ffrac ()

double precision function dflmin ()

double precision function dflmax ()

double precision function dffrac ()

function inmax ()

DESCRIPTION

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

The functions *ffrac* and *dffrac* return the fractional accuracy of single and double precision floating point numbers respectively. These are the smallest numbers that can be added to 1.0 without being lost.

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

FILES

/usr/lib/libF77.a

NAME
flush − flush output to a logical unit

SYNOPSIS
subroutine flush (lunit)

DESCRIPTION
Flush causes the contents of the buffer for logical unit *lunit* to be flushed to the associated file. This is most useful for logical units 0 and 6 when they are both associated with the control terminal.

FILES
/usr/lib/libI77.a

SEE ALSO
fclose(3S)

NAME

fork — create a copy of this process

SYNOPSIS

integer function fork()

DESCRIPTION

Fork creates a copy of the calling process. The only distinction between the 2 processes is that the value returned to one of them (referred to as the 'parent' process) will be the process id if the copy. The copy is usually referred to as the 'child' process. The value returned to the 'child' process will be zero.

All logical units open for writing are flushed before the fork to avoid duplication of the contents of I/O buffers in the external file(s).

If the returned value is negative, it indicates an error and will be the negation of the system error code. See perror(3F).

A corresponding *exec* routine has not been provided because there is no satisfactory way to retain open logical units across the exec. However, the usual function of *fork/exec* can be performed using *system*(3F).

FILES

/usr/lib/libU77.a

SEE ALSO

fork(2), wait(3F), kill(3F), system(3F), perror(3F)

NAME

fseek, ftell — reposition a file on a logical unit

SYNOPSIS

integer function fseek (lunit, offset, from)
integer offset, from

integer function ftell (lunit)

DESCRIPTION

lunit must refer to an open logical unit. *offset* is an offset in bytes relative to the position specified by *from*. Valid values for *from* are:

0 meaning 'beginning of the file'
1 meaning 'the current position'
2 meaning 'the end of the file'

The value returned by *fseek* will be 0 if successful, a system error code otherwise. (See perror(3F))

Ftell returns the current position of the file associated with the specified logical unit. The value is an offset, in bytes, from the beginning of the file. If the value returned is negative, it indicates an error and will be the negation of the system error code. (See perror(3F))

FILES

/usr/lib/libU77.a

SEE ALSO

fseek(3S), perror(3F)

NAME

getarg, iargc — return command line arguments

SYNOPSIS

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

function iargc ()

DESCRIPTION

A call to *getarg* will return the **k***th* command line argument in character string *arg*. The 0*th* argument is the command name.

Iargc returns the index of the last command line argument.

FILES

/usr/lib/libU77.a

SEE ALSO

getenv(3F), execve(2)

NAME

getc, fgetc — get a character from a logical unit

SYNOPSIS

integer function getc (char)
character char

integer function fgetc (lunit, char)
character char

DESCRIPTION

These routines return the next character from a file associated with a fortran logical unit, bypassing normal fortran I/O. *Getc* reads from logical unit 5, normally connected to the control terminal input.

The value of each function is a system status code. Zero indicates no error occured on the read; −1 indicates end of file was detected. A positive value will be either a UNIX system error code or an f77 I/O error code. See perror(3F).

FILES

/usr/lib/libU77.a

SEE ALSO

getc(3S), intro(2), perror(3F)

NAME

getcwd − get pathname of current working directory

SYNOPSIS

integer function getcwd (dirname)
character*(*) dirname

DESCRIPTION

The pathname of the default directory for creating and locating files will be returned in *dirname*. The value of the function will be zero if successful; an error code otherwise.

FILES

/usr/lib/libU77.a

SEE ALSO

chdir(3F), perror(3F)

BUGS

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

NAME

getenv — get value of environment variables

SYNOPSIS

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

DESCRIPTION

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

FILES

/usr/lib/libU77.a

SEE ALSO

environ(7), execve(2)

NAME

getfsent, getfsspec, getfsfile, getfstype, setfsent, endfsent — get file system descriptor file entry

SYNOPSIS

#include <fstab.h>

struct fstab *getfsent()

struct fstab *getfsspec(spec)
char *spec;

struct fstab *getfsfile(file)
char *file;

struct fstab *getfstype(type)
char *type;

int setfsent()

int endfsent()

DESCRIPTION

Getfsent, *getfsspec*, *getfstype*, and *getfsfile* each return a pointer to an object with the following structure containing the broken-out fields of a line in the file system description file, < fstab.h >.

```
struct fstab{
        char    *fs_spec;
        char    *fs_file;
        char    *fs_type;
        int     fs_freq;
        int     fs_passno;
};
```

The fields have meanings described in *fstab*(5).

Getfsent reads the next line of the file, opening the file if necessary.

Setfsent opens and rewinds the file.

Endfsent closes the file.

Getfsspec and *getfsfile* sequentially search from the beginning of the file until a matching special file name or file system file name is found, or until EOF is encountered. *Getfstype* does likewise, matching on the file system type field.

FILES

/etc/fstab

SEE ALSO

fstab(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

getlog − get user's login name

SYNOPSIS

subroutine getlog (name)
character*(*) name

character*(*) function getlog()

DESCRIPTION

Getlog will return the user's login name or all blanks if the process is running detached from a terminal.

FILES

/usr/lib/libU77.a

SEE ALSO

getlogin(3)

NAME

getpid – get process id

SYNOPSIS

integer function getpid()

DESCRIPTION

Getpid returns the process ID number of the current process.

FILES

/usr/lib/libU77.a

SEE ALSO

getpid(2)

NAME

getuid, getgid – get user or group ID of the caller

SYNOPSIS

integer function getuid()

integer function getgid()

DESCRIPTION

These functions return the real user or group ID of the user of the process.

FILES

/usr/lib/libU77.a

SEE ALSO

getuid(2)

NAME

hostnm – get name of current host

SYNOPSIS

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

DESCRIPTION

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

FILES

/usr/lib/libU77.a

SEE ALSO

gethostname(2)

NAME

idate, itime − return date or time in numerical form

SYNOPSIS

subroutine idate (iarray)
integer iarray(3)

subroutine itime (iarray)
integer iarray(3)

DESCRIPTION

Idate returns the current date in *iarray*. The order is: day, mon, year. Month will be in the range 1-12. Year will be ⩾ 1969.

Itime returns the current time in *iarray*. The order is: hour, minute, second.

FILES

/usr/lib/libU77.a

SEE ALSO

ctime(3F), fdate(3F)

NAME

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

SYNOPSIS

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

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

function lnblnk (string)
character*(*) string

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

DESCRIPTION

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

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

FILES

/usr/lib/libF77.a

NAME

ioinit − change f77 I/O initialization

SYNOPSIS

logical function ioinit (cctl, bzro, apnd, prefix, vrbose)
logical cctl, bzro, apnd, vrbose
character*(*) prefix

DESCRIPTION

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

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

By default, trailing and embedded blanks in input data fields are ignored. If *bzro* is **.true.** then such blanks will be treated as zero's. Otherwise the default will be restored.

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

Many systems provide an automatic association of global names with fortran logical units when a program is run. There is no such automatic association in f77. However, if the argument *prefix* is a non-blank string, then names of the form **prefix**NN will be sought in the program environment. The value associated with each such name found will be used to open logical unit NN for formatted sequential access. For example, if f77 program *myprogram* included the call

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

then when the following sequence

```
% setenv FORT01 mydata
% setenv FORT12 myresults
% myprogram
```

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

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

The effect of

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

can be achieved without the actual call by including "−I66" on the *f77* command line. This gives carriage control on all logical units except 0, causes files to be opened at their beginning, and causes blanks to be interpreted as zero's.

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

```
integer*2 ieof, ictl, ibzr
```

```
        common /ioiflg/ ieof, ictl, ibzr
```

FILES

/usr/lib/libI77.a	f77 I/O library
/usr/lib/libI66.a	sets older fortran I/O modes

SEE ALSO

getarg(3F), getenv(3F), "Introduction to the f77 I/O Library"

BUGS

Prefix can be no longer than 30 characters. A pathname associated with an environment name can be no longer than 255 characters.

The "+" carriage control does not work.

NAME

kill − send a signal to a process

SYNOPSIS

function kill (pid, signum)
integer pid, signum

DESCRIPTION

Pid must be the process id of one of the user's processes. *Signum* must be a valid signal number (see sigvec(2)). The returned value will be 0 if successful; an error code otherwise.

FILES

/usr/lib/libU77.a

SEE ALSO

kill(2), sigvec(2), signal(3F), fork(3F), perror(3F)

NAME

link – make a link to an existing file

SYNOPSIS

function link (name1, name2)
character*(*) name1, name2

integer function symlnk (name1, name2)
character*(*) name1, name2

DESCRIPTION

Name1 must be the pathname of an existing file. *Name2* is a pathname to be linked to file *name1*. *Name2* must not already exist. The returned value will be 0 if successful; a system error code otherwise.

Symlnk creates a symbolic link to *name1*.

FILES

/usr/lib/libU77.a

SEE ALSO

link(2), symlink(2), perror(3F), unlink(3F)

BUGS

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

NAME
loc − return the address of an object

SYNOPSIS
function loc (arg)

DESCRIPTION
The returned value will be the address of *arg*.

FILES
/usr/lib/libU77.a

NAME

long, short — integer object conversion

SYNOPSIS

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

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

DESCRIPTION

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

FILES

/usr/lib/libF77.a

NAME

perror, gerror, ierrno − get system error messages

SYNOPSIS

subroutine perror (string)
character*(*) string

subroutine gerror (string)
character*(*) string

character*(*) function gerror()

function ierrno()

DESCRIPTION

Perror will write a message to fortran logical unit 0 appropriate to the last detected system error. *String* will be written preceding the standard error message.

Gerror returns the system error message in character variable *string*. *Gerror* may be called either as a subroutine or as a function.

Ierrno will return the error number of the last detected system error. This number is updated only when an error actually occurs. Most routines and I/O statements that might generate such errors return an error code after the call; that value is a more reliable indicator of what caused the error condition.

FILES

/usr/lib/libU77.a

SEE ALSO

intro(2), perror(3)
D. L. Wasley, *Introduction to the f77 I/O Library*

BUGS

String in the call to *perror* can be no longer than 127 characters.

The length of the string returned by *gerror* is determined by the calling program.

NOTES

UNIX system error codes are described in *intro*(2). The f77 I/O error codes and their meanings are:

100	"error in format"
101	"illegal unit number"
102	"formatted io not allowed"
103	"unformatted io not allowed"
104	"direct io not allowed"
105	"sequential io not allowed"
106	"can't backspace file"
107	"off beginning of record"
108	"can't stat file"
109	"no * after repeat count"
110	"off end of record"
111	"truncation failed"
112	"incomprehensible list input"
113	"out of free space"
114	"unit not connected"
115	"read unexpected character"

116	"blank logical input field"
117	"'new' file exists"
118	"can't find 'old' file"
119	"unknown system error"
120	"requires seek ability"
121	"illegal argument"
122	"negative repeat count"
123	"illegal operation for unit"

NAME

putc, fputc — write a character to a fortran logical unit

SYNOPSIS

integer function putc (char)
character char

integer function fputc (lunit, char)
character char

DESCRIPTION

These funtions write a character to the file associated with a fortran logical unit bypassing normal fortran I/O. *Putc* writes to logical unit 6, normally connected to the control terminal output.

The value of each function will be zero unless some error occurred; a system error code otherwise. See perror(3F).

FILES

/usr/lib/libU77.a

SEE ALSO

putc(3S), intro(2), perror(3F)

NAME

qsort — quick sort

SYNOPSIS

subroutine qsort (array, len, isize, compar)
external compar
integer*2 compar

DESCRIPTION

One dimensional *array* contains the elements to be sorted. *len* is the number of elements in the array. *isize* is the size of an element, typically -

4 for **integer** and **real**
8 for **double precision** or **complex**
16 for **double complex**
(length of character object) for **character** arrays

Compar is the name of a user supplied integer*2 function that will determine the sorting order. This function will be called with 2 arguments that will be elements of *array*. The function must return -

negative if arg 1 is considered to precede arg 2
zero if arg 1 is equivalent to arg 2
positive if arg 1 is considered to follow arg 2

On return, the elements of *array* will be sorted.

FILES

/usr/lib/libU77.a

SEE ALSO

qsort(3)

NAME

rand, drand, irand − return random values

SYNOPSIS

function irand (iflag)

function rand (iflag)

double precision function drand (iflag)

DESCRIPTION

These functions use *rand*(3C) to generate sequences of random numbers. If *iflag* is '1', the generator is restarted and the first random value is returned. If *iflag* is otherwise non-zero, it is used as a new seed for the random number generator, and the first new random value is returned.

Irand returns positive integers in the range 0 through 2147483647. *Rand* and *drand* return values in the range 0. through 1.0 .

FILES

/usr/lib/libF77.a

SEE ALSO

rand(3C)

BUGS

The algorithm returns a 15 bit quantity on the PDP11; a 31 bit quantity on the VAX. *Irand* on the PDP11 calls *rand*(3C) twice to form a 31 bit quantity, but bit 15 will always be 0.

NAME

flmin, flmax, dflmin, dflmax, inmax — return extreme values

SYNOPSIS

function flmin ()

function flmax ()

double precision function dflmin ()

double precision function dflmax ()

function inmax ()

DESCRIPTION

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

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

FILES

/usr/lib/libF77.a

NAME

rename − rename a file

SYNOPSIS

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

DESCRIPTION

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

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

FILES

/usr/lib/libU77.a

SEE ALSO

rename(2), perror(3F)

BUGS

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

NAME

signal – change the action for a signal

SYNOPSIS

integer function signal(signum, proc, flag)
integer signum, flag
external proc

DESCRIPTION

When a process incurs a signal (see *signal*(3C)) the default action is usually to clean up and abort. The user may choose to write an alternative signal handling routine. A call to *signal* is the way this alternate action is specified to the system.

Signum is the signal number (see *signal*(3C)). If *flag* is negative, then *proc* must be the name of the user signal handling routine. If *flag* is zero or positive, then *proc* is ignored and the value of *flag* is passed to the system as the signal action definition. In particular, this is how previously saved signal actions can be restored. Two possible values for *flag* have specific meanings: 0 means "use the default action" (See NOTES below), 1 means "ignore this signal".

A positive returned value is the previous action definition. A value greater than 1 is the address of a routine that was to have been called on occurrence of the given signal. The returned value can be used in subsequent calls to *signal* in order to restore a previous action definition. A negative returned value is the negation of a system error code. (See *perror*(3F))

FILES

/usr/lib/libU77.a

SEE ALSO

signal(3C), kill(3F), kill(1)

NOTES

f77 arranges to trap certain signals when a process is started. The only way to restore the default **f77** action is to save the returned value from the first call to *signal.*

If the user signal handler is called, it will be passed the signal number as an integer argument.

NAME

stat, lstat, fstat − get file status

SYNOPSIS

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

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

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

DESCRIPTION

These routines return detailed information about a file. *Stat* and *lstat* return information about file *name*; *fstat* returns information about the file associated with fortran logical unit *lunit.* The order and meaning of the information returned in array *statb* is as described for the structure *stat* under *stat*(2). The "spare" values are not included.

The value of either function will be zero if successful; an error code otherwise.

FILES

/usr/lib/libU77.a

SEE ALSO

stat(2), access(3F), perror(3F), time(3F)

BUGS

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

NAME
sleep — suspend execution for an interval

SYNOPSIS
subroutine sleep (itime)

DESCRIPTION
Sleep causes the calling process to be suspended for *itime* seconds. The actual time can be up to 1 second less than *itime* due to granularity in system timekeeping.

FILES
/usr/lib/libU77.a

SEE ALSO
sleep(3)

NAME

system — execute a UNIX command

SYNOPSIS

integer function system (string)
character•(•) string

DESCRIPTION

System causes *string* to be given to your shell as input as if the string had been typed as a command. If environment variable SHELL is found, its value will be used as the command interpreter (shell); otherwise *sh*(1) is used.

The current process waits until the command terminates. The returned value will be the exit status of the shell. See *wait*(2) for an explanation of this value.

FILES

/usr/lib/libU77.a

SEE ALSO

exec(2), wait(2), system(3)

BUGS

String can not be longer than NCARGS−50 characters, as defined in <*sys/param.h*>.

NAME

syslog, openlog, closelog — control system log

SYNOPSIS

#include <syslog.h>

openlog(ident, logstat)
char *ident;

syslog(priority, message, parameters ...)
char *message;

closelog()

DESCRIPTION

Syslog arranges to write the *message* onto the system log maintained by *syslog*(8). The message is tagged with *priority*. The message looks like a *printf*(3) string except that **%m** is replaced by the current error message (collected from *errno*). A trailing newline is added if needed. This message will be read by *syslog*(8) and output to the system console or files as appropriate.

If special processing is needed, *openlog* can be called to initialize the log file. Parameters are *ident* which is prepended to every message, and *logstat* which is a bit field indicating special status; current values are:

LOG_PID	log the process id with each message: useful for identifying instantiations of daemons.

Openlog returns zero on success. If it cannot open the file */dev/log,* it writes on */dev/console* instead and returns −1.

Closelog can be used to close the log file.

EXAMPLES

```
syslog(LOG_SALERT, "who: internal error 23");

openlog("serverftp", LOG_PID);
syslog(LOG_INFO, "Connection from host %d", CallingHost);
```

SEE ALSO

syslog(8)

NAME

time, ctime, ltime, gmtime — return system time

SYNOPSIS

integer function time()

character*(*) function ctime (stime)
integer stime

subroutine ltime (stime, tarray)
integer stime, tarray(9)

subroutine gmtime (stime, tarray)
integer stime, tarray(9)

DESCRIPTION

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds. This is the value of the UNIX system clock.

Ctime converts a system time to a 24 character ASCII string. The format is described under *ctime*(3). No 'newline' or NULL will be included.

Ltime and *gmtime* disect a UNIX time into month, day, etc., either for the local time zone or as GMT. The order and meaning of each element returned in *tarray* is described under *ctime*(3).

FILES

/usr/lib/libU77.a

SEE ALSO

ctime(3), itime(3F), idate(3F), fdate(3F)

NAME

topen, tclose, tread, twrite, trewin, tskipf, tstate − f77 tape I/O

SYNOPSIS

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

integer function tclose (tlu)
integer tlu

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

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

integer function trewin (tlu)
integer tlu

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

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

DESCRIPTION

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

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

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

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

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

Trewin rewinds the tape associated with *tlu* to the beginning of the first data file. If the tape is a labelled tape (see *topen* above) then the label is skipped over after rewinding. The normal returned value is 0. A negative value indicates an error.

Tskipf allows the user to skip over files and/or records. First, *nfiles* end-of-file marks are skipped. If the current file is at EOF, this counts as 1 file to skip. (Note: This is the way to reset the EOF status for a *tlu.*) Next, *nrecs* physical records are skipped over. The normal returned value is 0. A negative value indicates an error.

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

FILES

/usr/lib/libU77.a

SEE ALSO

ht(4), perror(3F), rewind(1)

NAME

traper — trap arithmetic errors

SYNOPSIS

integer function traper (mask)

DESCRIPTION

NOTE: This routine applies only to the VAX. It is ignored on the PDP11.

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

The argument has the following meaning:

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

The previous value of these bits is returned.

FILES

/usr/lib/libF77.a

SEE ALSO

signal(3C), signal(3F)

NAME

trapov — trap and repair floating point overflow

SYNOPSIS

subroutine trapov (numesg, rtnval)
double precision rtnval

DESCRIPTION

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

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

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

FILES

/usr/lib/libF77.a

SEE ALSO

trpfpe(3F), signal(3F), range(3F)

BUGS

Other arithmetic exceptions can be trapped but not repaired.

There is no way to distinguish between an integer value of 32768 and the illegal floating point form. Therefore such an integer value may get replaced while repairing the use of an illegal operand.

NAME

trpfpe, fpecnt — trap and repair floating point faults

SYNOPSIS

subroutine trpfpe (numesg, rtnval)
double precision rtnval

integer function fpecnt ()

common /fpeflt/ fperr
logical fperr

DESCRIPTION

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

Trpfpe sets up a signal handler to trap arithmetic exceptions. If the exception is due to a floating point arithmetic fault, the result of the operation is replaced with the *rtnval* specified. *Rtnval* must be a double precision value. For example, "0d0" or "dflmax()".

The first *numesg* occurrences of a floating point arithmetic error will cause a message to be written to the standard error file. Any exception that can't be repaired will result in the default action, typically an abort with core image.

Fpecnt returns the number of faults since the last call to *trpfpe*.

The logical value in the common block labelled *fpeflt* will be set to **.true.** each time a fault occurs.

FILES

/usr/lib/libF77.a

SEE ALSO

signal(3F), range(3F)

BUGS

This routine works only for *faults*, not *traps*. This is primarily due to the Vax architecture.

If the operation involves changing the stack pointer, it can't be repaired. This seldom should be a problem with the f77 compiler, but such an operation might be produced by the optimizer.

The POLY and EMOD opcodes are not dealt with.

NAME

ttynam, isatty − find name of a terminal port

SYNOPSIS

character*(*) function ttynam (lunit)

logical function isatty (lunit)

DESCRIPTION

Ttynam returns a blank padded path name of the terminal device associated with logical unit *lunit*.

Isatty returns **.true.** if *lunit* is associated with a terminal device, **.false.** otherwise.

FILES

/dev/*
/usr/lib/libU77.a

DIAGNOSTICS

Ttynam returns an empty string (all blanks) if *lunit* is not associated with a terminal device in directory '/dev'.

NAME

unlink – remove a directory entry

SYNOPSIS

integer function unlink (name)
character*(*) name

DESCRIPTION

Unlink causes the directory entry specified by pathname *name* to be removed. If this was the last link to the file, the contents of the file are lost. The returned value will be zero if successful; a system error code otherwise.

FILES

/usr/lib/libU77.a

SEE ALSO

unlink(2), link(3F), filsys(5), perror(3F)

BUGS

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

NAME
wait − wait for a process to terminate

SYNOPSIS
integer function wait (status)
integer status

DESCRIPTION
Wait causes its caller to be suspended until a signal is received or one of its child processes terminates. If any child has terminated since the last *wait,* return is immediate; if there are no children, return is immediate with an error code.

If the returned value is positive, it is the process ID of the child and *status* is its termination status (see *wait*(2)). If the returned value is negative, it is the negation of a system error code.

FILES
/usr/lib/libU77.a

SEE ALSO
wait(2), signal(3F), kill(3F), perror(3F)

NAME

intro − introduction to mathematical library functions

DESCRIPTION

These functions constitute the math library, *libm.* They are automatically loaded as needed by the Fortran compiler *f77*(1). The link editor searches this library under the "−lm" option. Declarations for these functions may be obtained from the include file <*math.h*>.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
acos	sin.3m	trigonometric functions
asin	sin.3m	trigonometric functions
atan	sin.3m	trigonometric functions
atan2	sin.3m	trigonometric functions
cabs	hypot.3m	Euclidean distance
ceil	floor.3m	absolute value, floor, ceiling functions
cos	sin.3m	trigonometric functions
cosh	sinh.3m	hyperbolic functions
exp	exp.3m	exponential, logarithm, power, square root
fabs	floor.3m	absolute value, floor, ceiling functions
floor	floor.3m	absolute value, floor, ceiling functions
gamma	gamma.3m	log gamma function
hypot	hypot.3m	Euclidean distance
j0	j0.3m	bessel functions
j1	j0.3m	bessel functions
jn	j0.3m	bessel functions
log	exp.3m	exponential, logarithm, power, square root
log10	exp.3m	exponential, logarithm, power, square root
pow	exp.3m	exponential, logarithm, power, square root
sin	sin.3m	trigonometric functions
sinh	sinh.3m	hyperbolic functions
sqrt	exp.3m	exponential, logarithm, power, square root
tan	sin.3m	trigonometric functions
tanh	sinh.3m	hyperbolic functions
y0	j0.3m	bessel functions
y1	j0.3m	bessel functions
yn	j0.3m	bessel functions

NAME

exp, log, log10, pow, sqrt − exponential, logarithm, power, square root

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

Exp returns the exponential function of *x*.

Log returns the natural logarithm of *x*; *log10* returns the base 10 logarithm.

Pow returns x^y.

Sqrt returns the square root of *x*.

SEE ALSO

hypot(3M), sinh(3M), intro(3M)

DIAGNOSTICS

Exp and *pow* return a huge value when the correct value would overflow; *errno* is set to ERANGE. *Pow* returns 0 and sets *errno* to EDOM when the second argument is negative and non-integral and when both arguments are 0.

Log returns 0 when *x* is zero or negative; *errno* is set to EDOM.

Sqrt returns 0 when *x* is negative; *errno* is set to EDOM.

NAME

fabs, floor, ceil — absolute value, floor, ceiling functions

SYNOPSIS

#include <math.h>

double floor(x)
double x;

double ceil(x)
double x;

double fabs(x)
double x;

DESCRIPTION

Fabs returns the absolute value $|x|$.

Floor returns the largest integer not greater than *x*.

Ceil returns the smallest integer not less than *x*.

SEE ALSO

abs(3)

NAME

gamma − log gamma function

SYNOPSIS

#include <math.h>

double gamma(x)
double x;

DESCRIPTION

Gamma returns ln $|\Gamma(|x|)|$. The sign of $\Gamma(|x|)$ is returned in the external integer *signgam*. The following C program might be used to calculate Γ:

```
y = gamma(x);
if (y > 88.0)
        error();
y = exp(y);
if(signgam)
        y = −y;
```

DIAGNOSTICS

A huge value is returned for negative integer arguments.

BUGS

There should be a positive indication of error.

NAME

hypot, cabs − Euclidean distance

SYNOPSIS

#include <math.h>

double hypot(x, y)
double x, y;

double cabs(z)
struct { double x, y;} z;

DESCRIPTION

Hypot and *cabs* return

sqrt(x*x + y*y),

taking precautions against unwarranted overflows.

SEE ALSO

exp(3M) for *sqrt*

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)
double x;

double y0(x)
double x;

double y1(x)
double x;

double yn(n, x)
double x;
```

DESCRIPTION

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

DIAGNOSTICS

Negative arguments cause *y0, y1,* and *yn* to return a huge negative value and set *errno* to EDOM.

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 asin(x)
double x;

double acos(x)
double x;

double atan(x)
double x;

double atan2(x, y)
double x, y;
```

DESCRIPTION

Sin, cos and *tan* return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

Asin returns the arc sin in the range $-\pi/2$ to $\pi/2$.

Acos returns the arc cosine in the range 0 to π.

Atan returns the arc tangent of *x* in the range $-\pi/2$ to $\pi/2$.

Atan2 returns the arc tangent of *x/y* in the range $-\pi$ to π.

DIAGNOSTICS

Arguments of magnitude greater than 1 cause *asin* and *acos* to return value 0; *errno* is set to EDOM. The value of *tan* at its singular points is a huge number, and *errno* is set to ERANGE.

BUGS

The value of *tan* for arguments greater than about 2**31 is garbage.

NAME

sinh, cosh, tanh — hyperbolic functions

SYNOPSIS

#include <math.h>

double sinh(x)

double cosh(x)
double x;

double tanh(x)
double x;

DESCRIPTION

These functions compute the designated hyperbolic functions for real arguments.

DIAGNOSTICS

Sinh and *cosh* return a huge value of appropriate sign when the correct value would overflow.

NAME

htonl, htons, ntohl, ntohs – convert values between host and network byte order

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>

netlong = htonl(hostlong);
u_long netlong, hostlong;

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

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

hostshort = ntohs(netshort);
u_short hostshort, netshort;
```

DESCRIPTION

These routines convert 16 and 32 bit quantities between network byte order and host byte order. On machines such as the SUN these routines are defined as null macros in the include file <*netinet/in.h*>.

These routines are most often used in conjunction with Internet addresses and ports as returned by *gethostent*(3N) and *getservent*(3N).

SEE ALSO

gethostent(3N), getservent(3N)

BUGS

The VAX handles bytes backwards from most everyone else in the world. This is not expected to be fixed in the near future.

NAME

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;

sethostent(stayopen)
int stayopen

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.

FILES

/etc/hosts

SEE ALSO

hosts(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.

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 non-zero, 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.

FILES

/etc/networks

SEE ALSO

networks(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only Internet network numbers are currently understood. Expecting network numbers to fit in no more than 32 bits is probably naive.

NAME

getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent — get protocol entry

SYNOPSIS

#include <netdb.h>

struct protoent *getprotoent()

struct protoent *getprotobyname(name)
char *name;

struct protoent *getprotobynumber(proto)
int proto;

setprotoent(stayopen)
int stayopen

endprotoent()

DESCRIPTION

Getprotoent, *getprotobyname*, and *getprotobynumber* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network protocol data base, */etc/protocols.*

```
struct  protoent {
        char    *p_name;        /* official name of protocol */
        char    **p_aliases;    /* alias list */
        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 non-zero, 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.

FILES

/etc/protocols

SEE ALSO

protocols(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet protocols are currently understood.

NAME

getservent, getservbyport, getservbyname, setservent, endservent — get service entry

SYNOPSIS

#include <netdb.h>

struct servent *getservent()

struct servent *getservbyname(name, proto)
char *name, *proto;

struct servent *getservbyport(port, proto)
int port; char *proto;

setservent(stayopen)
int stayopen

endservent()

DESCRIPTION

Getservent, *getservbyname*, and *getservbyport* each return a pointer to an object with the 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.

FILES

/etc/services

SEE ALSO

getprotoent(3N), services(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Expecting port numbers to fit in a 32 bit quantity is probably naive.

NAME

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. Note that when an Internet address is viewed as a 32-bit integer quantity on the VAX the bytes referred to above appear as "d.c.b.a". That is, VAX bytes are ordered from right to left.

When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three part address format convenient for specifying Class B network addresses as "128.net.host".

When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as "net.host".

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied as "parts" in a "." notation may be decimal, octal, or hexadecimal, as specified in the C language (i.e. a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

SEE ALSO

gethostent(3N), getnetent(3N), hosts(5), networks(5),

DIAGNOSTICS

The value −1 is returned by *inet_addr* and *inet_network* for malformed requests.

BUGS

The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed. The string returned by *inet_ntoa* resides in a static memory area.

NAME

stdio − standard buffered input/output package

SYNOPSIS

#include <stdio.h>

FILE *stdin;
FILE *stdout;
FILE *stderr;

DESCRIPTION

The functions described in section 3S constitute a user-level buffering scheme. The in-line macros *getc* and *putc*(3S) handle characters quickly. The higher level routines *gets*, *fgets*, *scanf*, *fscanf*, *fread*, *puts*, *fputs*, *printf*, *fprintf*, *fwrite* all use *getc* and *putc*; they can be freely intermixed.

A file with associated buffering is called a *stream*, and is declared to be a pointer to a defined type **FILE**. *Fopen*(3S) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

stdin	standard input file
stdout	standard output file
stderr	standard error file

A constant 'pointer' **NULL** (0) designates no stream at all.

An integer constant **EOF** (−1) is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file <*stdio.h*> of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include file and need no further declaration. The constants, and the following 'functions' are implemented as macros; redeclaration of these names is perilous: *getc*, *getchar*, *putc*, *putchar*, *feof*, *ferror*, *fileno*.

SEE ALSO

open(2), close(2), read(2), write(2), fread(3S), fseek(3S), f*(3S)

DIAGNOSTICS

The value **EOF** is returned uniformly to indicate that a **FILE** pointer has not been initialized with *fopen*, input (output) has been attempted on an output (input) stream, or a **FILE** pointer designates corrupt or otherwise unintelligible **FILE** data.

For purposes of efficiency, this implementation of the standard library has been changed to line buffer output to a terminal by default and attempts to do this transparently by flushing the output whenever a *read*(2) from the standard input is necessary. This is almost always transparent, but may cause confusion or malfunctioning of programs which use standard i/o routines but use *read*(2) themselves to read from the standard input.

In cases where a large amount of computation is done after printing part of a line on an output terminal, it is necessary to *fflush*(3S) the standard output before going off and computing so that the output will appear.

BUGS

The standard buffered functions do not interact well with certain other library and system functions, especially *vfork* and *abort*.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
clearerr	ferror.3s	stream status inquiries
fclose	fclose.3s	close or flush a stream

feof	ferror.3s	stream status inquiries
ferror	ferror.3s	stream status inquiries
fflush	fclose.3s	close or flush a stream
fgetc	getc.3s	get character or word from stream
fgets	gets.3s	get a string from a stream
fileno	ferror.3s	stream status inquiries
fprintf	printf.3s	formatted output conversion
fputc	putc.3s	put character or word on a stream
fputs	puts.3s	put a string on a stream
fread	fread.3s	buffered binary input/output
fscanf	scanf.3s	formatted input conversion
fseek	fseek.3s	reposition a stream
ftell	fseek.3s	reposition a stream
fwrite	fread.3s	buffered binary input/output
getc	getc.3s	get character or word from stream
getchar	getc.3s	get character or word from stream
gets	gets.3s	get a string from a stream
getw	getc.3s	get character or word from stream
printf	printf.3s	formatted output conversion
putc	putc.3s	put character or word on a stream
putchar	putc.3s	put character or word on a stream
puts	puts.3s	put a string on a stream
putw	putc.3s	put character or word on a stream
rewind	fseek.3s	reposition a stream
scanf	scanf.3s	formatted input conversion
setbuf	setbuf.3s	assign buffering to a stream
setbuffer	setbuf.3s	assign buffering to a stream
setlinebuf	setbuf.3s	assign buffering to a stream
sprintf	printf.3s	formatted output conversion
sscanf	scanf.3s	formatted input conversion
ungetc	ungetc.3s	push character back into input stream

NAME

fclose, fflush — close or flush a stream

SYNOPSIS

#include <stdio.h>

fclose(stream)
FILE *stream;

fflush(stream)
FILE *stream;

DESCRIPTION

Fclose causes any buffers for the named *stream* to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling *exit*(3).

Fflush causes any buffered data for the named output *stream* to be written to that file. The stream remains open.

SEE ALSO

close(2), fopen(3S), setbuf(3S)

DIAGNOSTICS

These routines return **EOF** if *stream* is not associated with an output file, or if buffered data cannot be transferred to that file.

NAME

ferror, feof, clearerr, fileno — stream status inquiries

SYNOPSIS

#include <stdio.h>

feof(stream)
FILE *stream;

ferror(stream)
FILE *stream

clearerr(stream)
FILE *stream

fileno(stream)
FILE *stream;

DESCRIPTION

Feof returns non-zero when end of file is read on the named input *stream*, otherwise zero.

Ferror returns non-zero when an error has occurred reading or writing the named *stream*, otherwise zero. Unless cleared by *clearerr*, the error indication lasts until the stream is closed.

Clrerr resets the error indication on the named *stream*.

Fileno returns the integer file descriptor associated with the *stream*, see *open*(2).

These functions are implemented as macros; they cannot be redeclared.

SEE ALSO

fopen(3S), open(2)

NAME

fopen, freopen, fdopen — open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *fopen(filename, type)
char *filename, *type;

FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen(fildes, type)
char *type;
```

DESCRIPTION

Fopen opens the file named by *filename* and associates a stream with it. *Fopen* returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

"r"	open for reading
"w"	create for writing
"a"	append: open for writing at end of file, or create for writing

In addition, each *type* may be followed by a '+' to have the file opened for reading and writing. "r+" positions the stream at the beginning of the file, "w+" creates or truncates it, and "a+" positions it at the end. Both reads and writes may be used on read/write streams, with the limitation that an *fseek, rewind,* or reading an end-of-file must be used between a read and a write or vice-versa.

Freopen substitutes the named file in place of the open *stream*. It returns the original value of *stream*. The original stream is closed.

Freopen is typically used to attach the preopened constant names, **stdin, stdout, stderr,** to specified files.

Fdopen associates a stream with a file descriptor obtained from *open, dup, creat,* or *pipe*(2). The *type* of the stream must agree with the mode of the open file.

SEE ALSO

open(2), fclose(3)

DIAGNOSTICS

Fopen and *freopen* return the pointer NULL if *filename* cannot be accessed.

BUGS

Fdopen is not portable to systems other than UNIX.

The read/write *types* do not exist on all systems. Those systems without read/write modes will probably treat the *type* as if the '+' was not present. These are unreliable in any event.

NAME

fread, fwrite — buffered binary input/output

SYNOPSIS

```
#include <stdio.h>

fread(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

fwrite(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;
```

DESCRIPTION

Fread reads, into a block beginning at *ptr*, *nitems* of data of the type of **ptr* from the named input *stream*. It returns the number of items actually read.

If *stream* is **stdin** and the standard output is line buffered, then any partial output line will be flushed before any call to *read*(2) to satisfy the *fread*.

Fwrite appends at most *nitems* of data of the type of **ptr* beginning at *ptr* to the named output *stream*. It returns the number of items actually written.

SEE ALSO

read(2), write(2), fopen(3S), getc(3S), putc(3S), gets(3S), puts(3S), printf(3S), scanf(3S)

DIAGNOSTICS

Fread and *fwrite* return 0 upon end of file or error.

NAME

fseek, ftell, rewind — reposition a stream

SYNOPSIS

#include <stdio.h>

fseek(stream, offset, ptrname)
FILE *stream;
long offset;

long ftell(stream)
FILE *stream;

rewind(stream)

DESCRIPTION

Fseek sets the position of the next input or output operation on the *stream*. The new position is at the signed distance *offset* bytes from the beginning, the current position, or the end of the file, according as *ptrname* has the value 0, 1, or 2.

Fseek undoes any effects of *ungetc*(3S).

Ftell returns the current value of the offset relative to the beginning of the file associated with the named *stream*. It is measured in bytes on UNIX; on some other systems it is a magic cookie, and the only foolproof way to obtain an *offset* for *fseek*.

Rewind(*stream*) is equivalent to *fseek*(*stream*, 0L, 0).

SEE ALSO

lseek(2), fopen(3S)

DIAGNOSTICS

Fseek returns −1 for improper seeks.

NAME

getc, getchar, fgetc, getw − get character or word from stream

SYNOPSIS

#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar()

int fgetc(stream)
FILE *stream;

int getw(stream)
FILE *stream;

DESCRIPTION

Getc returns the next character from the named input *stream*.

Getchar() is identical to *getc*(*stdin*).

Fgetc behaves like *getc*, but is a genuine function, not a macro; it may be used to save object text.

Getw returns the next word (in a 32-bit integer on a VAX-11) from the named input *stream*. It returns the constant **EOF** upon end of file or error, but since that is a good integer value, *feof* and *ferror*(3S) should be used to check the success of *getw*. *Getw* assumes no special alignment in the file.

SEE ALSO

fopen(3S), putc(3S), gets(3S), scanf(3S), fread(3S), ungetc(3S)

DIAGNOSTICS

These functions return the integer constant **EOF** at end of file or upon read error.

A stop with message, 'Reading bad file', means an attempt has been made to read from a stream that has not been opened for reading by *fopen*.

BUGS

The end-of-file return from *getchar* is incompatible with that in UNIX editions 1-6.

Because it is implemented as a macro, *getc* treats a *stream* argument with side effects incorrectly. In particular, 'getc(*f++);' doesn't work sensibly.

NAME

gets, fgets − get a string from a stream

SYNOPSIS

#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
FILE *stream;

DESCRIPTION

Gets reads a string into *s* from the standard input stream **stdin**. The string is terminated by a newline character, which is replaced in *s* by a null character. *Gets* returns its argument.

Fgets reads $n-1$ characters, or up to a newline character, whichever comes first, from the *stream* into the string *s*. The last character read into *s* is followed by a null character. *Fgets* returns its first argument.

SEE ALSO

puts(3S), getc(3S), scanf(3S), fread(3S), ferror(3S)

DIAGNOSTICS

Gets and *fgets* return the constant pointer **NULL** upon end of file or error.

BUGS

Gets deletes a newline, *fgets* keeps it, all in the name of backward compatibility.

NAME

printf, fprintf, sprintf — formatted output conversion

SYNOPSIS

#include <stdio.h>

printf(format [, arg] ... **)**
char *format;

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

sprintf(s, format [, arg] ... **)**
char *s, format;

#include <varargs.h>
_doprnt(format, args, stream)
char *format;
va_list *args;
FILE *stream;

DESCRIPTION

Printf places output on the standard output stream **stdout**. *Fprintf* places output on the named output *stream*. *Sprintf* places 'output' in the string *s*, followed by the character '\0'. All of these routines work by calling the internal routine **_doprnt**, using the variable-length argument facilities of *varargs*(3).

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive *arg printf.*

Each conversion specification is introduced by the character **%**. Following the **%**, there may be

- an optional minus sign '−' which specifies *left adjustment* of the converted value in the indicated field;
- an optional digit string specifying a *field width;* if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;
- an optional period '**.**' which serves to separate the field width from the next digit string;
- an optional digit string specifying a *precision* which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;
- an optional '#' character specifying that the value should be converted to an "alternate form". For **c**, **d**, **s**, and **u**, conversions, this option has no effect. For **o** conversions, the precision of the number is increased to force the first character of the output string to a zero. For **x**(**X**) conversion, a non-zero result has the string **0x**(**0X**) prepended to it. For **e**, **E**, **f**, **g**, and **G**, conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point only appears in the results of those conversions if a digit follows the decimal point). For **g** and **G** conversions, trailing zeros are not removed from the result as they would otherwise be.
- the character **l** specifying that a following **d**, **o**, **x**, or **u** corresponds to a long integer *arg*.
- a character which indicates the type of conversion to be applied.

A field width or precision may be '*' instead of a digit string. In this case an integer *arg* supplies the field width or precision.

The conversion characters and their meanings are

dox	The integer *arg* is converted to decimal, octal, or hexadecimal notation respectively.
f	The float or double *arg* is converted to decimal notation in the style '[−]ddd.ddd' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.
e	The float or double *arg* is converted in the style '[−]d.ddde±dd' where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced.
g	The float or double *arg* is printed in style **d**, in style **f**, or in style **e**, whichever gives full precision in minimum space.
c	The character *arg* is printed.
s	*Arg* is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.
u	The unsigned integer *arg* is converted to decimal and printed (the result will be in the range 0 through MAXUINT, where MAXUINT equals 4294967295 on a VAX-11 and 65535 on a PDP-11).
%	Print a '%'; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by *printf* are printed by *putc*(3S).

Examples

To print a date and time in the form 'Sunday, July 3, 10:02', where *weekday* and *month* are pointers to null-terminated strings:

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

To print π to 5 decimals:

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

SEE ALSO

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

BUGS

Very wide fields (>128 characters) fail.

NAME

putc, putchar, fputc, putw − put character or word on a stream

SYNOPSIS

#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

putchar(c)

fputc(c, stream)
FILE *stream;

putw(w, stream)
FILE *stream;

DESCRIPTION

Putc appends the character *c* to the named output *stream*. It returns the character written.

Putchar(*c*) is defined as *putc*(*c*, **stdout**).

Fputc behaves like *putc*, but is a genuine function rather than a macro.

Putw appends word (that is, **int**) *w* to the output *stream*. It returns the word written. *Putw* neither assumes nor causes special alignment in the file.

SEE ALSO

fopen(3S), fclose(3S), getc(3S), puts(3S), printf(3S), fread(3S)

DIAGNOSTICS

These functions return the constant **EOF** upon error. Since this is a good integer, *ferror*(3S) should be used to detect *putw* errors.

BUGS

Because it is implemented as a macro, *putc* treats a *stream* argument with side effects improperly. In particular

```
putc(c, *f++);
```

doesn't work sensibly.

Errors can occur long after the call to *putc*.

NAME

puts, fputs − put a string on a stream

SYNOPSIS

#include <stdio.h>

puts(s)
char *s;

fputs(s, stream)
char *s;
FILE *stream;

DESCRIPTION

Puts copies the null-terminated string *s* to the standard output stream **stdout** and appends a newline character.

Fputs copies the null-terminated string *s* to the named output *stream*.

Neither routine copies the terminal null character.

SEE ALSO

fopen(3S), gets(3S), putc(3S), printf(3S), ferror(3S)
fread(3S) for *fwrite*

BUGS

Puts appends a newline, *fputs* does not, all in the name of backward compatibility.

NAME

scanf, fscanf, sscanf − formatted input conversion

SYNOPSIS

```
#include <stdio.h>

scanf(format [ , pointer ] . . . )
char *format;

fscanf(stream, format [ , pointer ] . . . )
FILE *stream;
char *format;

sscanf(s, format [ , pointer ] . . . )
char *s, *format;
```

DESCRIPTION

Scanf reads from the standard input stream **stdin**. *Fscanf* reads from the named input *stream*. *Sscanf* reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string *format*, described below, and a set of *pointer* arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines, which match optional white space in the input.
2. An ordinary character (not %) which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

% a single '%' is expected in the input at this point; no assignment is done.

d a decimal integer is expected; the corresponding argument should be an integer pointer.

o an octal integer is expected; the corresponding argument should be a integer pointer.

x a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

s a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating '\0', which will be added. The input field is terminated by a space character or a newline.

c a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try '%1s'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

e
f a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits possibly containing a decimal point, followed by an optional exponent field consisting of an E or e followed by

an optionally signed integer.

[indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters **d**, **o** and **x** may be capitalized or preceded by **l** to indicate that a pointer to **long** rather than to **int** is in the argument list. Similarly, the conversion characters **e** or **f** may be capitalized or preceded by **l** to indicate a pointer to **double** rather than to **float**. The conversion characters **d**, **o** and **x** may be preceded by **h** to indicate a pointer to **short** rather than to **int**.

The *scanf* functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant **EOF** is returned upon end of input; note that this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```
        int i; float x; char name[50];
        scanf("%d%f%s", &i, &x, name);
```

with the input line

```
    25   54.32E−1  thompson
```

will assign to *i* the value 25, *x* the value 5.432, and *name* will contain *'thompson\0'*. Or,

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

with input

```
    56789 0123 56a72
```

will assign 56 to *i*, 789.0 to *x*, skip '0123', and place the string '56\0' in *name*. The next call to *getchar* will return 'a'.

SEE ALSO

atof(3), getc(3S), printf(3S)

DIAGNOSTICS

The *scanf* functions return **EOF** on end of input, and a short count for missing or illegal data items.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.

NAME

setbuf, setbuffer, setlinebuf — assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;

setbuffer(stream, buf, size)
FILE *stream;
char *buf;
int size;

setlinebuf(stream)
FILE *stream;
```

DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a newline is encountered or input is read from stdin. *Fflush* (see *fclose*(3S)) may be used to force the block out early. Normally all files are block buffered. A buffer is obtained from *malloc*(3) upon the first *getc* or *putc*(3S) on the file. If the standard stream **stdout** refers to a terminal it is line buffered. The standard stream **stderr** is always unbuffered.

Setbuf is used after a stream has been opened but before it is read or written. The character array *buf* is used instead of an automatically allocated buffer. If *buf* is the constant pointer NULL, input/output will be completely unbuffered. A manifest constant **BUFSIZ** tells how big an array is needed:

```
char buf[BUFSIZ];
```

Setbuffer, an alternate form of *setbuf*, is used after a stream has been opened but before it is read or written. The character array *buf* whose size is determined by the *size* argument is used instead of an automatically allocated buffer. If *buf* is the constant pointer NULL, input/output will be completely unbuffered.

Setlinebuf is used to change *stdout* or *stderr* from block buffered or unbuffered to line buffered. Unlike *setbuf* and *setbuffer* it can be used at any time that the file descriptor is active.

A file can be changed from unbuffered or line buffered to block buffered by using *freopen* (see *fopen*(3S)). A file can be changed from block buffered or line buffered to unbuffered by using *freopen* followed by *setbuf* with a buffer argument of NULL.

SEE ALSO

fopen(3S), getc(3S), putc(3S), malloc(3), fclose(3S), puts(3S), printf(3S), fread(3S)

BUGS

The standard error stream should be line buffered by default.

The *setbuffer* and *setlinebuf* functions are not portable to non 4.2 BSD versions of UNIX.

NAME

ungetc − push character back into input stream

SYNOPSIS

#include <stdio.h>

ungetc(c, stream)
FILE *stream;

DESCRIPTION

Ungetc pushes the character *c* back on an input stream. That character will be returned by the next *getc* call on that stream. *Ungetc* returns *c*.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push EOF are rejected.

Fseek(3S) erases all memory of pushed back characters.

SEE ALSO

getc(3S), setbuf(3S), fseek(3S)

DIAGNOSTICS

Ungetc returns **EOF** if it can't push a character back.

NAME

intro − introduction to miscellaneous library functions

DESCRIPTION

These functions constitute minor libraries and other miscellaneous run-time facilities. Most are available only when programming in C. The list below includes libraries which provide device independent plotting functions, terminal independent screen management routines for two dimensional non-bitmap display terminals, functions for managing data bases with inverted indexes, and sundry routines used in executing commands on remote machines. The routines *getdiskbyname*, *rcmd*, *rresvport*, *ruserok*, and *rexec* reside in the standard C run-time library "−lc". All other functions are located in separate libraries indicated in each manual entry.

FILES

/lib/libc.a
/usr/lib/libdbm.a
/usr/lib/libtermcap.a
/usr/lib/libcurses.a
/usr/lib/lib2648.a
/usr/lib/libplot.a

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
arc	plot.3x	graphics interface
assert	assert.3x	program verification
circle	plot.3x	graphics interface
closepl	plot.3x	graphics interface
cont	plot.3x	graphics interface
curses	curses.3x	screen functions with "optimal" cursor motion
dbminit	dbm.3x	data base subroutines
delete	dbm.3x	data base subroutines
endfsent	getfsent.3x	get file system descriptor file entry
erase	plot.3x	graphics interface
fetch	dbm.3x	data base subroutines
firstkey	dbm.3x	data base subroutines
getdiskbyname	getdisk.3x	get disk description by its name
getfsent	getfsent.3x	get file system descriptor file entry
getfsfile	getfsent.3x	get file system descriptor file entry
getfsspec	getfsent.3x	get file system descriptor file entry
getfstype	getfsent.3x	get file system descriptor file entry
initgroups	initgroups.3x	initialize group access list
label	plot.3x	graphics interface
lib2648	lib2648.3x	subroutines for the HP 2648 graphics terminal
line	plot.3x	graphics interface
linemod	plot.3x	graphics interface
move	plot.3x	graphics interface
nextkey	dbm.3x	data base subroutines
plot: openpl	plot.3x	graphics interface
point	plot.3x	graphics interface
rcmd	rcmd.3x	routines for returning a stream to a remote command
rexec	rexec.3x	return stream to a remote command
rresvport	rcmd.3x	routines for returning a stream to a remote command
ruserok	rcmd.3x	routines for returning a stream to a remote command
setfsent	getfsent.3x	get file system descriptor file entry
space	plot.3x	graphics interface

store	dbm.3x	data base subroutines
tgetent	termcap.3x	terminal independent operation routines
tgetflag	termcap.3x	terminal independent operation routines
tgetnum	termcap.3x	terminal independent operation routines
tgetstr	termcap.3x	terminal independent operation routines
tgoto	termcap.3x	terminal independent operation routines
tputs	termcap.3x	terminal independent operation routines

NAME

assert − program verification

SYNOPSIS

#include <assert.h>

assert(expression)

DESCRIPTION

Assert is a macro that indicates *expression* is expected to be true at this point in the program. It causes an *exit*(2) with a diagnostic comment on the standard output when *expression* is false (0). Compiling with the *cc*(1) option **−DNDEBUG** effectively deletes *assert* from the program.

DIAGNOSTICS

'Assertion failed: file *f* line *n.*' *F* is the source file and *n* the source line number of the *assert* statement.

NAME

curses — screen functions with "optimal" cursor motion

SYNOPSIS

cc [flags] files **−lcurses −ltermcap** [libraries]

DESCRIPTION

These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen, and the user sets up an image of a new one. Then the *refresh()* tells the routines to make the current screen look like the new one. In order to initialize the routines, the routine *initscr()* must be called before any of the other routines that deal with windows and screens are used. The routine *endwin()* should be called before exiting.

SEE ALSO

Screen Updating and Cursor Movement Optimization: A Library Package, Ken Arnold, ioctl(2), getenv(3), tty(4), termcap(5)

AUTHOR

Ken Arnold

FUNCTIONS

addch(ch)	add a character to *stdscr*
addstr(str)	add a string to *stdscr*
box(win,vert,hor)	draw a box around a window
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*

BUGS

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.**

Keys and *contents* are described by the *datum* typedef. A *datum* specifies a string of *dsize* bytes pointed to by *dptr.* Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has '.dir' as its suffix. The second file contains all data and has '.pag' as its suffix.

Before a database can be accessed, it must be opened by *dbminit.* At the time of this call, the files *file*.**dir** and *file*.**pag** must exist. (An empty database is created by creating zero-length '.dir' and '.pag' files.)

Once open, the data stored under a key is accessed by *fetch* and data is placed under a key by *store.* A key (and its associated contents) is deleted by *delete.* A linear pass through all keys in a database may be made, in an (apparently) random order, by use of *firstkey* and *nextkey. Firstkey* will return the first key in the database. With any key *nextkey* will return the next key in the database. This code will traverse the data base:

```
for (key = firstkey(); key.dptr != NULL; key = nextkey(key))
```

DIAGNOSTICS

All functions that return an *int* indicate errors with negative values. A zero return indicates ok. Routines that return a *datum* indicate errors with a null (0) *dptr.*

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.

Because of ambiguities in this man page, there is no consistent style as to whether or not the "dsize" field includes the trailing null in a name. If you are reading a database generated by some other program, as for example the sendmail alias database /usr/lib/aliases, you must know whether its usage has included the null in the character count (aliases does).

NAME

getdiskbyname − get disk description by its name

SYNOPSIS

#include <disktab.h>

struct disktab *
getdiskbyname(name)
char *name;

DESCRIPTION

Getdiskbyname takes a disk name (e.g. rm03) and returns a structure describing its geometry information and the standard disk partition tables. All information obtained from the *disktab*(5) file.

<disktab.h> has the following form:

```
/*      @(#)disktab.h 4.2 (Berkeley) 3/6/83   */

/*
 * Disk description table, see disktab(5)
 */
#define DISKTAB                 "/etc/disktab"

struct  disktab {
        char    *d_name;                /* drive name */
        char    *d_type;                /* drive type */
        int     d_secsize;              /* sector size in bytes */
        int     d_ntracks;              /* # tracks/cylinder */
        int     d_nsectors;             /* # sectors/track */
        int     d_ncylinders;           /* # cylinders */
        int     d_rpm;                  /* revolutions/minute */
        struct  partition {
                int     p_size;         /* #sectors in partition */
                short   p_bsize;/* block size in bytes */
                short   p_fsize; /* frag size in bytes */
        } d_partitions[8];
};

struct  disktab *getdiskbyname();
```

SEE ALSO

disktab(5)

BUGS

This information should be obtained from the system for locally available disks (in particular, the disk partition tables).

NAME

initgroups – initialize group access list

SYNOPSIS

initgroups(name, basegid)
char *name;
int basegid;

DESCRIPTION

Initgroups reads through the group file and sets up, using the *setgroups*(2) call, the group access list for the user specified in *name*. The *basegid* is automatically included in the groups list. Typically this value is given as the group number from the password file.

FILES

/etc/group

SEE ALSO

setgroups(2)

DIAGNOSTICS

Initgroups returns −1 if it was not invoked by the super-user.

BUGS

Initgroups uses the routines based on *getgrent*(3). If the invoking program uses any of these routines, the group structure will be overwritten in the call to *initgroups*.

Noone seems to keep /etc/group up to date.

NAME

lib2648 − subroutines for the HP 2648 graphics terminal

SYNOPSIS

#include <stdio.h>

typedef char *bitmat;
FILE *trace;

cc file.c **−l2648**

DESCRIPTION

Lib2648 is a general purpose library of subroutines useful for interactive graphics on the Hewlett-Packard 2648 graphics terminal. To use it you must call the routine *ttyinit*() at the beginning of execution, and *done*() at the end of execution. All terminal input and output must go through the routines *rawchar*, *readline*, *outchar*, and *outstr*.

Lib2648 does the necessary ^E/^F handshaking if *getenv("TERM")* returns "hp2648", as it will if set by *tset*(1). Any other value, including for example "2648", will disable handshaking.

Bit matrix routines are provided to model the graphics memory of the 2648. These routines are generally useful, but are specifically useful for the *update* function which efficiently changes what is on the screen to what is supposed to be on the screen. The primative bit matrix routines are *newmat*, *mat*, and *setmat*.

The file *trace*, if non-null, is expected to be a file descriptor as returned by *fopen*. If so, *lib2648* will trace the progress of the output by writing onto this file. It is provided to make debugging output feasible for graphics programs without messing up the screen or the escape sequences being sent. Typical use of trace will include:

```
switch (argv[1][1]) {
case 'T':
        trace = fopen("trace", "w");
        break;
...
if (trace)
        fprintf(trace, "x is %d, y is %s\n", x, y);
...
dumpmat("before update", xmat);
```

ROUTINES

agoto(x, y)
Move the alphanumeric cursor to position (x, y), measured from the upper left corner of the screen.

aoff() Turn the alphanumeric display off.

aon() Turn the alphanumeric display on.

areaclear(rmin, cmin, rmax, cmax)
Clear the area on the graphics screen bordered by the four arguments. In normal mode the area is set to all black, in inverse video mode it is set to all white.

beep() Ring the bell on the terminal.

bitcopy(dest, src, rows, cols) bitmat dest,
Copy a *rows* by *cols* bit matrix from *src* to (user provided) *dest.*

cleara()
Clear the alphanumeric display.

clearg()

Clear the graphics display. Note that the 2648 will only clear the part of the screen that is visible if zoomed in.

curoff()
Turn the graphics cursor off.

curon()
Turn the graphics cursor on.

dispmsg(str, x, y, maxlen) char *str;
Display the message *str* in graphics text at position *(x, y)*. The maximum message length is given by *maxlen*, and is needed to for dispmsg to know how big an area to clear before drawing the message. The lower left corner of the first character is at *(x, y)*.

done() Should be called before the program exits. Restores the tty to normal, turns off graphics screen, turns on alphanumeric screen, flushes the standard output, etc.

draw(x, y)
Draw a line from the pen location to *(x, y)*. As with all graphics coordinates, *(x, y)* is measured from the bottom left corner of the screen. *(x, y)* coordinates represent the first quadrant of the usual Cartesian system.

drawbox(r, c, color, rows, cols)
Draw a rectangular box on the graphics screen. The lower left corner is at location *(r, c)*. The box is *rows* rows high and *cols* columns wide. The box is drawn if *color* is 1, erased if *color* is 0. *(r, c)* absolute coordinates represent row and column on the screen, with the origin at the lower left. They are equivalent to *(x, y)* except for being reversed in order.

dumpmat(msg, m, rows, cols) char *msg; bitmat m;
If *trace* is non-null, write a readable ASCII representation of the matrix *m* on *trace*. *Msg* is a label to identify the output.

emptyrow(m, rows, cols, r) bitmat m;
Returns 1 if row *r* of matrix *m* is all zero, else returns 0. This routine is provided because it can be implemented more efficiently with a knowledge of the internal representation than a series of calls to *mat*.

error(msg) char *msg;
Default error handler. Calls *message(msg)* and returns. This is called by certain routines in *lib2648*. It is also suitable for calling by the user program. It is probably a good idea for a fancy graphics program to supply its own error procedure which uses *setjmp*(3) to restart the program.

gdefault()
Set the terminal to the default graphics modes.

goff() Turn the graphics display off.

gon() Turn the graphics display on.

koff() Turn the keypad off.

kon() Turn the keypad on. This means that most special keys on the terminal (such as the alphanumeric arrow keys) will transmit an escape sequence instead of doing their function locally.

line(x1, y1, x2, y2)
Draw a line in the current mode from *(x1, y1)* to *(x2, y2)*. This is equivalent to *move(x1, y1); draw(x2, y2);* except that a bug in the terminal involving repeated lines from the same point is compensated for.

lowleft()
Move the alphanumeric cursor to the lower left (home down) position.

mat(m, rows, cols, r, c) bitmat m;
Used to retrieve an element from a bit matrix. Returns 1 or 0 as the value of the *[r, c]* element of the *rows* by *cols* matrix *m*. Bit matrices are numbered *(r, c)* from the upper left corner of the matrix, beginning at (0, 0). *R* represents the row, and *c* represents the column.

message(str) char *str;
Display the text message *str* at the bottom of the graphics screen.

minmax(g, rows, cols, rmin, cmin, rmax, cmax) bitmat g;
int *rmin, *cmin, *rmax, *cmax;
Find the smallest rectangle that contains all the 1 (on) elements in the bit matrix g. The coordinates are returned in the variables pointed to by rmin, cmin, rmax, cmax.

move(x, y)
Move the pen to location *(x, y)*. Such motion is internal and will not cause output until a subsequent *sync()*.

movecurs(x, y)
Move the graphics cursor to location *(x, y)*.

bitmat newmat(rows, cols)
Create (with *malloc*(3)) a new bit matrix of size *rows* by *cols*. The value created (e.g. a pointer to the first location) is returned. A bit matrix can be freed directly with *free*.

outchar(c) char c;
Print the character *c* on the standard output. All output to the terminal should go through this routine or *outstr*.

outstr(str) char *str;
Print the string str on the standard output by repeated calls to *outchar*.

printg()
Print the graphics display on the printer. The printer must be configured as device 6 (the default) on the HPIB.

char rawchar()
Read one character from the terminal and return it. This routine or *readline* should be used to get all input, rather than *getchar*(3).

rboff() Turn the rubber band line off.

rbon() Turn the rubber band line on.

char *rdchar(c) char c;
Return a readable representation of the character *c*. If *c* is a printing character it returns itself, if a control character it is shown in the ^X notation, if negative an apostrophe is prepended. Space returns ^`, rubout returns ^?.

NOTE: A pointer to a static place is returned. For this reason, it will not work to pass rdchar twice to the same *fprintf/sprintf* call. You must instead save one of the values in your own buffer with strcpy.

readline(prompt, msg, maxlen) char *prompt, *msg;
Display *prompt* on the bottom line of the graphics display and read one line of text from the user, terminated by a newline. The line is placed in the buffer *msg*, which has size *maxlen* characters. Backspace processing is supported.

setclear()

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

setmat(m, rows, cols, r, c, val) bitmat m;
The basic operation to store a value in an element of a bit matrix. The *[r, c]* element of *m* is set to *val,* which should be either 0 or 1.

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

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

sync() Force all accumulated output to be displayed on the screen. This should be followed by fflush(stdout). The cursor is not affected by this function. Note that it is normally never necessary to call *sync*, since *rawchar* and *readline* call *sync()* and *fflush(stdout)* automatically.

togvid()
Toggle the state of video. If in normal mode, go into inverse video mode, and vice versa. The screen is reversed as well as the internal state of the library.

ttyinit()
Set up the terminal for processing. This routine should be called at the beginning of execution. It places the terminal in CBREAK mode, turns off echo, sets the proper modes in the terminal, and initializes the library.

update(mold, mnew, rows, cols, baser, basec) bitmat mold, mnew;
Make whatever changes are needed to make a window on the screen look like *mnew. Mold* is what the window on the screen currently looks like. The window has size *rows* by *cols*, and the lower left corner on the screen of the window is *[baser, basec].* Note: *update* was not intended to be used for the entire screen. It would work but be very slow and take 64K bytes of memory just for mold and mnew. It was intended for 100 by 100 windows with objects in the center of them, and is quite fast for such windows.

vidinv()
Set inverse video mode.

vidnorm()
Set normal video mode.

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

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

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

zoomon()
Turn zoom on. This restores the screen to the previously specified zoom size.

DIAGNOSTICS

The routine *error* is called when an error is detected. The only error currently detected is overflow of the buffer provided to *readline*.

Subscripts out of bounds to *setmat* return without setting anything.

FILES

/usr/lib/lib2648.a

SEE ALSO
fed(1)

AUTHOR
Mark Horton

BUGS
This library is not supported. It makes no attempt to use all of the features of the terminal, only those needed by fed. Contributions from users will be accepted for addition to the library.

The HP 2648 terminal is somewhat unreliable at speeds over 2400 baud, even with the ^E/^F handshaking. In an effort to improve reliability, handshaking is done every 32 characters. (The manual claims it is only necessary every 80 characters.) Nonetheless, I/O errors sometimes still occur.

There is no way to control the amount of debugging output generated on *trace* without modifying the source to the library.

NAME

plot: openpl, erase, label, line, circle, arc, move, cont, point, linemod, space, closepl — graphics interface

SYNOPSIS

openpl()

erase()

label(s)
char s[];

line(x1, y1, x2, y2)

circle(x, y, r)

arc(x, y, x0, y0, x1, y1)

move(x, y)

cont(x, y)

point(x, y)

linemod(s)
char s[];

space(x0, y0, x1, y1)

closepl()

DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. See *plot*(5) for a description of their effect. *Openpl* must be used before any of the others to open the device for writing. *Closepl* flushes the output.

String arguments to *label* and *linemod* are null-terminated, and do not contain newlines.

Various flavors of these functions exist for different output devices. They are obtained by the following *ld*(1) options:

−lplot	device-independent graphics stream on standard output for *plot*(1) filters
−l300	GSI 300 terminal
−l300s	GSI 300S terminal
−l450	DASI 450 terminal
−l4014	Tektronix 4014 terminal

SEE ALSO

plot(5), plot(1G), graph(1G)

NAME

rcmd, rresvport, ruserok — routines for returning a stream to a remote command

SYNOPSIS

```
rem = rcmd(ahost, inport, locuser, remuser, cmd, fd2p);
char **ahost;
u_short inport;
char *locuser, *remuser, *cmd;
int *fd2p;

s = rresvport(port);
int *port;

ruserok(rhost, superuser, ruser, luser);
char *rhost;
int superuser;
char *ruser, *luser;
```

DESCRIPTION

Rcmd is a routine used by the super-user to execute a command on a remote machine using an authentication scheme based on reserved port numbers. *Rresvport* is a routine which returns a descriptor to a socket with an address in the privileged port space. *Ruserok* is a routine used by servers to authenticate clients requesting service with *rcmd.* All three functions are present in the same file and are used by the *rshd*(8C) server (among others).

Rcmd looks up the host **ahost* using *gethostbyname*(3N), returning −1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port *inport.*

If the 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 non-zero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in **fd2p.* The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as being UNIX signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the **stderr** (unit 2 of the remote command) will be made the same as the **stdout** and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

The protocol is described in detail in *rshd*(8C).

The *rresvport* routine is used to obtain a socket with a privileged address bound to it. This socket is suitable for use by *rcmd* and sevral other routines. Privileged addresses consist of a port in the range 0 to 1023. Only the super-user is allowed to bind an address of this sort to a socket.

Ruserok takes a remote host's name, as returned by a *gethostent*(3N) routine, two user names and a flag indicating if the local user's name is the super-user. It then checks the files */etc/hosts.equiv* and, possibly, *.rhosts* in the current working directory (normally the local user's home directory) to see if the request for service is allowed. A 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

rlogin(1C), rsh(1C), rexec(3X), rexecd(8C), rlogind(8C), rshd(8C)

BUGS

There is no way to specify options to the *socket* call which *rcmd* makes.

NAME

rexec − return stream to a remote command

SYNOPSIS

rem = 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 *gethostbyname*(3N), returning −1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host. If a username and password are both specified, then these are used to authenticate to the foreign host; otherwise the environment and then the user's *.netrc* file in his home directory are searched for appropriate information. If all this fails, the user is prompted for the information.

The port *inport* specifies which well-known DARPA Internet port to use for the connection; 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*(8C).

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 non-zero, 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 UNIX signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the **stderr** (unit 2 of the remote command) will be made the same as the **stdout** and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

SEE ALSO

rcmd(3X), rexecd(8C)

BUGS

There is no way to specify options to the *socket* call which *rexec* makes.

NAME

tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs — terminal independent operation routines

SYNOPSIS

```
char PC;
char *BC;
char *UP;
short ospeed;

tgetent(bp, name)
char *bp, *name;

tgetnum(id)
char *id;

tgetflag(id)
char *id;

char *
tgetstr(id, area)
char *id, **area;

char *
tgoto(cm, destcol, destline)
char *cm;

tputs(cp, affcnt, outc)
register char *cp;
int affcnt;
int (*outc)();
```

DESCRIPTION

These functions extract and use capabilities from the terminal capability data base *termcap*(5). These are low level routines; see *curses*(3X) for a higher level package.

Tgetent extracts the entry for terminal *name* into the buffer at *bp*. *Bp* should be a character buffer of size 1024 and must be retained through all subsequent calls to *tgetnum*, *tgetflag*, and *tgetstr*. *Tgetent* returns −1 if it cannot open the *termcap* file, 0 if the terminal name given does not have an entry, and 1 if all goes well. It will look in the environment for a TERMCAP variable. If found, and the value does not begin with a slash, and the terminal type **name** is the same as the environment string TERM, the TERMCAP string is used instead of reading the termcap file. If it does begin with a slash, the string is used as a path name rather than */etc/termcap*. This can speed up entry into programs that call *tgetent*, as well as to help debug new terminal descriptions or to make one for your terminal if you can't write the file */etc/termcap*.

Tgetnum gets the numeric value of capability *id*, returning −1 if is not given for the terminal. *Tgetflag* returns 1 if the specified capability is present in the terminal's entry, 0 if it is not. *Tgetstr* 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*(5), 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 which call tgoto should be sure to turn off the XTABS bit(s), since *tgoto* may now output a tab. Note that programs using termcap should in general turn off XTABS anyway since some terminals use control I for other functions, such as nondestructive space.) If a **%** sequence is given which is not understood, then *tgoto* returns "OOPS".

Tputs decodes the leading padding information of the string *cp*; *affcnt* gives the number of lines affected by the operation, or 1 if this is not applicable, *outc* is a routine which is called with each character in turn. The external variable *ospeed* should contain the output speed of the terminal as encoded by *stty*(3). The external variable **PC** should contain a pad character to be used (from the **pc** capability) if a null (^@) is inappropriate.

FILES

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

SEE ALSO

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

AUTHOR

William Joy

NAME

intro − introduction to compatibility library functions

DESCRIPTION

These functions constitute the compatibility library portion of *libc*. They are automatically loaded as needed by the C compiler *cc*(1). The link editor searches this library under the "−lc" option. Use of these routines should, for the most part, be avoided. Manual entries for the functions in this library describe the proper routine to use.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
alarm	alarm.3c	schedule signal after specified time
ftime	time.3c	get date and time
getpw	getpw.3c	get name from uid
gtty	stty.3c	set and get terminal state (defunct)
nice	nice.3c	set program priority
pause	pause.3c	stop until signal
rand	rand.3c	random number generator
signal	signal.3c	simplified software signal facilities
srand	rand.3c	random number generator
stty	stty.3c	set and get terminal state (defunct)
time	time.3c	get date and time
times	times.3c	get process times
utime	utime.3c	set file times
vlimit	vlimit.3c	control maximum system resource consumption
vtimes	vtimes.3c	get information about resource utilization

NAME

alarm − schedule signal after specified time

SYNOPSIS

alarm(seconds)
unsigned seconds;

DESCRIPTION

This interface is obsoleted by setitimer(2).

Alarm causes signal SIGALRM, see *signal*(3C), to be sent to the invoking process in a number of seconds given by the argument. Unless caught or ignored, the signal terminates the process.

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is 0, any alarm request is canceled. Because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is 2147483647 seconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO

sigpause(2), sigvec(2), signal(3C), sleep(3)

NAME

getpw − get name from uid

SYNOPSIS

getpw(uid, buf)
char *buf;

DESCRIPTION

Getpw is obsoleted by getpwuid(3).

Getpw searches the password file for the (numerical) *uid*, and fills in *buf* with the corresponding line; it returns non-zero if *uid* could not be found. The line is null-terminated.

FILES

/etc/passwd

SEE ALSO

getpwent(3), passwd(5)

DIAGNOSTICS

Non-zero return on error.

NAME

nice − set program priority

SYNOPSIS

nice(incr)

DESCRIPTION

This interface is obsoleted by setpriority(2).

The scheduling priority of the process is augmented by *incr*. Positive priorities get less service than normal. Priority 10 is recommended to users who wish to execute long-running programs without flak from the administration.

Negative increments are ignored except on behalf of the super-user. The priority is limited to the range −20 (most urgent) to 20 (least).

The priority of a process is passed to a child process by *fork*(2). For a privileged process to return to normal priority from an unknown state, *nice* should be called successively with arguments −40 (goes to priority −20 because of truncation), 20 (to get to 0), then 0 (to maintain compatibility with previous versions of this call).

SEE ALSO

nice(1), setpriority(2), fork(2), renice(8)

NAME

pause — stop until signal

SYNOPSIS

pause()

DESCRIPTION

Pause never returns normally. It is used to give up control while waiting for a signal from *kill*(2) or an interval timer, see *setitimer*(2). Upon termination of a signal handler started during a *pause,* the *pause* call will return.

RETURN VALUE

Always returns −1.

ERRORS

Pause always returns:

[EINTR]	The call was interrupted.

SEE ALSO

kill(2), select(2), sigpause(2)

NAME

rand, srand — random number generator

SYNOPSIS

```
srand(seed)
int seed;

rand()
```

DESCRIPTION

The newer random(3) should be used in new applications; rand remains for compatibilty.

Rand uses a multiplicative congruential random number generator with period 2^{32} to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$.

The generator is reinitialized by calling *srand* with 1 as argument. It can be set to a random starting point by calling *srand* with whatever you like as argument.

SEE ALSO

random(3)

NAME

signal — simplified software signal facilities

SYNOPSIS

#include <signal.h>

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

DESCRIPTION

Signal is a simplified interface to the more general *sigvec*(2) facility.

A signal is generated by some abnormal event, initiated by a user at a terminal (quit, interrupt, stop), by a program error (bus error, etc.), by request of another program (kill), or when a process is stopped because it wishes to access its control terminal while in the background (see *tty*(4)). Signals are optionally generated when a process resumes after being stopped, when the status of child processes changes, or when input is ready at the control terminal. Most signals cause termination of the receiving process if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the SIGKILL and SIGSTOP signals, the *signal* call allows signals either to be ignored or to cause an interrupt to a specified location. The following is a list of all signals with names as in the include file <*signal.h*>:

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction
SIGTRAP	5*	trace trap
SIGIOT	6*	IOT instruction
SIGEMT	7*	EMT instruction
SIGFPE	8*	floating point exception
SIGKILL	9	kill (cannot be caught or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGURG	16●	urgent condition present on socket
SIGSTOP	17†	stop (cannot be caught or ignored)
SIGTSTP	18†	stop signal generated from keyboard
SIGCONT	19●	continue after stop
SIGCHLD	20●	child status has changed
SIGTTIN	21†	background read attempted from control terminal
SIGTTOU	22†	background write attempted to control terminal
SIGIO	23●	i/o is possible on a descriptor (see *fcntl*(2))
SIGXCPU	24	cpu time limit exceeded (see *setrlimit*(2))
SIGXFSZ	25	file size limit exceeded (see *setrlimit*(2))
SIGVTALRM	26	virtual time alarm (see *setitimer*(2))
SIGPROF	27	profiling timer alarm (see *setitimer*(2))

The starred signals in the list above cause a core image if not caught or ignored.

If *func* is SIG_DFL, the default action for signal *sig* is reinstated; this default is termination (with a core image for starred signals) except for signals marked with ● or †. Signals marked with ● are discarded if the action is SIG_DFL; signals marked with † cause the process to stop. If *func* is SIG_IGN the signal is subsequently ignored and pending instances of the signal are discarded. Otherwise, when the signal occurs further occurences of the signal are automatically blocked and *func* is called.

A return from the function unblocks the handled signal and continues the process at the point it was interrupted. **Unlike previous signal facilities, the handler *func* remains installed after a signal has been delivered.**

If a caught signal occurs during certain system calls, causing the call to terminate prematurely, the call is automatically restarted. In particular this can occur during a *read* or *write*(2) on a slow device (such as a terminal; but not a file) and during a *wait*(2).

The value of *signal* is the previous (or initial) value of *func* for the particular signal.

After a *fork*(2) or *vfork*(2) the child inherits all signals. *Execve*(2) resets all caught signals to the default action; ignored signals remain ignored.

RETURN VALUE

The previous action is returned on a successful call. Otherwise, −1 is returned and *errno* is set to indicate the error.

ERRORS

Signal will fail and no action will take place if one of the following occur:

[EINVAL]	*Sig* is not a valid signal number.
[EINVAL]	An attempt is made to ignore or supply a handler for SIGKILL or SIGSTOP.
[EINVAL]	An attempt is made to ignore SIGCONT (by default SIGCONT is ignored).

SEE ALSO

kill(1), ptrace(2), kill(2), sigvec(2), sigblock(2), sigsetmask(2), sigpause(2), sigstack(2), setjmp(3), tty(4)

NOTES (VAX-11)

The handler routine can be declared:

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

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

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

Hardware condition	Signal	Code
Arithmetic traps:		
Integer overflow	SIGFPE	FPE_INTOVF_TRAP
Integer division by zero	SIGFPE	FPE_INTDIV_TRAP
Floating overflow trap	SIGFPE	FPE_FLTOVF_TRAP
Floating/decimal division by zero	SIGFPE	FPE_FLTDIV_TRAP
Floating underflow trap	SIGFPE	FPE_FLTUND_TRAP
Decimal overflow trap	SIGFPE	FPE_DECOVF_TRAP
Subscript-range	SIGFPE	FPE_SUBRNG_TRAP
Floating overflow fault	SIGFPE	FPE_FLTOVF_FAULT
Floating divide by zero fault	SIGFPE	FPE_FLTDIV_FAULT
Floating underflow fault	SIGFPE	FPE_FLTUND_FAULT
Length access control	SIGSEGV	
Protection violation	SIGBUS	

Reserved instruction	SIGILL	ILL_RESAD_FAULT
Customer-reserved instr.	SIGEMT	
Reserved operand	SIGILL	ILL_PRIVIN_FAULT
Reserved addressing	SIGILL	ILL_RESOP_FAULT
Trace pending	SIGTRAP	
Bpt instruction	SIGTRAP	
Compatibility-mode	SIGILL	hardware supplied code
Chme	SIGSEGV	
Chms	SIGSEGV	
Chmu	SIGSEGV	

NAME

stty, gtty — set and get terminal state (defunct)

SYNOPSIS

```
#include <sgtty.h>

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

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

DESCRIPTION

This interface is obsoleted by ioctl(2).

Stty sets the state of the terminal associated with *fd*. *Gtty* retrieves the state of the terminal associated with *fd*. To set the state of a terminal the call must have write permission.

The *stty* call is actually "ioctl(fd, TIOCSETP, buf)", while the *gtty* call is "ioctl(fd, TIOCGETP, buf)". See *ioctl*(2) and *tty*(4) for an explanation.

DIAGNOSTICS

If the call is successful 0 is returned, otherwise −1 is returned and the global variable *errno* contains the reason for the failure.

SEE ALSO

ioctl(2), tty(4)

NAME

time, ftime – get date and time

SYNOPSIS

```
long time(0)

long time(tloc)
long *tloc;

#include <sys/types.h>
#include <sys/timeb.h>
ftime(tp)
struct timeb *tp;
```

DESCRIPTION

These interfaces are obsoleted by gettimeofday(2).

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

If *tloc* is nonnull, the return value is also stored in the place to which *tloc* points.

The *ftime* entry fills in a structure pointed to by its argument, as defined by <*sys/timeb.h*>:

```
/*	timeb.h	6.183/07/29*/

/*
 * Structure returned by ftime system call
 */
struct timeb
{
	time_t	time;
	unsigned short millitm;
	short	timezone;
	short	dstflag;
};
```

The structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-precise interval, the local time zone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time applies locally during the appropriate part of the year.

SEE ALSO

date(1), gettimeofday(2), settimeofday(2), ctime(3)

NAME

times − get process times

SYNOPSIS

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

times (buffer)
struct tms *buffer;

DESCRIPTION

This interface is obsoleted by getrusage(2).

Times returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/HZ seconds, where HZ is 60.

This is the structure returned by *times*:

```
/*      times.h 6.1     83/07/29        */

/*
 * Structure returned by times()
 */
struct tms {
        time_t  tms_utime;              /* user time */
        time_t  tms_stime;              /* system time */
        time_t  tms_cutime;             /* user time, children */
        time_t  tms_cstime;             /* system time, children */
};
```

The children times are the sum of the children's process times and their children's times.

SEE ALSO

time(1), getrusage(2), wait3(2), time(3)

NAME

utime − set file times

SYNOPSIS

#include <sys/types.h>

utime(file, timep)
char *file;
time_t timep[2];

DESCRIPTION

This interface is obsoleted by utimes(2).

The *utime* call uses the 'accessed' and 'updated' times in that order from the *timep* vector to set the corresponding recorded times for *file.*

The caller must be the owner of the file or the super-user. The 'inode-changed' time of the file is set to the current time.

SEE ALSO

utimes(2), stat(2)

NAME

vlimit − control maximum system resource consumption

SYNOPSIS

#include <sys/vlimit.h>

vlimit(resource, value)

DESCRIPTION

This facility is superseded by getrlimit(2).

Limits the consumption by the current process and each process it creates to not individually exceed *value* on the specified *resource.* If *value* is specified as −1, then the current limit is returned and the limit is unchanged. The resources which are currently controllable are:

LIM_NORAISE	A pseudo-limit; if set non-zero then the limits may not be raised. Only the super-user may remove the *noraise* restriction.
LIM_CPU	the maximum number of cpu-seconds to be used by each process
LIM_FSIZE	the largest single file which can be created
LIM_DATA	the maximum growth of the data+stack region via *sbrk*(2) beyond the end of the program text
LIM_STACK	the maximum size of the automatically-extended stack region
LIM_CORE	the size of the largest core dump that will be created.
LIM_MAXRSS	a soft limit for the amount of physical memory (in bytes) to be given to the program. If memory is tight, the system will prefer to take memory from processes which are exceeding their declared LIM_MAXRSS.

Because this information is stored in the per-process information this system call must be executed directly by the shell if it is to affect all future processes created by the shell; *limit* is thus a built-in command to *csh*(1).

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way; a *break* call fails if the data space limit is reached, or the process is killed when the stack limit is reached (since the stack cannot be extended, there is no way to send a signal!).

A file i/o operation which would create a file which is too large will cause a signal SIGXFSZ to be generated, this normally terminates the process, but may be caught. When the cpu time limit is exceeded, a signal SIGXCPU is sent to the offending process; to allow it time to process the signal it is given 5 seconds grace by raising the cpu time limit.

SEE ALSO

csh(1)

BUGS

If LIM_NORAISE is set, then no grace should be given when the cpu time limit is exceeded.

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

This call is peculiar to this version of UNIX. The options and specifications of this system call and even the call itself are subject to change. It may be extended or replaced by other facilities in future versions of the system.

NAME

vtimes — get information about resource utilization

SYNOPSIS

vtimes(par_vm, ch_vm)
struct vtimes *par_vm, *ch_vm;

DESCRIPTION

This facility is superseded by getrusage(2).

Vtimes returns accounting information for the current process and for the terminated child processes of the current process. Either *par_vm* or *ch_vm* or both may be 0, in which case only the information for the pointers which are non-zero is returned.

After the call, each buffer contains information as defined by the contents of the include file */usr/include/sys/vtimes.h:*

```
struct vtimes {
        int     vm_utime;           /* user time (*HZ) */
        int     vm_stime;           /* system time (*HZ) */
        /* divide next two by utime+stime to get averages */
        unsigned vm_idsrss;         /* integral of d+s rss */
        unsigned vm_ixrss;          /* integral of text rss */
        int     vm_maxrss;          /* maximum rss */
        int     vm_majflt;          /* major page faults */
        int     vm_minflt;          /* minor page faults */
        int     vm_nswap;           /* number of swaps */
        int     vm_inblk;           /* block reads */
        int     vm_oublk;           /* block writes */
};
```

The *vm_utime* and *vm_stime* fields give the user and system time respectively in 60ths of a second (or 50ths if that is the frequency of wall current in your locality.) The *vm_idrss* and *vm_ixrss* measure memory usage. They are computed by integrating the number of memory pages in use each over cpu time. They are reported as though computed discretely, adding the current memory usage (in 512 byte pages) each time the clock ticks. If a process used 5 core pages over 1 cpu-second for its data and stack, then *vm_idsrss* would have the value 5*60, where *vm_utime+vm_stime* would be the 60. *Vm_idsrss* integrates data and stack segment usage, while *vm_ixrss* integrates text segment usage. *Vm_maxrss* reports the maximum instantaneous sum of the text+data+stack core-resident page count.

The *vm_majflt* field gives the number of page faults which resulted in disk activity; the *vm_minflt* field gives the number of page faults incurred in simulation of reference bits; *vm_nswap* is the number of swaps which occurred. The number of file system input/output events are reported in *vm_inblk* and *vm_oublk* These numbers account only for real i/o; data supplied by the caching mechanism is charged only to the first process to read or write the data.

SEE ALSO

time(2), wait3(2)

BUGS

This call is peculiar to this version of UNIX. The options and specifications of this system call are subject to change. It may be extended to include additional information in future versions of the system.

NAME

intro − introduction to special files and hardware support

DESCRIPTION

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

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

VAX DEVICE SUPPORT

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

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

The autoconfiguration system is described in *autoconf*(4). VAX specific device support is described in "4V" entries. A list of the supported devices is given below.

SEE ALSO

intro(4), intro(4N), autoconf(4), config(8)

LIST OF DEVICES

The devices listed below are supported in this incarnation of the system. Devices are indicated by their functional interface. If second vendor products provide functionally identical interfaces they should be usable with the supplied software. **(Beware however that we promise the software works ONLY with the hardware indicated on the appropriate manual page.)**

acc	ACC LH/DH IMP communications interface
ad	Data translation A/D interface
css	DEC IMP-11A communications interface
ct	C/A/T phototypesetter
dh	DH-11 emulators, terminal multiplexor
dmc	DEC DMC-11/DMR-11 point-to-point communications device
dmf	DEC DMF-32 terminal multiplexor
dn	DEC DN-11 autodialer interface
dz	DZ-11 terminal multiplexor
ec	3Com 10Mb/s Ethernet controller
en	Xerox 3Mb/s Ethernet controller (obsolete)
kg	KL-11/DL-11W line clock
fl	VAX-11/780 console floppy interface
hk	RK6-11/RK06 and RK07 moving head disk

hp	MASSBUS disk interface (with RP06, RM03, RM05, etc.)
ht	TM03 MASSBUS tape drive interface (with TE-16, TU-45, TU-77)
hy	DR-11B or GI-13 interface to an NSC Hyperchannel
ik	Ikonas frame buffer graphics device interface
il	Interlan 10Mb/s Ethernet controller
lp	LP-11 parallel line printer interface
mt	TM78 MASSBUS tape drive interface
pcl	DEC PCL-11 communications interface
ps	Evans and Sutherland Picture System 2 graphics interface
rx	DEC RX02 floppy interface
tm	TM-11/TE-10 tape drive interface
ts	TS-11 tape drive interface
tu	VAX-11/730 TU58 console cassette interface
uda	DEC UDA-50 disk controller
un	DR-11W interface to Ungermann-Bass
up	Emulex SC-21V UNIBUS disk controller
ut	UNIBUS TU-45 tape drive interface
uu	TU58 dual cassette drive interface (DL11)
va	Benson-Varian printer/plotter interface
vp	Versatec printer/plotter interface
vv	Proteon proNET 10Mb/s ring network interface

NAME
networking – introduction to networking facilities

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

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

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

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

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

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 */
```

ROUTING

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

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

```
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 non-zero), 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 non-existant 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*(4), 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

```
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;
#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 */
};
```

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).
 */
struct  ifconf {
        int     ifc_len;        /* size of associated buffer */
        union {
                caddr_t ifcu_buf;
                struct  ifreq *ifcu_req;
        } ifc_ifcu;
#define ifc_buf ifc_ifcu.ifcu_buf       /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req/* array of structures returned */
};
```

SEE ALSO

socket(2), ioctl(2), intro(4), config(8), routed(8C)

NAME

acc – ACC LH/DH IMP interface

SYNOPSIS

pseudo-device imp
device acc0 at uba0 csr 167600 vector accrint accxint

DESCRIPTION

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

DIAGNOSTICS

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

acc%d: can't initialize. Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).

acc%d: imp doesn't respond, icsr=%b. The driver attempted to initialize the device, but the IMP failed to respond after 500 tries. Check the cabling.

acc%d: stray xmit interrupt, csr=%b. An interrupt occurred when no output had previously been started.

acc%d: output error, ocsr=%b, icsr=%b. The device indicated a problem sending data on output.

acc%d: input error, csr=%b. The device indicated a problem receiving data on input.

acc%d: bad length=%d. An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never happen as the maximum size of a host-IMP message is 1008 bytes.

NAME

ad − Data Translation A/D converter

SYNOPSIS

device ad0 at uba0 csr 0170400 vector adintr

DESCRIPTION

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

The driver communicates to a user process by means of ioctls. The AD_CHAN ioctl selects which channel of the board to read. For example,

```
        chan = 5; ioctl(fd, AD_CHAN, &chan);
```

selects channel 5. The AD_READ ioctl actually reads the data and returns it to the user process. An example is

```
        ioctl(fd, AD_READ, &data);
```

FILES

/dev/ad

DIAGNOSTICS

None.

NAME

arp — Address Resolution Protocol

SYNOPSIS

pseudo-device ether

DESCRIPTION

ARP is a protocol used to dynamically map between DARPA Internet and 10Mb/s Ethernet addresses. It is used by all the 10Mb/s Ethernet interface drivers.

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 will queue at most one packet while waiting for a mapping request to be responded to; only the most recently "transmitted" packet is kept.

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

```
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
struct arpreq arpreq;

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

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

```
/*
 * ARP ioctl request
 */
struct arpreq {
        struct sockaddr  arp_pa;          /* protocol address */
        struct sockaddr  arp_ha;          /* hardware address */
        int        arp_flags;             /* flags */
};
/*  arp_flags field values */
#define ATF_COM              2        /* completed entry (arp_ha valid) */
#defineATF_PERM     4        /* permanent entry */
#defineATF_PUBL     8        /* publish (respond for other host) */
```

The address family for the *arp_pa* sockaddr must be AF_INET; for the *arp_ha* sockaddr it must be AF_UNSPEC. The only flag bits which may be written are ATF_PERM and ATF_PUBL. ATF_PERM causes the entry to be permanent if the ioctl call succeeds. The peculiar nature of the ARP tables may cause the ioctl to fail if more than 4 (permanent) Internet host addresses hash to the same slot. ATF_PUBL specifies that the ARP code should respond to ARP requests for the indicated host coming from other machines. This allows a host to act as an "ARP server" which may be useful in convincing an ARP-only machine to talk to a non-ARP machine.

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

DIAGNOSTICS

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

SEE ALSO

ec(4), de(4), il(4), inet(4F), arp(8C), ifconfig(8C)
An Ethernet Address Resolution Protocol, RFC826, Dave Plummer, MIT.

BUGS

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

NAME

autoconf − diagnostics from the autoconfiguration code

DESCRIPTION

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

Devices in NEXUS slots are normally noted, thus memory controllers, UNIBUS and MASSBUS adaptors. Devices which are not supported which are found in NEXUS slots are noted also.

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

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

A generic system may be built which picks its root device at boot time as the "best" available device (MASSBUS disks are better than SMD UNIBUS disks are better than RK07's; the device must be drive 0 to be considered.) If such a system is booted with the RB_ASKNAME option of (see *reboot*(2)), then the name of the root device is read from the console terminal at boot time, and any available device may be used.

SEE ALSO

intro(4), config(8)

DIAGNOSTICS

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

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

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

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

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

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

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

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

%s%d at mba%d drive %d. A tape formatter or a disk was found on the MASSBUS; for disks %s%d will look like "hp0", for tape formatters like "ht1". The drive number comes from the unit plug on the drive or in the TM formatter (**not** on the tape drive; see below).

%s%d at %s%d slave %d. (For MASSBUS devices). Which would look like "tu0 at ht0 slave 0", where **tu0** is the name for the tape device and **ht0** is the name for the formatter. A tape slave was found on the tape formatter at the indicated drive number (on the front of the tape drive). UNIX will call the device, e.g., **tu0**.

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

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

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

%s%d at %s%d slave %d. (For UNIBUS devices). Which would look like "up0 at sc0 slave 0", where **up0** is the name of a disk drive and **sc0** is the name of the controller. Analogous to MASSBUS case.

NAME

bk – line discipline for machine-machine communication (obsolete)

SYNOPSIS

pseudo-device bk

DESCRIPTION

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

The line discipline is enabled by a sequence:

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

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

The old standard teletype discipline can be restored by doing:

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

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

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

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

SEE ALSO

tty(4)

DIAGNOSTICS

None.

BUGS

The Purdue uploading line discipline, which provides 8 bits and uses timeout's to terminate uploading should be incorporated into the standard system, as it is much more suitable for microprocessor connections.

NAME

cons − VAX-11 console interface

DESCRIPTION

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

On an 11/780 you can return to the conversational mode using the command "set t p" (set terminal program) if the processor is still running or "continue" if it is halted. The latter command may be abbreviated "c". If you hit the break key on the console, then the console will go into ODT (console debugger mode). Hit a "P" (upper-case letter p) to get out of this mode.

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

With the above proviso's the console works like any other UNIX terminal.

FILES

/dev/console

SEE ALSO

tty(4), reboot(8)
VAX Hardware Handbook

DIAGNOSTICS

None.

NAME

css − DEC IMP-11A LH/DH IMP interface

SYNOPSIS

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

DESCRIPTION

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

DIAGNOSTICS

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

css%d: can't initialize. Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).

css%d: imp doesn't respond, icsr=%b. The driver attempted to initialize the device, but the IMP failed to respond after 500 tries. Check the cabling.

css%d: stray output interrupt csr=%b. An interrupt occurred when no output had previously been started.

css%d: output error, ocsr=%b icsr=%b. The device indicated a problem sending data on output.

css%d: recv error, csr=%b. The device indicated a problem receiving data on input.

css%d: bad length=%d. An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never happen as the maximum size of a host-IMP message is 1008 bytes.

NAME

ct − phototypesetter interface

SYNOPSIS

device ct0 at uba0 csr 0167760 vector ctintr

DESCRIPTION

This provides an interface to a Graphic Systems C/A/T phototypesetter. Bytes written on the file specify font, size, and other control information as well as the characters to be flashed. The coding is not described here.

Only one process may have this file open at a time. It is write-only.

FILES

/dev/cat

SEE ALSO

troff(1)
Phototypesetter interface specification

DIAGNOSTICS

None.

NAME

de – DEC DEUNA 10 Mb/s Ethernet interface

SYNOPSIS

device de0 at uba0 csr 0174510 vector deintr

DESCRIPTION

The *de* interface provides access to a 10 Mb/s Ethernet network through a DEUNA controller.

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

The interface normally tries to use a "trailer" encapsulation to minimize copying data on input and output. This may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS

de%d: command failed, csr0=%b csr1=%b. Here **command** is one of **reset**, **pcbb**, **rdphyad**, **wtring**, or **wtmode**. This message is printed if there is an error on device initialization.

de%d: buffer unavailable. Packets are being received by the interface faster than they can be serviced by the driver.

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

SEE ALSO

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

BUGS

The PUP protocol family should be added.

NAME

dh − DH-11/DM-11 communications multiplexer

SYNOPSIS

device dh0 at uba0 csr 0160020 vector dhrint dhxint
device dm0 at uba0 csr 0170500 vector dmintr

DESCRIPTION

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

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

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

The dh driver normally uses input silos and polls for input at each clock tick (10 milliseconds) rather than taking an interrupt on each input character.

FILES

/dev/tty[hi][0-9a-f]
/dev/ttyd[0-9a-f]

SEE ALSO

tty(4)

DIAGNOSTICS

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

dh%d: silo overflow. The character input silo overflowed before it could be serviced. This can happen if a hard error occurs when the CPU is running with elevated priority, as the system will then print a message on the console with interrupts disabled. If the Berknet is running on a *dh* line at high speed (e.g. 9600 baud), there is only 1/15th of a second of buffering capacity in the silo, and overrun is possible. This may cause a few input characters to be lost to users and a network packet is likely to be corrupted, but the network will recover. It is not serious.

NAME

dmc − DEC DMC-11/DMR-11 point-to-point communications device

SYNOPSIS

device dmc0 at uba0 csr 167600 vector dmcrint dmcxint

DESCRIPTION

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

The *dmc* interface driver also supports a DEC DMR-11 providing point-to-point communication running at data rates from 2.4 Kb/s to 1 Mb/s. DMR-11's are a more recent design and thus are preferred over DMC-11's.

The host's address must be specified with an SIOCSIFADDR ioctl before the interface will transmit or recive any packets.

DIAGNOSTICS

dmc%d: bad control %o. A bad parameter was passed to the *dmcload* routine.

dmc%d: unknown address type %d. An input packet was received which contained a type of address unknown to the driver.

DMC FATAL ERROR 0%o.

DMC SOFT ERROR 0%o.

dmc%d: af%d not supported. The interface was handed a message which has addresses formatted in an unsuitable address family.

SEE ALSO

intro(4N), inet(4F)

BUGS

Should allow multiple outstanding DMA requests, but due to the design of the current UNIBUS support routines this is very difficult.

NAME

dmf − DMF-32, terminal multiplexor

SYNOPSIS

device dmf0 at uba? csr 0170000
vector dmfsrint dmfsxint dmfdaint dmfdbint dmfrint dmfxint dmflint

DESCRIPTION

The *dmf* device provides 8 lines of asynchronous serial line support with full modem control (the DMF-32 provides other services, but these are not supported by the driver).

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

Bit *i* of flags may be specified for a *dmf* to to say that a line is not properly connected, and that the line should be treated as hard-wired with carrier always present. Thus specifying "flags 0x0004" in the specification of *dmf0* would cause line ttyh2 to be treated in this way.

The *dmf* driver normally uses input silos and polls for input at each clock tick (10 milliseconds).

FILES

/dev/tty[hi][0-9a-f]
/dev/ttyd[0-9a-f]

SEE ALSO

tty(4)

DIAGNOSTICS

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

dmf%d: silo overflow. The character input silo overflowed before it could be serviced. This can happen if a hard error occurs when the CPU is running with elevated priority, as the system will then print a message on the console with interrupts disabled. If the Berknet is running on a *dh* line at high speed (e.g. 9600 baud), there is only 1/15th of a second of buffering capacity in the silo, and overrun is possible. This may cause a few input characters to be lost to users and a network packet is likely to be corrupted, but the network will recover. It is not serious.

dmfsrint.
dmfsxint.
dmfdaint.
dmfdbint.
dmflint.
One of the unsupported parts of the dmf interrupted; something is amiss, check your interrupt vectors for a conflict with another device.

NAME

dn − DN-11 autocall unit interface

SYNOPSIS

device dn0 at uba? csr 0160020 vector dnintr

DESCRIPTION

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

The codes for the phone numbers are:

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

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

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

FILES

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

SEE ALSO

tip(1C)

DIAGNOSTICS

Two error numbers are of interest at open time.

[EBUSY] The dialer is in use.

[ENXIO] The device doesn't exist, or there's no power to it.

NAME

dr − DR11-B/DR11-W interface

SYNOPSIS

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

DESCRIPTION

A variety of user devices may be connected to the system via DR11-B or DR11-W general purpose DMA interfaces. Data written to these devices is transferred directly from the process's buffer to the user device and data read from these devices is supplied directly to the process's buffer by the user device. Details concerning individual features of particular user device interfaces can be found in the description of the relevant user device interfaces (see *dcb*(4) and *gmr*(4)).

IOCTL CALLS

Several *ioctl* calls are available for all DR11-B/DR11-W interfaced user devices unless otherwise specified in the description of that particular device.

The call

```
ioctl(fildes, DRIOCGETR, &drreg)
struct drreg drreg;
```

returns the contents of the four DR11-B (five DR11-W) device registers in the supplied structure. This structure is defined in <*sys/dr.h*> as:

```
struct drreg
{
     short    drr_wc;          /* word count register */
     u_short drr_ba;           /* bus address register */
     u_short drr_st;           /* status and command register */
     u_short drr_db;           /* data buffer register */
     u_short drr_er;           /* error information register (always 0 for DR11-B) */
};
```

Two additional calls provide a mechanism for selectively setting the contents of certain of these device registers. They are:

```
ioctl(fildes, DRIOCSSTR, &stbuf)
u_short stbuf;
```

which sets the contents of the status and command register to the supplied value (currently only the three function bits may be affected by this call), and

```
ioctl(fildes, DRIOCSDBR, &dbbuf)
u_short dbbuf;
```

which sets the contents of the data buffer register to the supplied value.

An additional call

```
ioctl(fildes, DRIOCENBS, &sign)
int sign;
```

enables the specified signal number to be sent to the user process issuing the call if an ATTN interrupt is generated by the DR11 device while no I/O operation is currently in progress. This signal will be reset once it is sent and must be specifically renabled each subsequent time it is needed. A signal number of zero disables any signal from being sent if one is currently enabled.

FILES

/dev/???   see individual device description

SEE ALSO

drb(4), gmr(4)

HISTORY

15-Jun-84 D.Sathyanarayanan (ads) at Stanford University
Resturctured to run under 4.2bsd.

30-Mar-82 Mike Accetta (mja) at Carnegie-Mellon University
Created.

NAME

drb — DR11-B/DR11-W general purpose user device interface

SYNOPSIS

#include <sys/types.h>
#include <vaxuba/drb.h>

DESCRIPTION

Devices *drb** provide the interface for general purpose user devices connected to the system via a DR11-B or DR11-W DMA interface. Data written to these devices is transferred directly from the process's buffer to the user device and data read from these devices is supplied directly to the process's buffer by the user device.

By default, the system uses the FNCT1 bit of the status and control register to distinguish between read and write commands issued by the DR11 to the user device (the other two function bits will be zero). A *read*(2) call will clear the bit before beginning the input operation and a *write*(2) call will set the bit before beginning the output operation.

IOCTL CALLS

All *ioctl* calls described in *dr*(4) may be applied to these devices. Two additional *ioctl* calls:

```
ioctl(fildes, DRBIOCGETP, &drbp)
struct drbparam drbp;
```

and

```
ioctl(fildes, DRBIOCSETP, &drbp)
struct drbparam drbp;
```

which, respectively, fetch and set the device parameters to/from the supplied structure are also available. This parameter structure is defined in <*sys/drb.h*> as:

```
struct drbparam
{
     u_short drbp_rcom;          /* command bits for read operation */
     u_short drbp_wcom;          /* command bits for write operation */
     long     drbp_mbz[9];       /* reserved for future expansion */
                                 /* (must be zero) */
};
```

The read and write command bits are set to the default values when the device is opened (as indicated above). Currently, only the three function bits (FNCT1, FNCT2, and FNCT3) and the prime bus cycles bit (CYCLE) may be set with the **DRBIOCSETP** call; all other bits are ignored.

FILES

/dev/drb* generic DR11-B/DR11-W interfaces

SEE ALSO

dr(4), gmr(4)

BUGS

Not Tested under 4.2bsd.

HISTORY

19-Jun-82 D.Sathyanarayanan (ads) at Stanford University
Restructured for 4.2bsd.

30-Mar-82 Mike Accetta (mja) at Carnegie-Mellon University
Created.

NAME

drum − paging device

DESCRIPTION

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

FILES

/dev/drum

BUGS

Reads from the drum are not allowed across the interleaving boundaries. Since these only occur every .5Mbytes or so, and since the system never allocates blocks across the boundary, this is usually not a problem.

NAME
dz − DZ-11 communications multiplexer

SYNOPSIS
device dz0 at uba0 csr 0160100 vector dzrint dzxint

DESCRIPTION
A dz-11 provides 8 communication lines with partial modem control, adequate for UNIX dialup use. Each line attached to the DZ-11 communications multiplexer behaves as described in *tty*(4) and may be set to run at any of 16 speeds; see *tty*(4) for the encoding.

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

The dz driver normally uses its input silos and polls for input at each clock tick (10 milliseconds) rather than taking an interrupt on each input character.

FILES

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

SEE ALSO
tty(4)

DIAGNOSTICS
dz%d: silo overflow. The 64 character input silo overflowed before it could be serviced. This can happen if a hard error occurs when the CPU is running with elevated priority, as the system will then print a message on the console with interrupts disabled. If the Berknet is running on a *dz* line at high speed (e.g. 9600 baud), there is only 1/15th of a second of buffering capacity in the silo, and overrun is possible. This may cause a few input characters to be lost to users and a network packet is likely to be corrupted, but the network will recover. It is not serious.

NAME

ec − 3Com 10 Mb/s Ethernet interface

SYNOPSIS

device ec0 at uba0 csr 161000 vector ecrint eccollide ecxint

DESCRIPTION

The *ec* interface provides access to a 10 Mb/s Ethernet network through a 3com controller.

The hardware has 32 kilobytes of dual-ported memory on the UNIBUS. This memory is used for internal buffering by the board, and the interface code reads the buffer contents directly through the UNIBUS.

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

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

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

The interface normally tries to use a "trailer" encapsulation to minimize copying data on input and output. This may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS

ec%d: send error. After 16 retransmissions using the exponential backoff algorithm described above, the packet was dropped.

ec%d: input error (offset=%d). The hardware indicated an error in reading a packet off the cable or an illegally sized packet. The buffer offset value is printed for debugging purposes.

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

SEE ALSO

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

BUGS

The PUP protocol family should be added.

The hardware is not capable of talking to itself. The software implements local sending and broadcast by sending such packets to the loop interface. This is a kludge.

Backoff delays are done in a software busy loop. This can degrade the system if the network experiences frequent collisions.

NAME

en − Xerox 3 Mb/s Ethernet interface

SYNOPSIS

device en0 at uba0 csr 161000 vector enrint enxint encollide

DESCRIPTION

The *en* interface provides access to a 3 Mb/s Ethernet network. Due to limitations in the hardware, DMA transfers to and from the network must take place in the lower 64K bytes of the UNIBUS address space.

The network number is specified with a SIOCSIFADDR ioctl; the host's address is discovered by probing the on-board Ethernet address register. No packets will be sent or accepted until a network number is supplied.

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

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

The interface handles both Internet and PUP protocol families, with the interface address maintained in Internet format. PUP addresses are converted to Internet addresses by subsituting PUP network and host values for Internet network and local part values.

The interface normally tries to use a "trailer" encapsulation to minimize copying data on input and output. This may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS

en%d: output error. The hardware indicated an error on the previous transmission.

en%d: send error. After 16 retransmissions using the exponential backoff algorithm described above, the packet was dropped.

en%d: input error. The hardware indicated an error in reading a packet off the cable.

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

SEE ALSO

intro(4N), inet(4F)

BUGS

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

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

NAME

enet – ethernet packet filter

DESCRIPTION

The files */dev/enet** provide a raw interface to 3mb and 10mb Ethernets. Packets received that are not used by the kernel (i.e., to support IP, and on some systems XNS, protocols) are available through this mechanism. Associated with each *enet* file is a user settable packet filter which is used to deliver incoming ethernet packets to the appropriate process. Whenever a packet is received from the net, successive packet filters from the list of filters for all open *enet* files are applied to the packet. When a filter accepts the packet, it is placed on the packet input queue of the associated file. If no filters accept the packet, it is discarded. The format of a packet filter is described below.

Each individual *enet* file may be opened by only one process at a time, although the ethernet may effectively be shared by multiple processes, each specifying its own individual filter on a different *enet* file.

Reads from these devices return the next packet from the packet input queue of the appropriate file. If insufficient buffer space to store the entire packet is specified in the read, the packet will be truncated and the trailing contents lost. Writes to these devices transmit packets on the ethernet with each write generating exactly one packet.

This device is an interface to a variety of different "Ethernet" data-link levels:

3mb Ethernet	packets consist of 4 or more bytes with the first byte specifying the source ethernet address, the second byte specifying the destination ethernet address, and the next two bytes specifying the packet type. (Actually, on the network the source and destination addresses are in the opposite order.)
byte-swapping 3mb Ethernet	packets consist of 4 or more bytes with the first byte specifying the source ethernet address, the second byte specifying the destination ethernet address, and the next two bytes specifying the packet type. **Each short word (pair of bytes) is swapped from the network byte order**; this device type is only provided as a concession to backwards-compatibility.
10mb Ethernet	packets consist of 14 or more bytes with the first six bytes specifying the destination ethernet address, the next six bytes the source ethernet address, and the next two bytes specifying the packet type.

The remaining words are interpreted according to the packet type. Note that 16 bit and 32-bit quantities may have to be byteswapped (and possible short-swapped) to be intelligable on a Vax.

IOCTL CALLS

In addition to FIONREAD, ten special *ioctl* calls may be applied to an open *enet* file. The first two set and fetch parameters for the file and are of the form:

```
#include <sys/types.h>
#include <sys/enet.h>
ioctl(fildes, code, param)
struct eniocb *param;
```

where *param* is defined in *<sys/enet.h>* as:

```
struct eniocb
{
        u_char  en_addr;
        u_char  en_maxfilters;
        u_char  en_maxwaiting;
        u_char  en_maxpriority;
        long    en_rtout;
};
```

with the applicable codes being:

EIOCGETP
: Fetch the parameters for this file.

EIOCSETP
: Set the parameters for this file.

On a 3mb ethernet, the address parameter is the ethernet address of the machine. [This field is essentially obsolete, but is included for compatibility with older code.] The maximum filter length parameter indicates the maximum possible packet filter command list length (see EIOCSETF below). The maximum input wait queue size parameter indicates the maximum number of packets which may be queued for an ethernet file at one time (see EIOCSETW below). The maximum priority parameter indicates the highest filter priority which may be set for the file (see EIOCSETF below). The read timeout parameter specifies the number of clock ticks to wait before timing out on a read request and returning an EOF. This parameter is initialized to zero by *open*(2), indicating no timeout. If it is negative, then read requests will return an EOF immediately if there are no packets in the input queue. (Note that all parameters except for the read timeout are read-only and are ignored when changed.)

A somewhat different *ioctl* is used to get device parameters of the ethernet underlying the minor device. It is of the form:

```
#include <sys/types.h>
#include <sys/enet.h>
ioctl(fildes, EIOCDEVP, param)
```

where *param* is defined in <*sys/enet.h*> as:

```
struct endevp {
        u_char  end_dev_type;
        u_char  end_addr_len;
        u_short end_hdr_len;
        u_short end_MTU;
        u_char  end_addr[EN_MAX_ADDR_LEN];
        u_char  end_broadaddr[EN_MAX_ADDR_LEN];
};
```

The fields are as follows:

end_dev_type	Specifies the device type; currently one of ENDT_3MB, ENDT_BS3MB or ENDT_10MB.
end_addr_len	Specifies the address length in bytes (e.g., 1 or 6).
end_hdr_len	Specifies the total header length in bytes (e.g., 4 or 14).
end_MTU	Specifies the maximum packet size, including header, in bytes.
end_addr	The address of this interface; aligned so that the low order byte of the address is the first byte in the array.
end_broadaddr	The hardware destination address for broadcasts on this network.

The next two calls enable and disable the input packet signal mechanism for the file and are of the form:

```
#include <sys/types.h>
#include <sys/enet.h>
ioctl(fildes, code, signp)
u_int *signp;
```

where *signp* is a pointer to a word containing the number of the signal to be sent when an input packet arrives and with the applicable codes being:

EIOCENBS
: Enable the specified signal when an input packet is received for this file. Further signals are automatically disabled whenever a signal is sent to prevent nesting and hence must be specifically re-enabled after processing (but see EIOCMBIS below). When a signal number of 0 is supplied, this call is equivalent to EIOCINHS.

EIOCINHS
: Disable any signal when an input packet is received for this file (the *signp* parameter is ignored). This is the default when the file is first opened.

The next two calls set and clear "mode bits" for the for the file and are of the form:

```
#include <sys/types.h>
#include <sys/enet.h>
ioctl(fildes, code, bits)
u_short *bits;
```

where *bits* is a short work bit-mask specifying which bits to set or clear. Currently, the only bit mask recognized is ENHOLDSIG, which (if set) tells the driver *not* to disable delivering a signal once it has done so. Setting this bit means that you need use EIOCENBS only once. The applicable codes are:

EIOMBIS
: Sets the specified mode bits

EIOCMBIC
: Clears the specified mode bits

Another *ioctl* call is used to set the maximum size of the packet input queue for an open *enet* file. It is of the form:

```
#include <sys/types.h>
#include <sys/enet.h>
ioctl(fildes, EIOCSETW, maxwaitingp)
u_int *maxwaitingp;
```

where *maxwaitingp* is a pointer to a word containing the input queue size to be set. If this is greater than maximum allowable size (see EIOCGETP above), it is set to the maximum, and if it is zero, it is set to a default value.

Another *ioctl* call flushes the queue of incoming packets. It is of the form:

```
#include <sys/types.h>
#include <sys/enet.h>
ioctl(fildes, EIOCFLUSH, 0)
```

The final *ioctl* call is used to set the packet filter for an open *enet* file. It is of the form:

```
#include <sys/types.h>
#include <sys/enet.h>
ioctl(fildes, EIOCSETF, filter)
struct enfilter *filter
```

where enfilter is defined in *<sys/enet.h>* as:

```
struct enfilter
{
        u_char   enf_Priority;
        u_char   enf_FilterLen;
        u_short  enf_Filter[ENMAXFILTERS];
};
```

A packet filter consists of a priority, the filter command list length (in shortwords), and the filter command list itself. Each filter command list specifies a sequence of actions which operate on an internal stack. Each shortword of the command list specifies an action from the set { **ENF_PUSHLIT**, **ENF_PUSHZERO**, **ENF_PUSHWORD+N** } which respectively push the next shortword of the command list, zero, or shortword N of the incoming packet on the stack, and a binary operator from the set { **ENF_EQ**, **ENF_NEQ**, **ENF_LT**, **ENF_LE**, **ENF_GT**, **ENF_GE**, **ENF_AND**, **ENF_OR**, **ENF_XOR** } which then operates on the top two elements of the stack and replaces them with its result. When both an action and operator are specified in the same shortword, the action is performed followed by the operation.

The binary operator can also be from the set { **ENF_COR**, **ENF_CAND**, **ENF_CNOR**, **ENF_CNAND** }. These are "short-circuit" operators, in that they terminate the execution of the filter immediately if the condition they are checking for is found, and continue otherwise. All pop two elements from the stack and compare them for equality; **ENF_CAND** returns false if the result is false; **ENF_COR** returns true if the result is true; **ENF_CNAND** returns true if the result is false; **ENF_CNOR** returns false if the result is true. Unlike the other binary operators, these four do not leave a result on the stack, even if they continue.

The short-circuit operators should be used when possible, to reduce the amount of time spent evaluating filters. When they are used, you should also arrange the order of the tests so that the filter will succeed or fail as soon as possible; for example, checking the Socket field of a Pup packet is more likely to indicate failure than the packet type field.

The special action **ENF_NOPUSH** and the special operator **ENF_NOP** can be used to only perform the binary operation or to only push a value on the stack. Since both are (conveniently) defined to be zero, indicating only an action actually specifies the action followed by **ENF_NOP**, and indicating only an operation actually specifies **ENF_NOPUSH** followed by the operation.

After executing the filter command list, a non-zero value (true) left on top of the stack (or an empty stack) causes the incoming packet to be accepted for the corresponding *enet* file and a zero value (false) causes the packet to be passed through the next packet filter. (If the filter exits as the result of a short-circuit operator, the top-of-stack value is ignored.) Specifying an undefined operation or action in the command list or performing an illegal operation or action (such as pushing a shortword offset past the end of the packet or executing a binary operator with fewer than two shortwords on the stack) causes a filter to reject the packet.

In an attempt to deal with the problem of overlapping and/or conflicting packet filters, the filters for each open *enet* file are ordered by the driver according to their priority (lowest priority is 0, highest is 255). When processing incoming ethernet packets, filters are applied according to their priority (from highest to lowest) and for identical priority values according to their relative "busyness" (the filter that has previously matched the most packets is checked first) until one or more filters accept the packet or all filters reject it and it is discarded.

Filters at a priority of 2 or higher are called "high priority" filters. Once a packet is delivered to one of these "high priority" *enet* files, no further filters are examined, i.e. the packet is delivered only to the first *enet* file with a "high priority" filter which accepts the packet. A packet may be delivered to more than one filter with a priority below 2; this might be useful, for example, in building replicated programs. However, the use of low-priority filters imposes an additional cost on the system, as these filters each must be checked against all packets not accepted by a high-priority filter.

The packet filter for an *enet* file is initialized with length 0 at priority 0 by *open*(2), and hence by default accepts all packets which no "high priority" filter is interested in.

Priorities should be assigned so that, in general, the more packets a filter is expected to match, the higher its priority. This will prevent a lot of needless checking of packets against filters that aren't likely to match them.

FILTER EXAMPLES

The following filter would accept all incoming *pup packets* on a 3mb ethernet with Pup types in the range 1-0200:

```
struct enfilter f =
{
        10, 19,                                              /* priority and length */
        ENF_PUSHWORD+1, ENF_PUSHLIT, 2, ENF_EQ,              /* packet type == PUP */
        ENF_PUSHWORD+3, ENF_PUSHLIT, 0xFF00, ENF_AND,        /* mask hi byte */
        ENF_PUSHZERO, ENF_GT,                                /* PupType > 0 */
        ENF_PUSHWORD+3, ENF_PUSHLIT, 0xFF00, ENF_AND,        /* mask hi byte */
        ENF_PUSHLIT, 0100, ENF_LE,                           /* PupType <= 0100 */
        ENF_AND,                                             /* 0 < PupType <= 0100 */
    ENF_AND                                                /* && packet type == PUP */
};
```

Note that shortwords, such as the packet type field, are byte-swapped and so the literals you compare them to must be byte-swapped. Also, although for this example the word offsets are constants, code that must run with either 3mb or 10mb ethernets must use offsets that depend on the device type.

By taking advantage of the ability to specify both an action and operation in each word of the command list, the filter could be abbreviated to:

```
struct enfilter f =
{
        10, 14,                                              /* priority and length */
        ENF_PUSHWORD+1, ENF_PUSHLIT | ENF_EQ, 2,             /* packet type == PUP */
        ENF_PUSHWORD+3, ENF_PUSHLIT | ENF_AND, 0xFF00,       /* mask hi byte */
        ENF_PUSHZERO | ENF_GT,                               /* PupType > 0 */
        ENF_PUSHWORD+3, ENF_PUSHLIT | ENF_AND, 0xFF00,       /* mask hi byte */
        ENF_PUSHLIT | ENF_LE, 0100,                          /* PupType <= 0100 */
        ENF_AND,                                             /* 0 < PupType <= 0100 */
    ENF_AND                                                /* && packet type == PUP */
};
```

A different example shows the use of "short-circuit" operators to create a more efficient filter. This one accepts Pup packets (on a 3Mbit ethernet) with a Socket field of 12345. Note that we check the Socket field before the packet type field, since in most packets the Socket is not likely to match.

```
struct enfilter f =
{
        10, 9,                                                  /* priority and length */
        ENF_PUSHWORD+7, ENF_PUSHLIT | ENF_CAND, 0,              /* Hi word of socket */
        ENF_PUSHWORD+8, ENF_PUSHLIT | ENF_CAND, 12345,          /* Lo word of socket */
        ENF_PUSHWORD+1, ENF_PUSHLIT | ENF_CAND, 2               /* packet type == Pup */
};
```

SEE ALSO

de(4), ec(4), en(4), il(4), enstat(8)

FILES

/dev/enet[a-h]{0,1,2,...,31}

BUGS

The current implementation can only filter on words within the first "mbuf" of the packet; this is around 100 bytes (or 50 words).

Because packets are streams of bytes, yet the filters operate on short words, and standard network byte order is usually opposite from Vax byte order, the relational operators ENF_LT, ENF_LE, ENF_GT, and ENF_GE are not all that useful. Fortunately, they were not often used when the packets were treated as streams of shorts, so this is probably not a severe problem. If this becomes a severe problem, a byte-swapping operator could be added.

HISTORY

18-Oct-84 Jeffrey Mogul at Stanford University
  Added short-circuit operators, changed discussion of priorities to reflect new arrangement.

18-Jan-84 Jeffrey Mogul at Stanford University
  Updated for 4.2BSD (device-independent) version, including documentation of all non-kernel ioctls.

17-Nov-81 Mike Accetta (mja) at Carnegie Mellon University
  Added mention of <sys/types.h> to include examples.

29-Sep-81 Mike Accetta (mja) at Carnegie-Mellon University
  Changed to describe new EIOCSETW and EIOCFLUSH ioctl calls and the new multiple packet queuing features.

12-Nov-80 Mike Accetta (mja) at Carnegie-Mellon University
  Added description of signal mechanism for input packets.

07-Mar-80 Mike Accetta (mja) at Carnegie-Mellon University
  Created.

NAME

fl − console floppy interface

DESCRIPTION

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

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

FILES

/dev/floppy

SEE ALSO

arff(8V)

DIAGNOSTICS

None.

BUGS

Multiple console floppies are not supported.

If a write is given with a count not a multiple of 128 bytes then the trailing portion of the last sector will be zeroed.

NAME

gmr − Grinnell Systems display

DESCRIPTION

Devices *gmr* and *gmr[0-3]* provide the interface to the Grinnell Systems graphics display. A Grinnell device may only be opened by at most one application at a time. Subsequent open attempts while a device is already in use return an error status.

On systems with only one Image Function Video Card, (currently the IUS-Vax and the VLSI-Vax), device *gmr* is used (as in previous implementations) for non-shared access to the Grinnell. On systems with four different Image Function Video Cards (currently the GP-Vax). The devices *gmr[0-3]* are used to allow up to 4 separate application access to the Grinnell at one time.

Bytes written to these devices are transferred to the Grinnell interface to perform control and I/O operations and bytes read from the device are supplied by the Grinnell interface depending on previous control messages. On systems where the Grinnell is shared between concurrent applications, a new set of sharing primitives is provided to permit rational access by multiple applications. On single-usage systems, the sharing primitives may be safely ignored (although they exist and function exactly as on shared systems).

IOCTL CALLS

Several *ioctl* calls are provided to manipulate the Grinnell state and facilitate sharing (all symbols are defined in <*sys/gmr.h*>).

The first two calls are used to modify the internal device state and are:

ioctl(fildes, GMRIOCNOWC, NULL)
ioctl(fildes, GMRIOCWC, NULL)

The **GMRIOCNOWC** call disables the word count check within the device for use with packed-byte writes, and the **GMRIOCWC** call enables it.

Five additional calls are also provided for shared Grinnell use on systems with multiple IFVC's. These sharing primitives provide a very simple allocation/deallocation mechanism based on the assumption that applications will not allocate resources without needing them and will not use resources which have not been allocated. They do not provide any level of regulation of Grinnell access beyond what the driver can determine from the state of its own data structures (in particular the driver does not intrepret in any manner the contents of Grinnell I/O requests to, for example, insure that only allocated memory channels are being used).

The first two of these calls are used to control access to the Grinnell I/O channel and are:

ioctl(fildes, GMRIOCRESERVE, NULL)
ioctl(fildes, GMRIOCRELEASE, NULL)

The **GMRIOCRESERVE** call reserves the I/O channel for the corresponding device. If the I/O channel is not currently reserved by some other device, it is reserved for the exclusive use of the current device. Any **GMRIOCRESERVE** *ioctl*, *read* or *write* call on another Grinnell device will block until the I/O channel is released and the call can be completed. If the I/O channel is already reserved by some other device, the call will block until the I/O channel is released and can be assigned to the current device.

The **GMRIOCRELEASE** call releases the reserved I/O channel. If the I/O channel had not been reserved by the corresponding device, an error is returned. If any other **GMRIOCRESERVE** *ioctl*, *read* or *write* calls are blocked waiting for the I/O channel, the least recently blocked such call is resumed.

Both the *read* and *write* system calls perform an implicit **GMRIOCRESERVE** operation on the I/O channel before proceding with the I/O request. Similarly, an implicit **GMRIOCRELEASE** operation on the I/O channel is also performed at the completion of the request if the I/O channel was implicitly reserved at the start of the request. It is therefore only necessary to explicitly reserve and release the I/O channel if it is important that some sequnece of read and write operations be performed without any intervening operations from one of the other Grinnell devices.

The final three calls manipulate the shared Grinnell memory channels which are allocated among the different applications. They are:

ioctl(fildes, GMRIOCCHECKMEM, vecp)
ioctl(fildes, GMRIOCALLOCMEM, vecp)
ioctl(fildes, GMRIOCDEALLOCMEM, vecp)

long *vecp

The **GMRIOCCHECKMEM** call reserves the memory channel allocation bit vector for the corresponding device. If the memory allocation vector is not currently reserved by some other device, it is reserved for the exclusive use of the current device and the contents of the bit vector is returned in the longword pointed to by *vecp*. Any **GMRIOCCHECKMEM** or **GMRIOCALLOCMEM** *ioctl* call on another Grinnell device will block until the memory allocation vector is released and the call can be completed. If the memory allocation vector is already reserved by some other device, the call will block until the vector is released and can be assigned to the current device.

The **GMRIOCALLOCMEM** call allocates the memory channels corresponding to the bits set in the vector pointed to by *vecp*. If the vector is not already reserved by the current device it will first be reserved for the current device (as above). If any of the indicated memory channels are already allocated by any device, an error is returned and no memory channels are allocated. At the completion of the call, the memory channel allocation vector is released and if any other **GMRIOCCHECKMEM** or **GMRIOCALLOCMEM** *ioctl* calls are blocked waiting to reserve the vector, the least recently blocked such call is resumed.

The **GMRIOCDEALLOCMEM** call deallocates the memory channels corresponding to the bits set in the vector pointed to by *vecp*. If any of the indicated memory channels are not allocated to the current device, an error is returned and no channels are deallocated.

It is considered extremely bad form to perform any operations which affect Grinnell memory corresponding to memory channels which have not been allocated to the current device. It is also advisable to only allocate as many channels as are actually needed and to deallocate them immediately after finishing with them.

FILES

/dev/gmr	on the IUS-Vax and the VLSI-Vax
/dev/gmr[0-3]	on the GP-Vax

SEE ALSO

GMR-27 Graphic Television Display System User's Manual

HISTORY

15-Jun-84 D.Sathyanarayanan (ads) at Stanford University
Restructured for 4.2bsd.

11-Jul-81 Mike Accetta (mja) at Carnegie-Mellon University

Modified to describe new multiple IFVC sharing mechanisms.

22-Mar-80 Mike Accetta (mja) at Carnegie-Mellon University
Added description of new ioctl calls for packed-byte writes.

04-Feb-80 Mike Accetta (mja) at Carnegie-Mellon University
Created.

NAME

hk − RK6-11/RK06 and RK07 moving head disk

SYNOPSIS

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

DESCRIPTION

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

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

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

DISK SUPPORT

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

RK07 partitions:

disk	start	length	cyl
hk?a	0	15884	0-240
hk?b	15906	10032	241-392
hk?c	0	53790	0-814
hk?g	26004	27786	393-813

RK06 partitions

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

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

FILES

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

SEE ALSO

hp(4), uda(4), up(4)

DIAGNOSTICS

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

rk%d: write locked. The write protect switch was set on the drive when a write was attempted. The write operation is not recoverable.

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

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

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

hk%d: lost interrupt. A timer watching the controller detected no interrupt for an extended period while an operation was outstanding. This indicates a hardware or software failure. There is currently a hardware/software problem with spinning down drives while they are being accessed which causes this error to occur. The error causes a UNIBUS reset, and retry of the pending operations. If the controller continues to lose interrupts, this error will recur a few seconds later.

BUGS

In raw I/O *read* and *write*(2) truncate file offsets to 512-byte block boundaries, and *write* scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read, write* and *lseek*(2) should always deal in 512-byte multiples.

DEC-standard error logging should be supported.

A program to analyze the logged error information (even in its present reduced form) is needed.

The partition tables for the file systems should be read off of each pack, as they are never quite what any single installation would prefer, and this would make packs more portable.

rk%d: write locked. The write protect switch was set on the drive when a write was attempted. The write operation is not recoverable.

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

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

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

hk%d: lost interrupt. A timer watching the controller detected no interrupt for an extended period while an operation was outstanding. This indicates a hardware or software failure. There is currently a hardware/software problem with spinning down drives while they are being accessed which causes this error to occur. The error causes a UNIBUS reset, and retry of the pending operations. If the controller continues to lose interrupts, this error will recur a few seconds later.

BUGS

In raw I/O *read* and *write*(2) truncate file offsets to 512-byte block boundaries, and *write* scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read, write* and *lseek*(2) should always deal in 512-byte multiples.

DEC-standard error logging should be supported.

A program to analyze the logged error information (even in its present reduced form) is needed.

The partition tables for the file systems should be read off of each pack, as they are never quite what any single installation would prefer, and this would make packs more portable.

NAME

hp – MASSBUS disk interface

SYNOPSIS

disk hp0 at mba0 drive 0

DESCRIPTION

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

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

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

DISK SUPPORT

This driver handles both standard DEC controllers and Emulex SC750 and SC780 controllers. Standard DEC drive types are recognized according to the MASSBUS drive type register. For the Emulex controller the *drive type register* should be configured to indicate the drive is an RM02. When this is encountered, the driver checks the holding register to find out the disk geometry and, based on this information, decides what the drive type is. The following disks are supported: RM03, RM05, RP06, RM80, RP05, RP07, ML11A, ML11B, CDC 9775, CDC 9730, AMPEX Capricorn (32 sectors/track), FUJITSU Eagle (48 sectors/track), and AMPEX 9300. The origin and size (in sectors) of the pseudo-disks on each drive are as follows:

RM03 partitions

disk	start	length	cyls
hp?a	0	15884	0-99
hp?b	16000	33440	100-309
hp?c	0	131680	0-822
hp?d	49600	15884	309-408
hp?e	65440	55936	409-758
hp?f	121440	10080	759-822
hp?g	49600	82080	309-822

RM05 partitions

disk	start	length	cyls
hp?a	0	15884	0-26
hp?b	16416	33440	27-81
hp?c	0	500384	0-822
hp?d	341696	15884	562-588
hp?e	358112	55936	589-680
hp?f	414048	86176	681-822
hp?g	341696	158528	562-822
hp?h	49856	291346	82-561

RP06 partitions

disk	start	length	cyls
hp?a	0	15884	0-37
hp?b	15884	33440	38-117

hp?c	0	340670	0-814
hp?d	49324	15884	118-155
hp?e	65208	55936	156-289
hp?f	121220	219296	290-814
hp?g	49324	291192	118-814

RM80 partitions

disk	start	length	cyls
hp?a	0	15884	0-36
hp?b	16058	33440	37-114
hp?c	0	242606	0-558
hp?d	49910	15884	115-151
hp?e	68096	55936	152-280
hp?f	125888	120466	281-558
hp?g	49910	192510	115-558

RP05 partitions

disk	start	length	cyls
hp?a	0	15884	0-37
hp?b	15884	33440	38-117
hp?c	0	171798	0-410
hp?d	2242	15884	118-155
hp?e	65208	55936	156-289
hp?f	121220	50424	290-410
hp?g	2242	122320	118-410

RP07 partitions

disk	start	length	cyls
hp?a	0	15884	0-9
hp?b	16000	66880	10-51
hp?c	0	1008000	0-629
hp?d	376000	15884	235-244
hp?e	392000	307200	245-436
hp?f	699200	308600	437-629
hp?g	376000	631800	235-629
hp?h	83200	291346	52-234

CDC 9775 partitions

disk	start	length	cyls
hp?a	0	15884	0-12
hp?b	16640	66880	13-65
hp?c	0	1079040	0-842
hp?d	376320	15884	294-306
hp?e	392960	307200	307-546
hp?f	700160	378720	547-842
hp?g	376320	702560	294-842
hp?h	84480	291346	66-293

CDC 9730 partitions

disk	start	length	cyls
hp?a	0	15884	0-49
hp?b	16000	33440	50-154
hp?c	0	263360	0-822
hp?d	49600	15884	155-204
hp?e	65600	55936	205-379
hp?f	121600	141600	380-822

hp?g	49600	213600	155-822

AMPEX Capricorn partitions

disk	start	length	cyls
hp?a	0	15884	0-31
hp?b	16384	33440	32-97
hp?c	0	524288	0-1023
hp?d	342016	15884	668-699
hp?e	358400	55936	700-809
hp?f	414720	109408	810-1023
hp?g	342016	182112	668-1023
hp?h	50176	291346	98-667

FUJITSU Eagle partitions

disk	start	length	cyls
hp?a	0	15884	0-16
hp?b	16320	66880	17-86
hp?c	0	808320	0-841
hp?d	375360	15884	391-407
hp?e	391680	55936	408-727
hp?f	698880	109248	728-841
hp?g	375360	432768	391-841
hp?h	83520	291346	87-390

AMPEX 9300 partitions

disk	start	length	cyl
hp?a	0	15884	0-26
hp?b	16416	33440	27-81
hp?c	0	495520	0-814
hp?d	341696	15884	562-588
hp?e	358112	55936	589-680
hp?f	414048	81312	681-814
hp?g	341696	153664	562-814
hp?h	49856	291346	82-561

It is unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter. The hp?a partition is normally used for the root file system, the hp?b partition as a paging area, and the hp?c partition for pack-pack copying (it maps the entire disk). On disks larger than about 205 Megabytes, the hp?h partition is inserted prior to the hp?d or hp?g partition; the hp?g partition then maps the remainder of the pack. All disk partition tables are calculated using the *diskpart*(8) program.

FILES

/dev/hp[0-7][a-h]	block files
/dev/rhp[0-7][a-h]	raw files

SEE ALSO

hk(4), uda(4), up(4)

DIAGNOSTICS

hp%d%c: hard error sn%d mbsr=%b er1=%b er2=%b. An unrecoverable error occurred during transfer of the specified sector of the specified disk partition. The MASSBUS status register is printed in hexadecimal and with the error bits decoded if any error bits other than MBEXC and DTABT are set. In any case the contents of the two error registers are also printed in octal and symbolically with bits decoded. (Note that er2 is what old rp06 manuals would call er3; the terminology is that of the rm disks). The error was either unrecoverable, or a large number of retry attempts (including offset positioning and drive recalibration) could not recover the error.

hp%d: write locked. The write protect switch was set on the drive when a write was attempted. The write operation is not recoverable.

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

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

During autoconfiguration one of the following messages may appear on the console indicating the appropriate drive type was recognized. The last message indicates the drive is of a unknown type.

hp%d: 9775 (direct).
hp%d: 9730 (direct).
hp%d: 9300.
hp%d: 9762.
hp%d: capricorn.
hp%d: eagle.
hp%d: ntracks %d, nsectors %d: unknown device.

BUGS

In raw I/O *read* and *write*(2) truncate file offsets to 512-byte block boundaries, and *write* scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read, write* and *lseek*(2) should always deal in 512-byte multiples.

DEC-standard error logging should be supported.

A program to analyze the logged error information (even in its present reduced form) is needed.

The partition tables for the file systems should be read off of each pack, as they are never quite what any single installation would prefer, and this would make packs more portable.

NAME

ht – TM-03/TE-16,TU-45,TU-77 MASSBUS magtape interface

SYNOPSIS

master ht0 at mba? drive ?
tape tu0 at ht0 slave 0

DESCRIPTION

The tm-03/transport combination provides a standard tape drive interface as described in *mtio*(4). All drives provide both 800 and 1600 bpi; the TE-16 runs at 45 ips, the TU-45 at 75 ips, while the TU-77 runs at 125 ips and autoloads tapes.

SEE ALSO

mt(1), tar(1), tp(1), mtio(4), tm(4), ts(4), mt(4), ut(4)

DIAGNOSTICS

tu%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

tu%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

tu%d: can't switch density in mid-tape. An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

tu%d: hard error bn%d mbsr=%b er=%b ds=%b. A tape error occurred at block *bn*; the ht error register and drive status register are printed in octal with the bits symbolically decoded. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.

BUGS

If any non-data error is encountered on non-raw tape, it refuses to do anything more until closed.

NAME

hy − Network Systems Hyperchannel interface

SYNOPSIS

device hy0 at uba0 csr 0172410 vector hyint

DESCRIPTION

The *hy* interface provides access to a Network Systems Corporation Hyperchannel Adapter.

The network to which the interface is attached is specified at boot time with an SIOCSIFADDR ioctl. The host's address is discovered by reading the adapter status register. The interface will not transmit or receive packets until the network number is known.

DIAGNOSTICS

hy%d: unit number 0x%x port %d type %x microcode level 0x%x. Identifies the device during autoconfiguration.

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

hy%d: can't initialize. The interface was unable to allocate UNIBUS resources. This is usually due to having too many network devices on an 11/750 where there are only 3 buffered data paths.

hy%d: NEX - Non Existent Memory. Non existent memory error returned from hardware.

hy%d: BAR overflow. Bus address register overflow error returned from hardware.

hy%d: Power Off bit set, trying to reset. Adapter has lost power, driver will reset the bit and see if power is still out in the adapter.

hy%d: Power Off Error, network shutdown. Power was really off in the adapter, network connections are dropped. Software does not shut down the network unless power has been off for a while.

hy%d: RECVD MP > MPSIZE (%d). A message proper was received that is too big. Probable a driver bug. Shouldn't happen.

hy%d: xmit error − len > hy_olen [%d > %d]. Probable driver error. Shouldn't happen.

hy%d: DRIVER BUG − INVALID STATE %d. The driver state machine reached a non-existent state. Definite driver bug.

hy%d: watchdog timer expired. A command in the adapter has taken too long to complete. Driver will abort and retry the command.

hy%d: adapter power restored. Software was able to reset the power off bit, indicating that the power has been restored.

SEE ALSO

intro(4N), inet(4F)

BUGS

If the adapter does not respond to the status command issued during autoconfigure, the adapter is assumed down. A reboot is required to recognize it.

The adapter power fail interrupt seems to occur sporadically when power has, in fact, not failed. The driver will believe that power has failed only if it can not reset the power fail latch after a "reasonable" time interval. These seem to appear about 2-4 times a day on some machines. There seems to be no correlation with adapter rev level, number of ports used etc. and whether a machine will get these "bogus powerfails". They don't seem to cause any real problems so they have been ignored.

NAME

ik — Ikonas frame buffer, graphics device interface

SYNOPSIS

device ik0 at uba? csr 0172460 vector ikintr

DESCRIPTION

Ik provides an interface to an Ikonas frame buffer graphics device. Each minor device is a different frame buffer interface board. When the device is opened, its interface registers are mapped, via virtual memory, into the user processes address space. This allows the user process very high bandwidth to the frame buffer with no system call overhead.

Bytes written or read from the device are DMA'ed from or to the interface. The frame buffer XY address, its addressing mode, etc. must be set up by the user process before calling write or read.

Other communication with the driver is via ioctls. The IK_GETADDR ioctl returns the virtual address where the user process can find the interface registers. The IK_WAITINT ioctl suspends the user process until the ikonas device has interrupted (for whatever reason — the user process has to set the interrupt enables).

FILES

/dev/ik

DIAGNOSTICS

None.

BUGS

An invalid access (e.g., longword) to a mapped interface register can cause the system to crash with a machine check. A user process could possibly cause infinite interrupts hence bringing things to a crawl.

NAME

il — Interlan 10 Mb/s Ethernet interface

SYNOPSIS

device il0 at uba0 csr 161000 vector ilrint ilcint

DESCRIPTION

The *il* interface provides access to a 10 Mb/s Ethernet network through an Interlan controller.

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

The interface normally tries to use a "trailer" encapsulation to minimize copying data on input and output. This may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS

il%d: input error. The hardware indicated an error in reading a packet off the cable or an illegally sized packet.

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

SEE ALSO

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

BUGS

The PUP protocol family should be added.

NAME

imp — 1822 network interface

SYNOPSIS

pseudo-device imp

DESCRIPTION

The *imp* interface, as described in BBN Report 1822, provides access to an intelligent message processor normally used when participating in the Department of Defense ARPA network. The network interface communicates through a device controller, usually an ACC LH/DH or DEC IMP-11A, with the IMP. The interface is "reliable" and "flow-controlled" by the host-IMP protocol.

To configure IMP support, one of *acc*(4) and *css(4)* must be included. The network number on which the interface resides is specified at boot time using the SIOCSIFADDR ioctl. The host number is discovered through receipt of NOOP messages from the IMP.

The network interface is always in one of four states: up, down, initializing, or going down. When the system is booted, the interface is marked down. If the hardware controller is successfully probed, the interface enters the initializing state and transmits three NOOP messages to the IMP. It then waits for the IMP to respond with two or more NOOP messages in reply. When it receives these messages it enters the up state. The going down state is entered only when notified by the IMP of an impending shutdown. Packets may be sent through the interface only while it is in the up state. Packets received in any other state are dropped with the error ENETDOWN returned to the caller.

DIAGNOSTICS

imp%d: leader error. The IMP reported an error in a leader (1822 message header). This causes the interface to be reset and any packets queued up for transmission to be purged.

imp%d: going down in 30 seconds.
imp%d: going down for hardware PM.
imp%d: going down for reload software.
imp%d: going down for emergency reset. The Network Control Center (NCC) is manipulating the IMP. By convention these messages are reported to all hosts on an IMP.

imp%d: reset (host %d/imp %d). The host has received a NOOP message which caused it to reset its notion of its current address. This normally occurs at boot time, though it may also occur while the system is running (for example, if the IMP-controller cable is disconnected, then reconnected).

imp%d: host dead. The IMP has noted a host, to which a prior packet was sent, is not up.

imp%d: host unreachable. The IMP has discovered a host, to which a prior packet was sent, is not accessible.

imp%d: data error. The IMP noted an error in data transmitted. The host-IMP interface is reset and the host enters the init state (awaiting NOOP messages).

imp%d: interface reset. The reset process has been completed.

imp%d: marked down. After receiving a "going down in 30 seconds" message, and waiting 30 seconds, the host has marked the IMP unavailable. Before packets may be sent to the IMP again, the IMP must notify the host, through a series of NOOP messages, that it is back up.

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

SEE ALSO

intro(4N), inet(4F), acc(4), css(4)

NAME

imp − IMP raw socket interface

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>
#include <netimp/if_imp.h>

s = socket(AF_IMPLINK, SOCK_RAW, IMPLINK_IP);

DESCRIPTION

The raw imp socket provides direct access to the *imp*(4) network interface. Users send packets through the interface using the *send*(2) calls, and receive packets with the *recv*(2), calls. All outgoing packets must have space for an 1822 96-bit leader on the front. Likewise, packets received by the user will have this leader on the front. The 1822 leader and the legal values for the various fields are defined in the include file *<netimp/if_imp.h>*.

The raw imp interface automatically installs the length and destination address in the 1822 leader of all outgoing packets; these need not be filled in by the user.

DIAGNOSTICS

An operation on a socket may fail with one of the following errors:

[EISCONN] when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;

[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;

[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

intro(4N), inet(4F), imp(4)

NAME

inet − Internet protocol family

SYNOPSIS

#include <sys/types.h>
#include <netinet/in.h>

DESCRIPTION

The Internet protocol family is a collection of protocols layered atop the *Internet Protocol* (IP) transport layer, and utilizing the Internet address format. The Internet family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types; the SOCK_RAW interface provides access to the IP protocol.

ADDRESSING

Internet addresses are four byte quantities, stored in network standard format (on the VAX these are word and byte reversed). The include file *<netinet/in.h>* defines this address as a discriminated union.

Sockets bound to the Internet protocol family utilize the following addressing structure,

```
struct sockaddr_in {
        short      sin_family;
        u_short    sin_port;
        struct     in_addr sin_addr;
        char       sin_zero[8];
};
```

Sockets may be created with the 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(4P), udp(4P), ip(4P)

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*(2) call may also be used to fix the destination for future packets (in which case the *read*(2) or *recv*(2) and *write*(2) or *send*(2) system calls may be used).

Outgoing packets automatically have an IP header prepended to them (based on the destination address and the protocol number the socket is created with). Likewise, incoming packets have their IP header stripped before being sent to the user.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
[ENOBUFS]	when the system runs out of memory for an internal data structure;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

send(2), recv(2), intro(4N), inet(4F)

BUGS

One should be able to send and receive ip options.

The protocol should be settable after socket creation.

NAME

ipbroadcast -- broadcasting Internet Protocol packets

SYNOPSIS

#include <sys/types.h> #include <netinet/in.h>

DESCRIPTION

Note: The addressing conventions described in this manual page conform to *de facto* Internet standards. They do *not* correspond to the conventions used by most Berkeley 4.2BSD implementations.

On networks that support it, it is possible to send an Internet Protocol (IP) packet as a broadcast; that is, it will be received by every host on that network. A set of host addresses is reserved to denote broadcasts; they have in common that fields normally used to specified a specific hosts are filled with bits all set to one.

There are three kinds of IP broadcast addresses:

This network	The distinguished address INADDR_BROADCAST (255.255.255.255) denotes a broadcast to be sent on the network this host is attached to; if it is attached to more than one network that supports broadcasting, the system might not send the packet on all of these networks. A packet with this destination address will never be forwarded by a gateway.
Specific Network	The destination address is composed of the network number for the desired destination network, with the rest of the address filled with ones. For example, to broadcast a packet on network 128.12.0.0 (a Class B network), address it to 128.12.255.255. It may not be possible to send this sort of broadcast to every network.
Specific Subnet	The destination address is composed of the network number and subnet number for the desired destination subnet, with the rest of the address filles with ones. For example, to broadcast a packet on subnet 40 of 128.12.0.0, address it to 128.12.40.255. For a network that is not divided into more than one subnet, this kind of address is identical to the "Specific Network" address mentioned above.

There is no single address to denote "broadcast to the entire Internet", as this considered a bad thing to do.

To find out the broadcast address for the network to which an interface is attached, use the SIOCGBRDADDR ioctl, which is exactly the same as the SIOCGIFADDR ioctl (see *intro*(4n)), except that it returns the "Specific Subnet" broadcast address for the interface. (If the interface does not support broadcasts, the ioctl will fail.)

The restriction that only the super-user may send a broadcast has been removed. It is possible to broadcast a TCP packet; it is not possible to do anything useful this way.

BUGS

The current implementation of IP broadcasts in 4.2 is deficient in several ways:

A broadcast sent to a "Specific Network" will not necessarily be delivered to all the hosts on that network, but rather to some subset of its subnets.

A broadcast received by Unix for which it is meant to act as a gateway will not be delivered properly if the destination is a locally-connected network.

Although broadcasts addressed by the old-style Berkeley addressing convention (all zeros instead of all ones) will be accepted by the system if received, they cannot be sent.

The inet_makeaddr() function (see *inet*(3n)) does not work properly for broadcast addresses. Also, inet_network() and inet_addr() return the same value (-1) for INADDR_BROADCAST and to indicate failure. This requires some caution on the part of the programmer.

NAME

kg – KL-11/DL-11W line clock

SYNOPSIS

device kg0 at uba0 csr 0176500 vector kglock

DESCRIPTION

A kl-11 or dl-11w can be used as an alternate real time clock source. When configured, certain system statistics and, optionally, system profiling work will be collected each time the clock interrupts. For optimum accuracy in profiling, the dl-11w should be configured to interrupt at the highest possible priority level. The *kg* device driver automatically calibrates itself to the line clock frequency.

SEE ALSO

kgmon(8), config(8)

NAME

lo − software loopback network interface

SYNOPSIS

pseudo-device loop

DESCRIPTION

The *loop* interface is a software loopback mechanism which may be used for performance analysis, software testing, and/or local communication. By default, the loopback interface is accessible at address 127.0.0.1 (non-standard); 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(4N), inet(4F)

BUGS

It should handle all address and protocol families. An approved network address should be reserved for this interface.

NAME

lp − line printer

SYNOPSIS

device lp0 at uba0 csr 0177514 vector lpintr

DESCRIPTION

Lp provides the interface to any of the standard DEC line printers on an LP-11 parallel interface. When it is opened or closed, a suitable number of page ejects is generated. Bytes written are printed.

The unit number of the printer is specified by the minor device after removing the low 3 bits, which act as per-device parameters. Currently only the lowest of the low three bits is interpreted: if it is set, the device is treated as having a 64-character set, rather than a full 96-character set. In the resulting half-ASCII mode, lower case letters are turned into upper case and certain characters are escaped according to the following table:

{	-(
}	-)
`	-'
|	-!
~	-^

The driver correctly interprets carriage returns, backspaces, tabs, and form feeds. Lines longer than the maximum page width are truncated. The default page width is 132 columns. This may be overridden by specifying, for example, "flags 256" .

FILES

/dev/lp

SEE ALSO

lpr(1)

DIAGNOSTICS

None.

NAME

mem, kmem − main memory

DESCRIPTION

Mem is a special file that is an image of the main memory of the computer. It may be used, for example, to examine (and even to patch) the system.

Byte addresses in *mem* are interpreted as physical memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file *kmem* is the same as *mem* except that kernel virtual memory rather than physical memory is accessed.

On PDP11's, the I/O page begins at location 0160000 of *kmem* and per-process data for the current process begins at 0140000. On VAX 11/780 the I/O space begins at physical address 20000000(16); on an 11/750 I/O space addresses are of the form fxxxxx(16); on all VAX'en per-process data for the current process is at virtual 7ffff000(16).

FILES

/dev/mem
/dev/kmem

BUGS

On PDP11's and VAX's, memory files are accessed one byte at a time, an inappropriate method for some device registers.

NAME
mt − TM78/TU-78 MASSBUS magtape interface

SYNOPSIS
master mt0 at mba? drive ?
tape mu0 at mt0 slave 0

DESCRIPTION
The tm78/tu-78 combination provides a standard tape drive interface as described in *mtio*(4). Only 1600 and 6250 bpi are supported; the TU-78 runs at 125 ips and autoloads tapes.

SEE ALSO
mt(1), tar(1), tp(1), mtio(4), tm(4), ts(4), ut(4)

DIAGNOSTICS
mu%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

mu%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

mu%d: can't switch density in mid-tape. An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

mu%d: hard error bn%d mbsr=%b er=%x ds=%b. A tape error occurred at block *bn*; the mt error register and drive status register are printed in octal with the bits symbolically decoded. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.

mu%d: blank tape. An attempt was made to read a blank tape (a tape without even end-of-file marks).

mu%d: offline. During an i/o operation the device was set offline. If a non-raw tape was used in the access it is closed.

BUGS
If any non-data error is encountered on non-raw tape, it refuses to do anything more until closed.

NAME

mtio — UNIX magtape interface

DESCRIPTION

The files *mt0, ..., mt15* refer to the UNIX magtape drives, which may be on the MASSBUS using the TM03 formatter *ht*(4), or TM78 formatter, *mt*(4), or on the UNIBUS using either the TM11 or TS11 formatters *tm*(4), TU45 compatible formatters, *ut*(4), or *ts*(4). The following description applies to any of the transport/controller pairs. The files *mt0, ..., mt7* are 800bpi, *mt8, ..., mt15* are 1600bpi, and *mt16, ..., mt23* are 6250bpi. (But note that only 1600 bpi is available with the TS11.) The files *mt0, ..., mt3*, *mt8, ..., mt11*, and *mt16, ..., mt19* are rewound when closed; the others are not. When a file open for writing is closed, two end-of-files are written. If the tape is not to be rewound it is positioned with the head between the two tapemarks.

A standard tape consists of a series of 1024 byte records terminated by an end-of-file. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it tends to create monstrous record gaps.

The *mt* files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the 'raw' interface is appropriate. The associated files are named *rmt0, ..., rmt23,* but the same minor-device considerations as for the regular files still apply. A number of other ioctl operations are available on raw magnetic tape. The following definitions are from <*sys/mtio.h*>:

```
/*
 * Structures and definitions for mag tape io control commands
 */

/* structure for MTIOCTOP - mag tape op command */
struct  mtop  {
        short   mt_op;          /* operations defined below */
        daddr_t mt_count;       /* how many of them */
};

/* operations */
#define MTWEOF      0       /* write an end-of-file record */
#define MTFSF       1       /* forward space file */
#define MTBSF       2       /* backward space file */
#define MTFSR       3       /* forward space record */
#define MTBSR       4       /* backward space record */
#define MTREW       5       /* rewind */
#define MTOFFL      6       /* rewind and put the drive offline */
#define MTNOP       7       /* no operation, sets status only */

/* structure for MTIOCGET - mag tape get status command */

struct  mtget  {
        short   mt_type;        /* type of magtape device */
/* the following two registers are grossly device dependent */
        short   mt_dsreg;       /* "drive status" register */
        short   mt_erreg;       /* "error" register */
/* end device-dependent registers */
        short   mt_resid;       /* residual count */
/* the following two are not yet implemented */
        daddr_t mt_fileno;      /* file number of current position */
        daddr_t mt_blkno;       /* block number of current position */
/* end not yet implemented */
};

/*
 * Constants for mt_type byte
 */
#define MT_ISTS                 0x01
#define MT_ISHT                 0x02
#define MT_ISTM                 0x03
#define MT_ISMT                 0x04
#define MT_ISUT                 0x05
#define MT_ISCPC        0x06
#define MT_ISAR                 0x07

/* mag tape io control commands */
#define MTIOCTOP        _IOW(m, 1, struct mtop)                 /* do a mag tape op */
#define MTIOCGET        _IOR(m, 2, struct mtget)        /* get tape status */

#ifndef KERNEL
#define DEFTAPE         "/dev/rmt12"
#endif
```

Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size; if the record is long, an error is indicated. In raw tape I/O seeks are ignored. A zero byte count is returned when a tape mark is read, but another read will fetch the first record of the new tape file.

FILES

/dev/mt?
/dev/rmt?

SEE ALSO

mt(1), tar(1), tp(1), ht(4), tm(4), ts(4), mt(4), ut(4)

BUGS

The status should be returned in a device independent format.

NAME

null − data sink

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

pcl − DEC CSS PCL-11 B Network Interface

SYNOPSIS

device pcl0 at uba? csr 164200 vector pclxint pclrint

DESCRIPTION

The *pcl* device provides an IP-only interface to the DEC CSS PCL-11 time division multiplexed network bus. The controller itself is not accessible to users.

The hosts's address is specified with the SIOCSIFADDR ioctl. The interface will not transmit or receive any data before its address is defined.

As the PCL-11 hardware is only capable of having 15 interfaces per network, a single-byte host-on-network number is used, with range [1..15] to match the TDM bus addresses of the interfaces.

The interface currently only supports the Internet protocol family and only provides "natural" (header) encapsulation.

DIAGNOSTICS

pcl%d: can't init. Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).

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

pcl%d: stray xmit interrupt. An interrupt occured when no output had previously been started.

pcl%d: master. The TDM bus had no station providing "bus master" timing signals, so this interface has assumed the "master" role. This message should only appear at most once per UNIBUS INIT on a single system. Unless there is a hardware failure, only one station may be master at at time.

pcl%d: send error, tcr=%b, tsr=%b. The device indicated a problem sending data on output. If a "receiver offline" error is detected, it is not normally logged unless the option PCL_TESTING has been selected, as this causes a lot of console chatter when sending to a down machine. However, this option is quite useful when debugging problems with the PCL interfaces.

pcl%d: rcv error, rcr=%b rsr=%b. The device indicated a problem receiving data on input.

pcl%d: bad len=%d. An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never happen as the maximum size of a PCL message has been agreed upon to be 1008 bytes (same as ArpaNet message).

SEE ALSO

intro(4N), inet(4F)

NAME

ps − Evans and Sutherland Picture System 2 graphics device interface

SYNOPSIS

device ps0 at uba? csr 0172460 vector psintr

DESCRIPTION

The *ps* driver provides access to an Evans and Sutherland Picture System 2 graphics device. Each minor device is a new PS2. When the device is opened, its interface registers are mapped, via virtual memory, into a user process's address space. This allows the user process very high bandwidth to the device with no system call overhead.

DMA to and from the PS2 is not supported. All read and write system calls will fail. All data is moved to and from the PS2 via programmed I/O using the device's interface registers.

Commands are fed to and from the driver using the following ioctls:

PSIOGETADDR
: Returns the virtual address through which the user process can access the device's interface registers.

PSIOAUTOREFRESH
: Start auto refreshing the screen. The argument is an address in user space where the following data resides. The first longword is a *count* of the number of static refresh buffers. The next *count* longwords are the addresses in refresh memory where the refresh buffers lie. The driver will cycle thru these refresh buffers displaying them one by one on the screen.

PSIOAUTOMAP
: Start automatically passing the display file thru the matrix processor and into the refresh buffer. The argument is an address in user memory where the following data resides. The first longword is a *count* of the number of display files to operate on. The next *count* longwords are the address of these display files. The final longword is the address in refresh buffer memory where transformed coordinates are to be placed if the driver is not in double buffer mode (see below).

PSIODOUBLEBUFFER
: Cause the driver to double buffer the output from the map that is going to the refresh buffer. The argument is again a user space address where the real arguments are stored. The first argument is the starting address of refresh memory where the two double buffers are located. The second argument is the length of each double buffer. The refresh mechanism displays the current double buffer, in addition to its static refresh lists, when in double buffer mode.

PSIOSINGLEREFRESH
: Single step the refresh process. That is, the driver does not continually refresh the screen.

PSIOSINGLEMAP
: Single step the matrix process. The driver does not automatically feed display files thru the matrix unit.

PSIOSINGLEBUFFER
: Turn off double buffering.

PSIOTIMEREFRESH
: The argument is a count of the number of refresh interrupts to take before turning off the screen. This is used to do time exposures.

PSIOWAITREFRESH
: Suspend the user process until a refresh interrupt has occurred. If in TIMEREFRESH

mode, suspend until count refreshes have occurred.

PSIOSTOPREFRESH
: Wait for the next refresh, stop all refreshes, and then return to user process.

PSIOWAITMAP
: Wait until a map done interrupt has occurred.

PSIOSTOPMAP
: Wait for a map done interrupt, do not restart the map, and then return to the user.

FILES

/dev/ps

DIAGNOSTICS

ps device intr.
ps dma intr. An interrupt was received from the device. This shouldn't happen, check your device configuration for overlapping interrupt vectors.

BUGS

An invalid access (e.g., longword) to a mapped interface register can cause the system to crash with a machine check. A user process could possibly cause infinite interrupts hence bringing things to a crawl.

NAME

pty — pseudo terminal driver

SYNOPSIS

pseudo-device pty

DESCRIPTION

The *pty* driver provides support for a device-pair termed a *pseudo terminal.* A pseudo terminal is a pair of character devices, a *master* device and a *slave* device. The slave device provides processes an interface identical to that described in *tty*(4). However, whereas all other devices which provide the interface described in *tty*(4) have a hardware device of some sort behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, anything written on the master device is given to the slave device as input and anything written on the slave device is presented as input on the master device.

In configuring, if no optional "count" is given in the specification, 16 pseudo terminal pairs are configured.

The following *ioctl* calls apply only to pseudo terminals:

TIOCSTOP
: Stops output to a terminal (e.g. like typing ^S). Takes no parameter.

TIOCSTART
: Restarts output (stopped by TIOCSTOP or by typing ^S). Takes no parameter.

TIOCPKT
: Enable/disable *packet* mode. Packet mode is enabled by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. When applied to the master side of a pseudo terminal, each subsequent *read* from the terminal will return data written on the slave part of the pseudo terminal preceded by a zero byte (symbolically defined as TIOCPKT_DATA), or a single byte reflecting control status information. In the latter case, the byte is an inclusive-or of zero or more of the bits:

  TIOCPKT_FLUSHREAD
  : whenever the read queue for the terminal is flushed.

  TIOCPKT_FLUSHWRITE
  : whenever the write queue for the terminal is flushed.

  TIOCPKT_STOP
  : whenever output to the terminal is stopped a la ^S.

  TIOCPKT_START
  : whenever output to the terminal is restarted.

  TIOCPKT_DOSTOP
  : whenever *t_stopc* is ^S and *t_startc* is ^Q.

  TIOCPKT_NOSTOP
  : whenever the start and stop characters are not ^S/^Q.

  This mode is used by *rlogin*(1C) and *rlogind*(8C) to implement a remote-echoed, locally ^S/^Q flow-controlled remote login with proper back-flushing of output; it can be used by other similar programs.

TIOCREMOTE
: A mode for the master half of a pseudo terminal, independent of TIOCPKT. This mode causes input to the pseudo terminal to be flow controlled and not input edited (regardless of the terminal mode). Each write to the control terminal produces a record boundary for the process reading the terminal. In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an end-of-file

character. TIOCREMOTE can be used when doing remote line editing in a window manager, or whenever flow controlled input is required.

FILES

/dev/pty[p-r][0-9a-f]	master pseudo terminals
/dev/tty[p-r][0-9a-f]	slave pseudo terminals

DIAGNOSTICS

None.

BUGS

It is not possible to send an EOT.

NAME

pup – Xerox PUP-I protocol family

SYNOPSIS

#include <sys/types.h>
#include <netpup/pup.h>

DESCRIPTION

The PUP-I protocol family is a collection of protocols layered atop the PUP Level-0 packet format, and utilizing the PUP Internet address format. The PUP family is currently supported only by a raw interface.

ADDRESSING

PUP addresses are composed of network, host, and port portions. The include file <*netpup/pup.h*> defines this address as,

```
struct  pupport {
        u_char      pup_net;
        u_char      pup_host;
        u_char      pup_socket[4];
};
```

Sockets bound to the PUP protocol family utilize the following addressing structure,

```
struct sockaddr_pup {
        short       spup_family;
        short       spup_zero1;
        u_char      spup_net;
        u_char      spup_host;
        u_char      spup_sock[4];
        char        spup_zero2[4];
};
```

HEADERS

The current PUP support provides only raw access to the 3Mb/s Ethernet. Packets sent through this interface must have space for the following packet header present at the front of the message,

```
struct pup_header {
        u_short     pup_length;
        u_char      pup_tcontrol;       /* transport control */
        u_char      pup_type;           /* protocol type */
        u_long      pup_id;             /* used by protocols */
        u_char      pup_dnet;           /* destination */
        u_char      pup_dhost;
        u_char      pup_dsock[4];
        u_char      pup_snet;           /* source */
        u_char      pup_shost;
        u_char      pup_ssock[4];
};
```

The sender should fill in the *pup_tcontrol*, *pup_type*, and *pup_id* fields. The remaining fields are filled in by the system. The system checks the message to insure its size is valid and, calulates a checksum for the message. If no checksum should be calculated, the checksum field (the last 16-bit word in the message) should be set to PUP_NOCKSUM.

The *pup_tcontrol* field is restricted to be 0 or PUP_TRACE; PUP_TRACE indicates packet tracing should be performed. The *pup_type* field may not be 0.

On input, the entire packet, including header, is provided the user. No checksum validation is performed.

SEE ALSO

intro(4N), pup(4P), en(4)

BUGS

The only interface which currently supports use of pup's is the Xerox 3 Mb/s *en*(4) interface.

With the release of the second generation, PUP-II, protocols it is not clear what future PUP-I has. Consequently, there has been little motivation to provide extensive kernel support.

NAME

pup — raw PUP socket interface

SYNOPSIS

#include <sys/socket.h>
#include <netpup/pup.h>

socket(AF_PUP, SOCK_RAW, PUPPROTO_BSP);

DESCRIPTION

A raw pup socket provides PUP-I access to an Ethernet network. Users send packets using the *sendto* call, and receive packets with the *recvfrom* call. All outgoing packets must have space present at the front of the packet to allow the PUP header to be filled in. The header format is described in *pup*(4F). Likewise, packets received by the user will have the PUP header on the front. The PUP header and legal values for the various fields are defined in the include file *< netpup/pup.h>*.

The raw pup interface automatically installs the length and source and destination addresses in the PUP header of all outgoing packets; these need not be filled in by the user. The only control bit that may be set in the *tcontrol* field of outgoing packets is the "trace" bit. A checksum is calculated unless the sender sets the checksum field to PUP_NOCKSUM.

DIAGNOSTICS

A socket operation may fail and one of the following will be 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.

A *sendto* operation may fail if one of the following is true:

[EINVAL] Insufficient space was left by the user for the PUP header.

[EINVAL] The *pup_type* field was 0 or the *pup_tcontrol* field had a bit other than PUP_TRACE set.

[EMSGSIZE] The message was not an even number of bytes, smaller than MINPUPSIZ, or large than MAXPUPSIZ.

[ENETUNREACH]
The destination address was on a network which was not directly reachable (the raw interface provides no routing support).

SEE ALSO

send(2), recv(2), intro(4N), pup(4F)

BUGS

The interface is untested against other PUP implementations.

NAME

rx − DEC RX02 floppy disk interface

SYNOPSIS

controller fx0 at uba0 csr 0177170 vector rxintr
disk rx0 at fx0 slave 0
disk rx1 at fx0 slave 1

DESCRIPTION

The *rx* device provides access to a DEC RX02 floppy disk unit with M8256 interface module (RX211 configuration). The RX02 uses 8-inch, single-sided, soft-sectored floppy disks (with pre-formatted industry-standard headers) in either single or double density.

Floppy disks handled by the RX02 contain 77 tracks, each with 26 sectors (for a total of 2,002 sectors). The sector size is 128 bytes for single density, 256 bytes for double density. Single density disks are compatible with the RX01 floppy disk unit and with IBM 3740 Series Diskette 1 systems.

In addition to normal ('block' and 'raw') i/o, the driver supports formatting of disks for either density and the ability to invoke a 2 for 1 interleaved sector mapping compatible with the DEC operating system RT-11.

The minor device number is interpreted as follows:

Bit	**Description**
0	Sector interleaving (1 disables interleaving)
1	Logical sector 1 is on track 1 (0 no, 1 yes)
2	Not used, reserved
Other	Drive number

The two drives in a single RX02 unit are treated as two disks attached to a single controller. Thus, if there are two RX02's on a system, the drives on the first RX02 are "rx0" and "rx1", while the drives on the second are "rx2" and "rx3".

When the device is opened, the density of the disk currently in the drive is automatically determined. If there is no floppy in the device, open will fail.

The interleaving parameters are represented in raw device names by the letters 'a' through 'd'. Thus, unit 0, drive 0 is called by one of the following names:

Mapping	**Device name**	**Starting track**
interleaved	/dev/rrx0a	0
direct	/dev/rrx0b	0
interleaved	/dev/rrx0c	1
direct	/dev/rrx0d	1

The mapping used on the 'c' device is compatible with the DEC operating system RT-11. The 'b' device accesses the sectors of the disk in strictly sequential order. The 'a' device is the most efficient for disk-to-disk copying.

I/O requests must start on a sector boundary, involve an integral number of complete sectors, and not go off the end of the disk.

NOTES

Even though the storage capacity on a floppy disk is quite small, it is possible to make filesystems on double density disks. For example, the command

```
        % mkfs /dev/rx0 1001 13 1 4096 512 32 0 4
```

makes a file system on the double density disk in rx0 with 436 kbytes available for file storage. Using *tar*(1) gives a more efficient utilization of the available space for file storage. Single density diskettes do not provide sufficient storage capacity to hold file systems.

A number of *ioctl*(2) calls apply to the rx devices, and have the form

```
        #include <vaxuba/rxreg.h>
        ioctl(fildes, code, arg)
        int *arg;
```

The applicable codes are:

RXIOC_FORMAT Format the diskette. The density to use is specified by the *arg* argument. 0 gives single density while non-zero gives double density.

RXIOC_GETDENS
Return the density of the diskette (0 or !=0 as above).

RXIOC_WDDMK On the next write, include a *deleted data address mark* in the header of the first sector.

RXIOC_RDDMK Return non-zero if the last sector read contained a *deleted data address mark* in its header, otherwise return 0.

ERRORS

The following errors may be returned by the above ioctl calls:

[ENODEV] Drive not ready; usually because no disk is in the drive or the drive door is open.

[ENXIO] Nonexistent drive (on open); offset is too large or not on a sector boundary or byte count is not a multiple of the sector size (on read or write); or bad (undefined) ioctl code.

[EIO] A physical error other than "not ready", probably bad media or unknown format.

[EBUSY] Drive has been opened for exclusive access.

[EBADF] No write access (on format), or wrong density; the latter can only happen if the disk is changed without closing the device (i.e., calling *close*(2)).

FILES

/dev/rx?
/dev/rrx?[a-d]

SEE ALSO

rxformat(8V), newfs(8), mkfs(8), tar(1), arff(8V)

DIAGNOSTICS

rx%d: hard error, trk %d psec %d cs=%b, db=%b, err=%x, %x, %x, %x. An unrecoverable error was encountered. The track and physical sector numbers, the device registers and the extended error status are displayed.

rx%d: state %d (reset). The driver entered a bogus state. This should not happen.

BUGS

A floppy may not be formatted if the header info on sector 1, track 0 has been damaged. Hence, it is not possible to format completely degaussed disks or disks with other formats than the two known by the hardware.

If the drive subsystem is powered down when the machine is booted, the controller won't interrupt.

NAME

tcp − Internet Transmission Control Protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, 0);

DESCRIPTION

The TCP protocol provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the SOCK_STREAM abstraction. TCP uses the standard Internet address format and, in addition, provides a per-host collection of "port addresses". Thus, each address is composed of an Internet address specifying the host and network, with a specific TCP port on the host identifying the peer entity.

Sockets utilizing the tcp protocol are either "active" or "passive". Active sockets initiate connections to passive sockets. By default TCP sockets are created active; to create a passive socket the *listen*(2) system call must be used after binding the socket with the *bind*(2) system call. Only passive sockets may use the *accept*(2) call to accept incoming connections. Only active sockets may use the *connect*(2) call to initiate connections.

Passive sockets may "underspecify" their location to match incoming connection requests from multiple networks. This technique, termed "wildcard addressing", allows a single server to provide service to clients on multiple networks. To create a socket which listens on all networks, the Internet address INADDR_ANY must be bound. The TCP port may still be specified at this time; if the port is not specified the system will assign one. Once a connection has been established the socket's address is fixed by the peer entity's location. The address assigned the socket is the address associated with the network interface through which packets are being transmitted and received. Normally this address corresponds to the peer entity's network.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one;
[ENOBUFS]	when the system runs out of memory for an internal data structure;
[ETIMEDOUT]	when a connection was dropped due to excessive retransmissions;
[ECONNRESET]	when the remote peer forces the connection to be closed;
[ECONNREFUSED]	when the remote peer actively refuses connection establishment (usually because no process is listening to the port);
[EADDRINUSE]	when an attempt is made to create a socket with a port which has already been allocated;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

intro(4N), inet(4F)

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

tm − TM-11/TE-10 magtape interface

SYNOPSIS

controller tm0 at uba? csr 0172520 vector tmintr
tape te0 at tm0 drive 0

DESCRIPTION

The tm-11/te-10 combination provides a standard tape drive interface as described in *mtio*(4). Hardware implementing this on the VAX is typified by the Emulex TC-11 controller operating with a Kennedy model 9300 tape transport, providing 800 and 1600 bpi operation at 125 ips.

SEE ALSO

mt(1), tar(1), tp(1), mtio(4), ht(4), ts(4), mt(4), ut(4)

DIAGNOSTICS

te%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

te%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

te%d: can't switch density in mid-tape. An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

te%d: hard error bn%d er=%b. A tape error occurred at block *bn*; the tm error register is printed in octal with the bits symbolically decoded. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.

te%d: lost interrupt. A tape operation did not complete within a reasonable time, most likely because the tape was taken off-line during rewind or lost vacuum. The controller should, but does not, give an interrupt in these cases. The device will be made available again after this message, but any current open reference to the device will return an error as the operation in progress aborts.

BUGS

If any non-data error is encountered on non-raw tape, it refuses to do anything more until closed.

NAME

ts – TS-11 magtape interface

SYNOPSIS

controller zs0 at uba? csr 0172520 vector tsintr
tape ts0 at zs0 drive 0

DESCRIPTION

The ts-11 combination provides a standard tape drive interface as described in *mtio*(4). The ts-11 operates only at 1600 bpi, and only one transport is possible per controller.

SEE ALSO

mt(1), tar(1), tp(1), mtio(4), ht(4), tm(4), mt(4), ut(4)

DIAGNOSTICS

ts%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

ts%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

ts%d: hard error bn%d xs0=%b. A hard error occurred on the tape at block *bn*; status register 0 is printed in octal and symbolically decoded as bits.

BUGS

If any non-data error is encountered on non-raw tape, it refuses to do anything more until closed.

The device lives at the same address as a tm-11 *tm*(4); as it is very difficult to get this device to interrupt, a generic system assumes that a ts is present whenever no tm-11 exists but the csr responds and a ts-11 is configured. This does no harm as long as a non-existent ts-11 is not accessed.

NAME

tty − general terminal interface

SYNOPSIS

#include <sgtty.h>

DESCRIPTION

This section describes both a particular special file **/dev/tty** and the terminal drivers used for conversational computing.

Line disciplines.

The system provides different *line disciplines* for controlling communications lines. In this version of the system there are three disciplines available:

old	The old (standard) terminal driver. This is used when using the standard shell *sh*(1) and for compatibility with other standard version 7 UNIX systems.
new	A newer terminal driver, with features for job control; this must be used when using *csh*(1).
net	A line discipline used for networking and loading data into the system over communications lines. It allows high speed input at very low overhead, and is described in *bk*(4).

Line discipline switching is accomplished with the TIOCSETD *ioctl:*

```
int ldisc = LDISC; ioctl(filedes, TIOCSETD, &ldisc);
```

where LDISC is OTTYDISC for the standard tty driver, NTTYDISC for the new driver and NETLDISC for the networking discipline. The standard (currently old) tty driver is discipline 0 by convention. The current line discipline can be obtained with the TIOCGETD ioctl. Pending input is discarded when the line discipline is changed.

All of the low-speed asynchronous communications ports can use any of the available line disciplines, no matter what hardware is involved. The remainder of this section discusses the "old" and "new" disciplines.

The control terminal.

When a terminal file is opened, it causes the process to wait until a connection is established. In practice, user programs seldom open these files; they are opened by *init*(8) and become a user's standard input and output file.

If a process which has no control terminal opens a terminal file, then that terminal file becomes the control terminal for that process. The control terminal is thereafter inherited by a child process during a *fork*(2), even if the control terminal is closed.

The file /dev/tty is, in each process, a synonym for a *control terminal* associated with that process. It is useful for programs that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand a file name for output, when typed output is desired and it is tiresome to find out which terminal is currently in use.

Process groups.

Command processors such as *csh*(1) can arbitrate the terminal between different *jobs* by placing related jobs in a single process group and associating this process group with the terminal. A terminals associated process group may be set using the TIOCSPGRP *ioctl*(2):

```
ioctl(fildes, TIOCSPGRP, &pgrp)
```

or examined using TIOCGPGRP rather than TIOCSPGRP, returning the current process group in *pgrp*. The new terminal driver aids in this arbitration by restricting access to the terminal by processes which are not in the current process group; see **Job access control** below.

Modes.

The terminal drivers have three major modes, characterized by the amount of processing on the input and output characters:

cooked	The normal mode. In this mode lines of input are collected and input editing is done. The edited line is made available when it is completed by a newline or when an EOT (control-D, hereafter ^D) is entered. A carriage return is usually made synonymous with newline in this mode, and replaced with a newline whenever it is typed. All driver functions (input editing, interrupt generation, output processing such as delay generation and tab expansion, etc.) are available in this mode.
CBREAK	This mode eliminates the character, word, and line editing input facilities, making the input character available to the user program as it is typed. Flow control, literal-next and interrupt processing are still done in this mode. Output processing is done.
RAW	This mode eliminates all input processing and makes all input characters available as they are typed; no output processing is done either.

The style of input processing can also be very different when the terminal is put in non-blocking i/o mode; see *fcntl*(2). In this case a *read*(2) from the control terminal will never block, but rather return an error indication (EWOULDBLOCK) if there is no input available.

A process may also request a SIGIO signal be sent it whenever input is present. To enable this mode the FASYNC flag should be set using *fcntl*(2).

Input editing.

A UNIX terminal ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely choked, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently this limit is 256 characters. In the old terminal driver all the saved characters are thrown away when the limit is reached, without notice; the new driver simply refuses to accept any further input, and rings the terminal bell.

Input characters are normally accepted in either even or odd parity with the parity bit being stripped off before the character is given to the program. By clearing either the EVEN or ODD bit in the flags word it is possible to have input characters with that parity discarded (see the **Summary** below.)

In all of the line disciplines, it is possible to simulate terminal input using the TIOCSTI ioctl, which takes, as its third argument, the address of a character. The system pretends that this character was typed on the argument terminal, which must be the control terminal except for the super-user (this call is not in standard version 7 UNIX).

Input characters are normally echoed by putting them in an output queue as they arrive. This may be disabled by clearing the ECHO bit in the flags word using the *stty*(3) call or the TIOCSETN or TIOCSETP ioctls (see the **Summary** below).

In cooked mode, terminal input is processed in units of lines. A program attempting to read will normally be suspended until an entire line has been received (but see the description of SIGTTIN in **Modes** above and FIONREAD in **Summary** below.) No matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, line editing is normally done, with the character '#' logically erasing the last character typed and the character '@' logically erasing the entire current input line. These are often reset on crt's, with ^H replacing #, and ^U replacing @. These characters never erase beyond the beginning of the current input line or an ^D. These characters may be entered literally by preceding them with '\'; in the old teletype driver both the '\' and the character entered literally will appear on the screen; in the new driver the '\' will normally disappear.

The drivers normally treat either a carriage return or a newline character as terminating an input line, replacing the return with a newline and echoing a return and a line feed. If the CRMOD bit is cleared in the local mode word then the processing for carriage return is disabled, and it is simply echoed as a return, and does not terminate cooked mode input.

In the new driver there is a literal-next character ^V which can be typed in both cooked and CBREAK mode preceding **any** character to prevent its special meaning. This is to be preferred to the use of '\' escaping erase and kill characters, but '\' is (at least temporarily) retained with its old function in the new driver for historical reasons.

The new terminal driver also provides two other editing characters in normal mode. The word-erase character, normally ^W, erases the preceding word, but not any spaces before it. For the purposes of ^W, a word is defined as a sequence of non-blank characters, with tabs counted as blanks. Finally, the reprint character, normally ^R, retypes the pending input beginning on a new line. Retyping occurs automatically in cooked mode if characters which would normally be erased from the screen are fouled by program output.

Input echoing and redisplay

In the old terminal driver, nothing special occurs when an erase character is typed; the erase character is simply echoed. When a kill character is typed it is echoed followed by a new-line (even if the character is not killing the line, because it was preceded by a '\'!.)

The new terminal driver has several modes for handling the echoing of terminal input, controlled by bits in a local mode word.

Hardcopy terminals. When a hardcopy terminal is in use, the LPRTERA bit is normally set in the local mode word. Characters which are logically erased are then printed out backwards preceded by '\' and followed by '/' in this mode.

Crt terminals. When a crt terminal is in use, the LCRTBS bit is normally set in the local mode word. The terminal driver then echoes the proper number of erase characters when input is erased; in the normal case where the erase character is a ^H this causes the cursor of the terminal to back up to where it was before the logically erased character was typed. If the input has become fouled due to interspersed asynchronous output, the input is automatically retyped.

Erasing characters from a crt. When a crt terminal is in use, the LCRTERA bit may be set to cause input to be erased from the screen with a "backspace-space-backspace" sequence when character or word deleting sequences are used. A LCRTKIL bit may be set as well, causing the input to be erased in this manner on line kill sequences as well.

Echoing of control characters. If the LCTLECH bit is set in the local state word, then non-printing (control) characters are normally echoed as ^X (for some X) rather than being echoed unmodified; delete is echoed as ^?.

The normal modes for using the new terminal driver on crt terminals are speed dependent. At speeds less than 1200 baud, the LCRTERA and LCRTKILL processing is painfully slow, so *stty*(1) normally just sets LCRTBS and LCTLECH; at speeds of 1200 baud or greater all of these bits are normally set. *Stty*(1) summarizes these option settings and the use of the new terminal driver as "newcrt."

Output processing.

When one or more characters are written, they are actually transmitted to the terminal as soon as previously-written characters have finished typing. (As noted above, input characters are normally echoed by putting them in the output queue as they arrive.) When a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold the program is resumed. Even parity is normally generated on output. The EOT character is not transmitted in cooked mode to prevent terminals that respond to it from hanging up; programs using raw or cbreak mode should be careful.

The terminal drivers provide necessary processing for cooked and CBREAK mode output including delay generation for certain special characters and parity generation. Delays are available after backspaces ^H, form feeds ^L, carriage returns ^M, tabs ^I and newlines ^J. The driver will also optionally expand tabs into spaces, where the tab stops are assumed to be set every eight columns. These functions are controlled by bits in the tty flags word; see **Summary** below.

The terminal drivers provide for mapping between upper and lower case on terminals lacking lower case, and for other special processing on deficient terminals.

Finally, in the new terminal driver, there is a output flush character, normally ^O, which sets the LFLUSHO bit in the local mode word, causing subsequent output to be flushed until it is cleared by a program or more input is typed. This character has effect in both cooked and CBREAK modes and causes pending input to be retyped if there is any pending input. An ioctl to flush the characters in the input and output queues TIOCFLUSH, is also available.

Upper case terminals and Hazeltines

If the LCASE bit is set in the tty flags, then all upper-case letters are mapped into the corresponding lower-case letter. The upper-case letter may be generated by preceding it by '\'. If the new terminal driver is being used, then upper case letters are preceded by a '\' when output. In addition, the following escape sequences can be generated on output and accepted on input:

for	`	|	~	{	}
use	\'	\!	\^	\(	\)

To deal with Hazeltine terminals, which do not understand that ~ has been made into an ASCII character, the LTILDE bit may be set in the local mode word when using the new terminal driver; in this case the character ~ will be replaced with the character ` on output.

Flow control.

There are two characters (the stop character, normally ^S, and the start character, normally ^Q) which cause output to be suspended and resumed respectively. Extra stop characters typed when output is already stopped have no effect, unless the start and stop characters are made the same, in which case output resumes.

A bit in the flags word may be set to put the terminal into TANDEM mode. In this mode the system produces a stop character (default ^S) when the input queue is in danger of overflowing, and a start character (default ^Q) when the input has drained sufficiently. This mode is useful when the terminal is actually another machine that obeys the conventions.

Line control and breaks.

There are several *ioctl* calls available to control the state of the terminal line. The TIOCSBRK ioctl will set the break bit in the hardware interface causing a break condition to exist; this can be cleared (usually after a delay with *sleep*(3)) by TIOCCBRK. Break conditions in the input are reflected as a null character in RAW mode or as the interrupt character in cooked or CBREAK mode. The TIOCCDTR ioctl will clear the data terminal ready condition; it can be set again by TIOCSDTR.

When the carrier signal from the dataset drops (usually because the user has hung up his terminal) a SIGHUP hangup signal is sent to the processes in the distinguished process group of the terminal; this usually causes them to terminate (the SIGHUP can be suppressed by setting the LNOHANG bit in the local state word of the driver.) Access to the terminal by other processes is then normally revoked, so any further reads will fail, and programs that read a terminal and test for end-of-file on their input will terminate appropriately.

When using an ACU it is possible to ask that the phone line be hung up on the last close with the TIOCHPCL ioctl; this is normally done on the outgoing line.

Interrupt characters.

There are several characters that generate interrupts in cooked and CBREAK mode; all are sent the processes in the control group of the terminal, as if a TIOCGPGRP ioctl were done to get the process group and then a *killpg*(2) system call were done, except that these characters also flush pending input and output when typed at a terminal (*à'la* TIOCFLUSH). The characters shown here are the defaults; the field names in the structures (given below) are also shown. The characters may be changed, although this is not often done.

^?	**t_intrc** (Delete) generates a SIGINT signal. This is the normal way to stop a process which is no longer interesting, or to regain control in an interactive program.
^\	**t_quitc** (FS) generates a SIGQUIT signal. This is used to cause a program to terminate and produce a core image, if possible, in the file **core** in the current directory.
^Z	**t_suspc** (EM) generates a SIGTSTP signal, which is used to suspend the current process group.
^Y	**t_dsuspc** (SUB) generates a SIGTSTP signal as ^Z does, but the signal is sent when a program attempts to read the ^Y, rather than when it is typed.

Job access control.

When using the new terminal driver, if a process which is not in the distinguished process group of its control terminal attempts to read from that terminal its process group is sent a SIGTTIN signal. This signal normally causes the members of that process group to stop. If, however, the process is ignoring SIGTTIN, has SIGTTIN blocked, is an *orphan process*, or is in the middle of process creation using *vfork*(2)), it is instead returned an end-of-file. (An *orphan process* is a process whose parent has exited and has been inherited by the *init*(8) process.) Under older UNIX systems these processes would typically have had their input files reset to **/dev/null,** so this is a compatible change.

When using the new terminal driver with the LTOSTOP bit set in the local modes, a process is prohibited from writing on its control terminal if it is not in the distinguished process group for that terminal. Processes which are holding or ignoring SIGTTOU signals, which are orphans, or which are in the middle of a *vfork*(2) are excepted and allowed to produce output.

Summary of modes.

Unfortunately, due to the evolution of the terminal driver, there are 4 different structures which contain various portions of the driver data. The first of these (**sgttyb**) contains that part of the information largely common between version 6 and version 7 UNIX systems. The second contains additional control characters added in version 7. The third is a word of local state peculiar to the new terminal driver, and the fourth is another structure of special characters added for the new driver. In the future a single structure may be made available to programs which need to access all this information; most programs need not concern themselves with all this state.

Basic modes: sgtty.

The basic *ioctls* use the structure defined in <*sgtty.h*>:

```
struct sgttyb {
        char    sg_ispeed;
        char    sg_ospeed;
        char    sg_erase;
        char    sg_kill;
        short   sg_flags;
};
```

The *sg_ispeed* and *sg_ospeed* fields describe the input and output speeds of the device according to the following table, which corresponds to the DEC DH-11 interface. If other hardware is used, impossible speed changes are ignored. Symbolic values in the table are as defined in <*sgtty.h*>.

B0	0	(hang up dataphone)
B50	1	50 baud
B75	2	75 baud
B110	3	110 baud
B134	4	134.5 baud
B150	5	150 baud
B200	6	200 baud
B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B1800	10	1800 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud
EXTA	14	External A
EXTB	15	External B

In the current configuration, only 110, 150, 300 and 1200 baud are really supported on dial-up lines. Code conversion and line control required for IBM 2741's (134.5 baud) must be implemented by the user's program. The half-duplex line discipline required for the 202 dataset (1200 baud) is not supplied; full-duplex 212 datasets work fine.

The *sg_erase* and *sg_kill* fields of the argument structure specify the erase and kill characters respectively. (Defaults are # and @.)

The *sg_flags* field of the argument structure contains several bits that determine the system's treatment of the terminal:

ALLDELAY	0177400	Delay algorithm selection
BSDELAY	0100000	Select backspace delays (not implemented):
BS0	0	
BS1	0100000	
VTDELAY	0040000	Select form-feed and vertical-tab delays:
FF0	0	
FF1	0100000	
CRDELAY	0030000	Select carriage-return delays:
CR0	0	
CR1	0010000	
CR2	0020000	
CR3	0030000	

TBDELAY	0006000	Select tab delays:
TAB0	0	
TAB1	0001000	
TAB2	0004000	
XTABS	0006000	
NLDELAY	0001400	Select new-line delays:
NL0	0	
NL1	0000400	
NL2	0001000	
NL3	0001400	
EVENP	0000200	Even parity allowed on input (most terminals)
ODDP	0000100	Odd parity allowed on input
RAW	0000040	Raw mode: wake up on all characters, 8-bit interface
CRMOD	0000020	Map CR into LF; echo LF or CR as CR-LF
ECHO	0000010	Echo (full duplex)
LCASE	0000004	Map upper case to lower on input
CBREAK	0000002	Return each character as soon as typed
TANDEM	0000001	Automatic flow control

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay.

Backspace delays are currently ignored but might be used for Terminet 300's.

If a form-feed/vertical tab delay is specified, it lasts for about 2 seconds.

Carriage-return delay type 1 lasts about .08 seconds and is suitable for the Terminet 300. Delay type 2 lasts about .16 seconds and is suitable for the VT05 and the TI 700. Delay type 3 is suitable for the concept-100 and pads lines to be at least 9 characters at 9600 baud.

New-line delay type 1 is dependent on the current column and is tuned for Teletype model 37's. Type 2 is useful for the VT05 and is about .10 seconds. Type 3 is unimplemented and is 0.

Tab delay type 1 is dependent on the amount of movement and is tuned to the Teletype model 37. Type 3, called XTABS, is not a delay at all but causes tabs to be replaced by the appropriate number of spaces on output.

Input characters with the wrong parity, as determined by bits 200 and 100, are ignored in cooked and CBREAK mode.

RAW disables all processing save output flushing with LFLUSHO; full 8 bits of input are given as soon as it is available; all 8 bits are passed on output. A break condition in the input is reported as a null character. If the input queue overflows in raw mode it is discarded; this applies to both new and old drivers.

CRMOD causes input carriage returns to be turned into new-lines; input of either CR or LF causes LF-CR both to be echoed (for terminals with a new-line function).

CBREAK is a sort of half-cooked (rare?) mode. Programs can read each character as soon as typed, instead of waiting for a full line; all processing is done except the input editing: character and word erase and line kill, input reprint, and the special treatment of \ or EOT are disabled.

TANDEM mode causes the system to produce a stop character (default ^S) whenever the input queue is in danger of overflowing, and a start character (default ^Q) when the input queue has drained sufficiently. It is useful for flow control when the 'terminal' is really another computer which understands the conventions.

Basic ioctls

In addition to the TIOCSETD and TIOCGETD disciplines discussed in **Line disciplines** above, a large number of other *ioctl*(2) calls apply to terminals, and have the general form:

#include <sgtty.h>

ioctl(fildes, code, arg)
struct sgttyb *arg;

The applicable codes are:

TIOCGETP	Fetch the basic parameters associated with the terminal, and store in the pointed-to *sgttyb* structure.
TIOCSETP	Set the parameters according to the pointed-to *sgttyb* structure. The interface delays until output is quiescent, then throws away any unread characters, before changing the modes.
TIOCSETN	Set the parameters like TIOCSETP but do not delay or flush input. Input is not preserved, however, when changing to or from RAW.

With the following codes the *arg* is ignored.

TIOCEXCL	Set "exclusive-use" mode: no further opens are permitted until the file has been closed.
TIOCNXCL	Turn off "exclusive-use" mode.
TIOCHPCL	When the file is closed for the last time, hang up the terminal. This is useful when the line is associated with an ACU used to place outgoing calls.
TIOCFLUSH	All characters waiting in input or output queues are flushed.

The remaining calls are not available in vanilla version 7 UNIX. In cases where arguments are required, they are described; *arg* should otherwise be given as 0.

TIOCSTI	the argument is the address of a character which the system pretends was typed on the terminal.
TIOCSBRK	the break bit is set in the terminal.
TIOCCBRK	the break bit is cleared.
TIOCSDTR	data terminal ready is set.
TIOCCDTR	data terminal ready is cleared.
TIOCGPGRP	arg is the address of a word into which is placed the process group number of the control terminal.
TIOCSPGRP	arg is a word (typically a process id) which becomes the process group for the control terminal.
FIONREAD	returns in the long integer whose address is arg the number of immediately readable characters from the argument unit. This works for files, pipes, and terminals, but not (yet) for multiplexed channels.

Tchars

The second structure associated with each terminal specifies characters that are special in both the old and new terminal interfaces: The following structure is defined in < *sys/ioctl.h*>, which is automatically included in < *sgtty.h*>:

```
struct tchars {
        char    t_intrc;        /* interrupt */
        char    t_quitc;        /* quit */
        char    t_startc;       /* start output */
        char    t_stopc;        /* stop output */
        char    t_eofc;         /* end-of-file */
        char    t_brkc;         /* input delimiter (like nl) */
};
```

The default values for these characters are ^?, ^\, ^Q, ^S, ^D, and −1. A character value of −1 eliminates the effect of that character. The *t_brkc* character, by default −1, acts like a new-line in that it terminates a 'line,' is echoed, and is passed to the program. The 'stop' and 'start' characters may be the same, to produce a toggle effect. It is probably counterproductive to make other special characters (including erase and kill) identical. The applicable ioctl calls are:

TIOCGETC Get the special characters and put them in the specified structure.

TIOCSETC Set the special characters to those given in the structure.

Local mode

The third structure associated with each terminal is a local mode word; except for the LNOHANG bit, this word is interpreted only when the new driver is in use. The bits of the local mode word are:

LCRTBS	000001	Backspace on erase rather than echoing erase
LPRTERA	000002	Printing terminal erase mode
LCRTERA	000004	Erase character echoes as backspace-space-backspace
LTILDE	000010	Convert ~ to ` on output (for Hazeltine terminals)
LMDMBUF	000020	Stop/start output when carrier drops
LLITOUT	000040	Suppress output translations
LTOSTOP	000100	Send SIGTTOU for background output
LFLUSHO	000200	Output is being flushed
LNOHANG	000400	Don't send hangup when carrier drops
LETXACK	001000	Diablo style buffer hacking (unimplemented)
LCRTKIL	002000	BS-space-BS erase entire line on line kill
LINTRUP	004000	Generate interrupt SIGTINT when input ready to read
LCTLECH	010000	Echo input control chars as ^X, delete as ^?
LPENDIN	020000	Retype pending input at next read or input character
LDECCTQ	040000	Only ^Q restarts output after ^S, like DEC systems

The applicable *ioctl* functions are:

TIOCLBIS arg is the address of a mask which is the bits to be set in the local mode word.

TIOCLBIC arg is the address of a mask of bits to be cleared in the local mode word.

TIOCLSET arg is the address of a mask to be placed in the local mode word.

TIOCLGET arg is the address of a word into which the current mask is placed.

Local special chars

The final structure associated with each terminal is the *ltchars* structure which defines interrupt characters for the new terminal driver. Its structure is:

```
struct ltchars {
        char    t_suspc;        /* stop process signal */
        char    t_dsuspc;       /* delayed stop process signal */
        char    t_rprntc;       /* reprint line */
        char    t_flushc;       /* flush output (toggles) */
        char    t_werasc;       /* word erase */
        char    t_lnextc;       /* literal next character */
};
```

The default values for these characters are ^Z, ^Y, ^R, ^O, ^W, and ^V. A value of −1 disables the character.

The applicable *ioctl* functions are:

TIOCSLTC args is the address of a *ltchars* structure which defines the new local special characters.

TIOCGLTC args is the address of a *ltchars* structure into which is placed the current set of local special characters.

FILES

/dev/tty
/dev/tty*
/dev/console

SEE ALSO

csh(1), stty(1), ioctl(2), sigvec(2), stty(3C), getty(8), init(8)

BUGS

Half-duplex terminals are not supported.

NAME

tu − VAX-11/730 and VAX-11/750 TU58 console cassette interface

SYNOPSIS

options MRSP (for VAX-11/750's with an MRSP prom)

DESCRIPTION

The *tu* interface provides access to the VAX 11/730 and 11/750 TU58 console cassette drive(s).

The interface supports only block i/o to the TU58 cassettes. The devices are normally manipulated with the *arff*(8V) program using the "f" and "m" options.

The device driver is automatically included when a system is configured to run on an 11/730 or 11/750.

The TU58 on an 11/750 uses the Radial Serial Protocol (RSP) to communicate with the cpu over a serial line. This protocol is inherently unreliable as it has no flow control measures built in. On an 11/730 the Modified Radial Serial Protocol is used. This protocol incorporates flow control measures which insure reliable data transfer between the cpu and the device. Certain 11/750's have been modified to use the MRSP prom used in the 11/730. To reliably use the console TU58 on an 11/750 under UNIX, the MRSP prom is required. For those 11/750's without an MRSP prom, an unreliable but often useable interface has been developed. This interface uses an assembly language "pseudo-dma" routine to minimize the receiver interrupt service latency. To include this code in the system, the configuration must **not** specify the system will run on an 11/730 or use an MRSP prom. This unfortunately makes it impossible to configure a single system which will properly handle TU58's on both an 11/750 and an 11/730 (unless both machines have MRSP proms).

FILES

/dev/tu0	
/dev/tu1	(only on a VAX-11/730)

SEE ALSO

arff(8V)

DIAGNOSTICS

tu%d: no bp, active %d. A transmission complete interrupt was received with no outstanding i/o request. This indicates a hardware problem.

tu%d protocol error, state=%s, op=%x, cnt=%d, block=%d. The driver entered an illegal state. The information printed indicates the illegal state, operation currently being executed, the i/o count, and the block number on the cassette.

tu%d receive state error, state=%s, byte=%x. The driver entered an illegal state in the receiver finite state machine. The state is shown along with the control byte of the received packet.

tu%d: read stalled. A timer watching the controller detected no interrupt for an extended period while an operation was outstanding. This usually indicates that one or more receiver interrupts were lost and the transfer is restarted (11/750 only).

tu%d: hard error bn%d, pk_mod %o. The device returned a status code indicating a hard error. The actual error code is shown in octal. No retries are attempted by the driver.

BUGS

The VAX-11/750 console interface without MRSP prom is unuseable while the system is multi-user; it should be used only with the system running single-user and, even then, with caution.

NAME

uda — UDA-50 disk controller interface

SYNOPSIS

controller uda0 at uba0 csr 0172150 vector udintr
disk ra0 at uda0 drive 0

DESCRIPTION

This is a driver for the DEC UDA-50 disk controller. The UDA-50 communicates with the host through a packet oriented protocol termed the Mass Storage Control Protocol (MSCP). Consult the file <*vax/mscp.h*> for a detailed description of this protocol.

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

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

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

DISK SUPPORT

This driver handles all drives which may be connected to the UDA-50. Drive types per se are not recognized, but rather the variable length partitions are defined as having an "infinite" length and the controller is relied on to return an error when an inaccessible block is requested. For constructing file systems, however the partitions sizes are required. The origin and size (in sectors) of the pseudo-disks on each drive are shown below. Partitions are not rounded to cylinder boundaries, as on other drives, because the type of drive attached to the controller is discovered too late in the autoconfiguration process to maintain separate partition tables for each drive. (The lack of proper drive type recognition would be more easily dealt with if the partition tables were read off the drive.)

RA60 partitions (Stanford-specific!)

disk	start	length
ra?a	0	15884
ra?b	15884	33440
ra?c	0	400176
ra?d	131304	200000
ra?e	331304	68872
ra?g	49324	82080
ra?h	131404	268772

RA80 partitions

disk	start	length
ra?a	0	15884
ra?b	15884	33440
ra?c	0	242606
ra?g	49324	82080
ra?h	131404	111202

RA81 partitions (Stanford-specific!)

disk	start	length
ra?a	0	15884

ra?b	15884	33440
ra?c	0	891072
ra?d	131304	200000
ra?e	331304	279884
ra?f	611188	279884
ra?g	49324	82080
ra?h	131404	759668

The ra?a partition is normally used for the root file system, the ra?b partition as a paging area, and the ra?c partition for pack-pack copying (it maps the entire disk).

FILES

/dev/ra[0-9][a-f]
/dev/rra[0-9][a-f]

DIAGNOSTICS

uda: ubinfo %x. (VAX 11/750 only.) When allocating UNIBUS resources, the driver found it already had resources previously allocated. This indicates a bug in the driver.

udasa %o, state %d. (Additional status information given after a hard i/o error.) The values of the UDA-50 status register and the internal driver state are printed.

uda%d: random interrupt ignored. An unexpected interrupt was received (e.g. when no i/o was pending). The interrupt is ignored.

uda%d: interrupt in unknown state %d ignored. An interrupt was received when the driver was in an unknown internal state. Indicates a hardware problem or a driver bug.

uda%d: fatal error (%o). The UDA-50 indicated a "fatal error" in the status returned to the host. The contents of the status register are displayed.

OFFLINE. (Additional status information given after a hard i/o error.) A hard i/o error occurred because the drive was not on-line.

status %o. (Additional status information given after a hard i/o error.) The status information returned from the UDA-50 is tacked onto the end of the hard error message printed on the console.

uda: unknown packet. An MSCP packet of unknown type was received from the UDA-50. Check the cabling to the controller.

The following errors are interpretations of MSCP error messages returned by the UDA-50 to the host.

uda%d: %s error, controller error, event 0%o.

uda%d: %s error, host memory access error, event 0%o, addr 0%o.

uda%d: %s error, disk transfer error, unit %d.

uda%d: %s error, SDI error, unit %d, event 0%o.

uda%d: %s error, small disk error, unit %d, event 0%o, cyl %d.

uda%d: %s error, unknown error, unit %d, format 0%o, event 0%o.

BUGS

The partition tables were so poorly laid out that they almost certainly forced each site to tailor them to their individual needs. The problem is even worse when a site has a mixed collection of drives. The best solution would be to read the partition tables off the drive.

The partitions in the Stanford version of the system are somewhat better, although they don't conform entirely with the results of *diskpart*(8).

NAME

udp − Internet User Datagram Protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

DESCRIPTION

UDP is a simple, unreliable datagram protocol which is used to support the SOCK_DGRAM abstraction for the Internet protocol family. UDP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls, though the *connect*(2) call may also be used to fix the destination for future packets (in which case the *recv*(2) or *read*(2) and *send*(2) or *write(2)* system calls may be used).

UDP address formats are identical to those used by TCP. In particular UDP provides a port identifier in addition to the normal Internet address format. Note that the UDP port space is separate from the TCP port space (i.e. a UDP port may not be "connected" to a TCP port). In addition broadcast packets may be sent (assuming the underlying network supports this) by using a reserved "broadcast address"; this address is network interface dependent.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
[ENOBUFS]	when the system runs out of memory for an internal data structure;
[EADDRINUSE]	when an attempt is made to create a socket with a port which has already been allocated;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

send(2), recv(2), intro(4N), inet(4F)

NAME

un — Ungermann-Bass interface

SYNOPSIS

device un0 at uba0 csr 0160210 vector unintr

DESCRIPTION

The *un* interface provides access to a 4 Mb/s baseband network. The hardware uses a standard DEC DR11-W DMA interface in communicating with the host. The Ungermann-Bass hardware incorporates substantial protocol software in the network device in an attempt to offload protocol processing from the host.

The network number on which the interface resides must be specified at boot time with an SIOCSIFADDR ioctl. The host's address is discovered by communicating with the interface. The interface will not transmit or receive any packets before the network number has been defined.

DIAGNOSTICS

un%d: can't initialize. Insufficient UNIBUS resources existed for the device to complete initialization. Usually caused by having multiple network interfaces configured using buffered data paths on a data path poor machine such as the 11/750.

un%d: unexpected reset. The controller indicated a reset when none had been requested. Check the hardware (but see the bugs section below).

un%d: stray interrupt. An unexpected interrupt was received. The interrupt was ignored.

un%d: input error csr=%b. The controller indicated an error on moving data from the device to host memory.

un%d: bad packet type %d. A packet was received with an unknown packet type. The packet is discarded.

un%d: output error csr=%b. The device indicated an error on moving data from the host to device memory.

un%d: invalid state %d csr=%b. The driver found itself in an invalid internal state. The state is reset to a base state.

un%d: can't handle af%d. A request was made to send a message with an address format which the driver does not understand. The message is discarded and an error is returned to the user.

un%d: error limit exceeded. Too many errors were encountered in normal operation. The driver will attempt to reset the device, desist from attempting any i/o for approximately 60 seconds, then reset itself to a base state in hopes of resyncing itself up with the hardware.

un%d: restarting. After exceeding its error limit and resetting the device, the driver is restarting operation.

SEE ALSO

intro(4N), inet(4F)

BUGS

The device does not reset itself properly resulting in the interface getting hung up in a state from which the only recourse is to reboot the system.

NAME

up − unibus storage module controller/drives

SYNOPSIS

controller sc0 at uba? csr 0176700 vector upintr
disk up0 at sc0 drive 0

DESCRIPTION

This is a generic UNIBUS storage module disk driver. It is specifically designed to work with the Emulex SC-21 controller. It can be easily adapted to other controllers (although bootstrapping will not necessarily be directly possible.)

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

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

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

DISK SUPPORT

The driver interrogates the controller's holding register to determine the type of drive attached. The driver recognizes four different drives: AMPEX 9300, CDC 9766, AMPEX Capricorn, and FUJITSU 160. The origin and size of the pseudo-disks on each drive are as follows:

CDC 9766 300M drive partitions:

disk	start	length	cyl
up?a	0	15884	0-26
up?b	16416	33440	27-81
up?c	0	500384	0-822
up?d	341696	15884	562-588
up?e	358112	55936	589-680
up?f	414048	861760	681-822
up?g	341696	158528	562-822
up?h	49856	291346	82-561

AMPEX 9300 300M drive partitions:

disk	start	length	cyl
up?a	0	15884	0-26
up?b	16416	33440	27-81
up?c	0	495520	0-814
up?d	341696	15884	562-588
up?e	358112	55936	589-680
up?f	414048	81312	681-814
up?g	341696	153664	562-814
up?h	49856	291346	82-561

AMPEX Capricorn 330M drive partitions:

disk	start	length	cyl
hp?a	0	15884	0-31
hp?b	16384	33440	32-97

hp?c	0	524288	0-1023
hp?d	342016	15884	668-699
hp?e	358400	55936	700-809
hp?f	414720	109408	810-1023
hp?g	342016	182112	668-1023
hp?h	50176	291346	98-667

FUJITSU 160M drive partitions:

disk	start	length	cyl
up?a	0	15884	0-49
up?b	16000	33440	50-154
up?c	0	263360	0-822
up?d	49600	15884	155-204
up?e	65600	55936	205-379
up?f	121600	141600	380-822
up?g	49600	213600	155-822

It is unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter. The up?a partition is normally used for the root file system, the up?b partition as a paging area, and the up?c partition for pack-pack copying (it maps the entire disk). On 160M drives the up?g partition maps the rest of the pack. On other drives both up?g and up?h are used to map the remaining cylinders.

FILES

/dev/up[0-7][a-h]	block files
/dev/rup[0-7][a-h]	raw files

SEE ALSO

hk(4), hp(4), uda(4)

DIAGNOSTICS

up%d%c: hard error sn%d cs2=%b er1=%b er2=%b. An unrecoverable error occurred during transfer of the specified sector in the specified disk partition. The contents of the cs2, er1 and er2 registers are printed in octal and symbolically with bits decoded. The error was either unrecoverable, or a large number of retry attempts (including offset positioning and drive recalibration) could not recover the error.

up%d: write locked. The write protect switch was set on the drive when a write was attempted. The write operation is not recoverable.

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

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

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

sc%d: lost interrupt. A timer watching the controller detecting no interrupt for an extended period while an operation was outstanding. This indicates a hardware or software failure. There is currently a hardware/software problem with spinning down drives while they are being accessed which causes this error to occur. The error causes a UNIBUS reset, and retry of the pending operations. If the controller continues to lose interrupts, this error will recur a few seconds later.

BUGS

In raw I/O *read* and *write*(2) truncate file offsets to 512-byte block boundaries, and *write* scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read, write* and *lseek*(2) should always deal in 512-byte multiples.

DEC-standard error logging should be supported.

A program to analyze the logged error information (even in its present reduced form) is needed.

The partition tables for the file systems should be read off of each pack, as they are never quite what any single installation would prefer, and this would make packs more portable.

NAME

ut − UNIBUS TU45 tri-density tape drive interface

SYNOPSIS

controller ut0 at uba0 csr 0172440 vector utintr
tape tj0 at ut0 drive 0

DESCRIPTION

The *ut* interface provides access to a standard tape drive interface as describe in *mtio*(4). Hardware implementing this on the VAX is typified by the System Industries SI 9700 tape subsystem. Tapes may be read or written at 800, 1600, and 6250 bpi.

SEE ALSO

mt(1), mtio(4)

DIAGNOSTICS

tj%d: no write ring. An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

tj%d: not online. An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

tj%d: can't change density in mid-tape. An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

ut%d: soft error bn%d cs1=%b er=%b cs2=%b ds=%b. The formatter indicated a corrected error at a density other than 800bpi. The data transferred is assumed to be correct.

ut%d: hard error bn%d cs1=%b er=%b cs2=%b ds=%b. A tape error occurred at block *bn*. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.

tj%d: lost interrupt. A tape operation did not complete within a reasonable time, most likely because the tape was taken off-line during rewind or lost vacuum. The controller should, but does not, give an interrupt in these cases. The device will be made available again after this message, but any current open reference to the device will return an error as the operation in progress aborts.

BUGS

If any non-data error is encountered on non-raw tape, it refuses to do anything more until closed.

NAME

uu − TU58/DECtape II UNIBUS cassette interface

SYNOPSIS

options UUDMA
device uu0 at uba0 csr 0176500 vector uurintr uuxintr

DESCRIPTION

The *uu* device provides access to dual DEC TU58 tape cartridge drives connected to the UNIBUS via a DL11-W interface module.

The interface supports only block i/o to the TU58 cassettes. The drives are normally manipulated with the *arff*(8V) program using the "m" and "f" options.

The driver provides for an optional write and verify (read after write) mode that is activated by specifying the "a" device.

The TU58 is treated as a single device by the system even though it has two separate drives, "uu0" and "uu1". If there is more than one TU58 unit on a system, the extra drives are named "uu2", "uu3" etc.

NOTES

Assembly language code to assist the driver in handling the receipt of data (using a pseudo-dma approach) should be included when using this driver; specify "options UUDMA" in the configuration file.

ERRORS

The following errors may be returned:

[ENXIO]	Nonexistent drive (on open); offset is too large or bad (undefined) ioctl code.
[EIO]	Open failed, the device could not be reset.
[EBUSY]	Drive in use.

FILES

/dev/uu?
/dev/uu?a

SEE ALSO

tu(4), arff(8V)

DIAGNOSTICS

uu%d: no bp, active %d. A transmission complete interrupt was received with no outstanding i/o request. This indicates a hardware problem.

uu%d protocol error, state=%s, op=%x, cnt=%d, block=%d. The driver entered an illegal state. The information printed indicates the illegal state, the operation currently being executed, the i/o count, and the block number on the cassette.

uu%d: break received, transfer restarted. The TU58 was sending a continuous break signal and had to be reset. This may indicate a hardware problem, but the driver will attempt to recover from the error.

uu%d receive state error, state=%s, byte=%x. The driver entered an illegal state in the receiver finite state machine. The state is shown along with the control byte of the received packet.

uu%d: read stalled. A timer watching the controller detected no interrupt for an extended period while an operation was outstanding. This usually indicates that one or more receiver interrupts were lost and the transfer is restarted.

uu%d: hard error bn%d, pk_mod %o. The device returned a status code indicating a hard error. The actual error code is shown in octal. No retries are attempted by the driver.

NAME

va − Benson-Varian interface

SYNOPSIS

controller va0 at uba0 csr 0164000 vector vaintr
disk vz0 at va0 drive 0

DESCRIPTION

(NOTE: the configuration description, while counter-intuitive, is actually as shown above.)

The Benson-Varian printer/plotter in normally used with the programs *vpr*(1), *vprint*(1) or *vtroff*(1). This description is designed for those who wish to drive the Benson-Varian directly.

In print mode, the Benson-Varian uses a modified ASCII character set. Most control characters print various non-ASCII graphics such as daggers, sigmas, copyright symbols, etc. Only LF and FF are used as format effectors. LF acts as a newline, advancing to the beginning of the next line, and FF advances to the top of the next page.

In plot mode, the Benson-Varian prints one raster line at a time. An entire raster line of bits (2112 bits = 264 bytes) is sent, and then the Benson-Varian advances to the next raster line.

Note: The Benson-Varian must be sent an even number of bytes. If an odd number is sent, the last byte will be lost. Nulls can be used in print mode to pad to an even number of bytes.

To use the Benson-Varian yourself, you must realize that you cannot open the device, */dev/va0* if there is a daemon active. You can see if there is an active daemon by doing a *lpq*(1) and seeing if there are any files being printed.

To set the Benson-Varian into plot mode include the file <*sys/vcmd.h*> and use the following *ioctl*(2) call

```
ioctl(fileno(va), VSETSTATE, plotmd);
```

where *plotmd* is defined to be

```
int plotmd[] = { VPLOT, 0, 0 };
```

and *va* is the result of a call to *fopen* on stdio. When you finish using the Benson-Varian in plot mode you should advance to a new page by sending it a FF after putting it back into print mode, i.e. by

```
int prtmd[] = { VPRINT, 0, 0 };
...
fflush(va);
ioctl(fileno(va), VSETSTATE, prtmd);
write(fileno(va), "\f\0", 2);
```

N.B.: If you use the standard I/O library with the Benson-Varian you **must** do

```
setbuf(vp, vpbuf);
```

where *vpbuf* is declared

```
char vpbuf[BUFSIZ];
```

otherwise the standard I/O library, thinking that the Benson-Varian is a terminal (since it is a character special file) will not adequately buffer the data you are sending to the Benson-Varian. This will cause it to run **extremely** slowly and tend to grind the system to a halt.

FILES

/dev/va0

SEE ALSO

vfont(5), lpr(1), lpd(8), vtroff(1), vp(4)

DIAGNOSTICS

The following error numbers are significant at the time the device is opened.

[ENXIO]	The device is already in use.
[EIO]	The device is offline.

The following message may be printed on the console.

va%d: npr timeout. The device was not able to get data from the UNIBUS within the timeout period, most likely because some other device was hogging the bus. (But see BUGS below).

BUGS

The 1's (one's) and l's (lower-case el's) in the Benson-Varian's standard character set look very similar; caution is advised.

The interface hardware is rumored to have problems which can play havoc with the UNIBUS. We have intermittent minor problems on the UNIBUS where our *va* lives, but haven't ever been able to pin them down completely.

NAME

vp − Versatec interface

SYNOPSIS

device vp0 at uba0 csr 0177510 vector vpintr vpintr

DESCRIPTION

The Versatec printer/plotter is normally used with the programs *vpr*(1), *vprint*(1) or *vtroff*(1). This description is designed for those who wish to drive the Versatec directly.

To use the Versatec yourself, you must realize that you cannot open the device, */dev/vp0* if there is a daemon active. You can see if there is a daemon active by doing a *lpq*(1), and seeing if there are any files being sent.

To set the Versatec into plot mode you should include <*sys/vcmd.h*> and use the *ioctl*(2) call

```
ioctl(fileno(vp), VSETSTATE, plotmd);
```

where *plotmd* is defined to be

```
int plotmd[] = { VPLOT, 0, 0 };
```

and *vp* is the result of a call to *fopen* on stdio. When you finish using the Versatec in plot mode you should eject paper by sending it a EOT after putting it back into print mode, i.e. by

```
int prtmd[] = { VPRINT, 0, 0 };
...
fflush(vp);
ioctl(fileno(vp), VSETSTATE, prtmd);
write(fileno(vp), "\04", 1);
```

N.B.: If you use the standard I/O library with the Versatec you **must** do

```
setbuf(vp, vpbuf);
```

where *vpbuf* is declared

```
char vpbuf[BUFSIZ];
```

otherwise the standard I/O library, thinking that the Versatec is a terminal (since it is a character special file) will not adequately buffer the data you are sending to the Versatec. This will cause it to run **extremely** slowly and tends to grind the system to a halt.

FILES

/dev/vp0

SEE ALSO

vfont(5), lpr(1), lpd(8), vtroff(1), va(4)

DIAGNOSTICS

The following error numbers are significant at the time the device is opened.

[ENXIO]	The device is already in use.
[EIO]	The device is offline.

BUGS

The configuration part of the driver assumes that the device is set up to vector print mode through 0174 and plot mode through 0200. As the configuration program can't be sure which vector interrupted at boot time, we specify that it has two interrupt vectors, and if an interrupt comes through 0200 it is reset to 0174. This is safe for devices with one or two vectors at these two addresses. Other configurations with 2 vectors may require changes in the driver.

NAME

vv – Proteon proNET 10 Megabit ring

SYNOPSIS

device vv0 at uba0 csr 161000 vector vvrint vvxint

DESCRIPTION

The *vv* interface provides access to a 10 Mb/s Proteon proNET ring network.

The network number to which the interface is attached must be specified with an SIOCSIFADDR ioctl before data can be transmitted or received. The host's address is discovered by putting the interface in digital loopback mode (not joining the ring) and sending a broadcast packet from which the source address is extracted. the Internet address of the interface would be 128.3.0.24.

The interface software implements error-rate limiting on the input side. This provides a defense against situations where other hosts or interface hardware failures cause a machine to be inundated with garbage packets. The scheme involves an exponential backoff where the input side of the interface is disabled for longer and longer periods. In the limiting case, the interface is turned on every two minutes or so to see if operation can resume.

If the installation is running CTL boards which use the old broadcast address of 0 instead of the new address of 0xff, the define OLD_BROADCAST should be specified in the driver.

If the installation has a Wirecenter, the define WIRECENTER should be specified in the driver. N.B.: Incorrect definition of WIRECENTER can cause hardware damage.

The interface normally tries to use a "trailer" encapsulation to minimize copying data on input and output. This may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS

vv%d: host %d. The software announces the host address discovered during autoconfiguration.

vv%d: can't initialize. The software was unable to discover the address of this interface, so it deemed "dead" will not be enabled.

vv%d: error vvocsr=%b. The hardware indicated an error on the previous transmission.

vv%d: output timeout. The token timer has fired and the token will be recreated.

vv%d: error vvicsr=%b. The hardware indicated an error in reading a packet off the ring.

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

vv%d: vs_olen=%d. The ring output routine has been handed a message with a preposterous length. This results in an immediate *panic: vs_olen.*

SEE ALSO

intro(4N), inet(4F)

NAME

a.out — assembler and link editor output

SYNOPSIS

#include <a.out.h>

DESCRIPTION

A.out is the output file of the assembler *as*(1) and the link editor *ld*(1). Both programs make *a.out* executable if there were no errors and no unresolved external references. Layout information as given in the include file for the VAX-11 is:

```
/*
 * Header prepended to each a.out file.
 */
struct exec {
        long      a_magic;  /* magic number */
        unsigned  a_text;   /* size of text segment */
        unsigned  a_data;   /* size of initialized data */
        unsigned  a_bss;    /* size of uninitialized data */
        unsigned  a_syms;   /* size of symbol table */
        unsigned  a_entry;  /* entry point */
        unsigned  a_trsize; /* size of text relocation */
        unsigned  a_drsize; /* size of data relocation */
};

#define  OMAGIC 0407        /* old impure format */
#define  NMAGIC 0410        /* read-only text */
#define  ZMAGIC 0413        /* demand load format */

/*
 * Macros which take exec structures as arguments and tell whether
 * the file has a reasonable magic number or offsets to text|symbols|strings.
 */
#define  N_BADMAG(x) \
   (((x).a_magic)!=OMAGIC && ((x).a_magic)!=NMAGIC && ((x).a_magic)!=ZMAGIC)

#define  N_TXTOFF(x) \
        ((x).a_magic==ZMAGIC ? 1024 : sizeof (struct exec))
#define N_SYMOFF(x) \
        (N_TXTOFF(x) + (x).a_text+(x).a_data + (x).a_trsize+(x).a_drsize)
#define  N_STROFF(x) \
        (N_SYMOFF(x) + (x).a_syms)
```

The file has five sections: a header, the program text and data, relocation information, a symbol table and a string table (in that order). The last three may be omitted if the program was loaded with the '−s' option of *ld* or if the symbols and relocation have been removed by *strip*(1).

In the header the sizes of each section are given in bytes. The size of the header is not included in any of the other sizes.

When an *a.out* file is executed, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized), and a stack. The text segment begins at 0 in the core image; the header is not loaded. If the magic number in the header is OMAGIC (0407), it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. This is the oldest kind of executable program and is rarely used. If the magic number is NMAGIC (0410) or ZMAGIC (0413), the data segment begins at the first 0 mod 1024 byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same file, they will share the text segment. For ZMAGIC format, the text segment begins at a 0 mod 1024 byte boundary in the *a.out* file, the remaining bytes after the header in the first block are reserved and should be zero. In this case the text and data sizes must both be multiples of 1024 bytes, and the pages of the file will be brought into the running image as needed, and not pre-loaded as with the other formats. This is especially suitable for very large programs and is the default format produced by *ld*(1).

The stack will occupy the highest possible locations in the core image: growing downwards from 0x7ffff000. The stack is automatically extended as required. The data segment is only extended as requested by *brk*(2).

After the header in the file follow the text, data, text relocation data relocation, symbol table and string table in that order. The text begins at the byte 1024 in the file for ZMAGIC format or just after the header for the other formats. The N_TXTOFF macro returns this absolute file position when given the name of an exec structure as argument. The data segment is contiguous with the text and immediately followed by the text relocation and then the data relocation information. The symbol table follows all this; its position is computed by the N_SYMOFF macro. Finally, the string table immediately follows the symbol table at a position which can be gotten easily using N_STROFF. The first 4 bytes of the string table are not used for string storage, but rather contain the size of the string table; this size INCLUDES the 4 bytes, the minimum string table size is thus 4.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file as follows:

```
/*
 * Format of a symbol table entry.
 */
struct nlist {
        union {
            char        *n_name;  /* for use when in-core */
            long        n_strx;   /* index into file string table */
        } n_un;
        unsigned char n_type;     /* type flag, i.e. N_TEXT etc; see below */
        char          n_other;
        short         n_desc;     /* see <stab.h> */
        unsigned      n_value;    /* value of this symbol (or offset) */
};
#define n_hash        n_desc      /* used internally by ld */

/*
 * Simple values for n_type.
 */
#define N_UNDF      0x0         /* undefined */
#define N_ABS       0x2         /* absolute */
#define N_TEXT      0x4         /* text */
#define N_DATA      0x6         /* data */
#define N_BSS       0x8         /* bss */
#define N_COMM      0x12        /* common (internal to ld) */
#define N_FN        0x1f        /* file name symbol */

#define N_EXT       01          /* external bit, or'ed in */
```

```
#define N_TYPE      0x1e        /* mask for all the type bits */

/*
 * Other permanent symbol table entries have some of the N_STAB bits set.
 * These are given in <stab.h>
 */
#define N_STAB      0xe0        /* if any of these bits set, don't discard */

/*
 * Format for namelist values.
 */
#define N_FORMAT "%08x"
```

In the *a.out* file a symbol's n_un.n_strx field gives an index into the string table. A n_strx value of 0 indicates that no name is associated with a particular symbol table entry. The field n_un.n_name can be used to refer to the symbol name only if the program sets this up using n_strx and appropriate data from the string table.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader *ld* as the name of a common region whose size is indicated by the value of the symbol.

The value of a byte in the text or data which is not a portion of a reference to an undefined external symbol is exactly that value which will appear in memory when the file is executed. If a byte in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the bytes in the file.

If relocation information is present, it amounts to eight bytes per relocatable datum as in the following structure:

```
/*
 * Format of a relocation datum.
 */
struct relocation_info {
        int       r_address;          /* address which is relocated */
        unsigned  r_symbolnum:24,     /* local symbol ordinal */
                  r_pcrel:1,          /* was relocated pc relative already */
                  r_length:2,         /* 0=byte, 1=word, 2=long */
                  r_extern:1,         /* does not include value of sym referenced */
                  :4;                 /* nothing, yet */
};
```

There is no relocation information if a_trsize+a_drsize==0. If r_extern is 0, then r_symbolnum is actually a n_type for the relocation (i.e. N_TEXT meaning relative to segment text origin.)

SEE ALSO

adb(1), as(1), ld(1), nm(1), dbx(1), stab(5), strip(1)

BUGS

Not having the size of the string table in the header is a loss, but expanding the header size would have meant stripped executable file incompatibility, and we couldn't hack this just now.

NAME

acct − execution accounting file

SYNOPSIS

#include <sys/acct.h>

DESCRIPTION

The *acct*(2) system call makes entries in an accounting file for each process that terminates. The accounting file is a sequence of entries whose layout, as defined by the include file is:

```
/*      acct.h      4.5         82/10/10*/

/*
 * Accounting structures;
 * these use a comp_t type which is a 3 bits base 8
 * exponent, 13 bit fraction "floating point" number.
 */
typedef u_short comp_t;

struct  acct
{
        char        ac_comm[10];  /* Accounting command name */
        comp_t      ac_utime;     /* Accounting user time */
        comp_t      ac_stime;     /* Accounting system time */
        comp_t      ac_etime;     /* Accounting elapsed time */
        time_t      ac_btime;     /* Beginning time */
        short       ac_uid;       /* Accounting user ID */
        short       ac_gid;       /* Accounting group ID */
        short       ac_mem;       /* average memory usage */
        comp_t      ac_io;        /* number of disk IO blocks */
        dev_t       ac_tty;       /* control typewriter */
        char        ac_flag;      /* Accounting flag */
};

#define AFORK       0001          /* has executed fork, but no exec */
#define ASU         0002          /* used super-user privileges */
#define ACOMPAT     0004          /* used compatibility mode */
#define ACORE       0010          /* dumped core */
#define AXSIG       0020          /* killed by a signal */

#define ACCTLO      30            /* acctg off when space < this */
#define ACCTHI      100           /* acctg resumes at this level */

#ifdef KERNEL
struct  acct        acctbuf;
struct  inode       *acctp;
#endif
```

If the process does an *execve*(2), the first 10 characters of the filename appear in *ac_comm*. The accounting flag contains bits indicating whether *execve*(2) was ever accomplished, and whether the process ever had super-user privileges.

SEE ALSO

acct(2), execve(2), sa(8)

NAME

aliases — aliases file for sendmail

SYNOPSIS

/usr/lib/aliases

DESCRIPTION

This file describes user id aliases used by */usr/lib/sendmail.* It is formatted as a series of lines of the form

    name: name_1, name2, name_3, . . .

The *name* is the name to alias, and the *name_n* are the aliases for that name. Lines beginning with white space are continuation lines. Lines beginning with '#' are comments.

Aliasing occurs only on local names. Loops can not occur, since no message will be sent to any person more than once.

After aliasing has been done, local and valid recipients who have a ".forward" file in their home directory have messages forwarded to the list of users defined in that file.

This is only the raw data file; the actual aliasing information is placed into a binary format in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag* using the program *newaliases*(1). A *newaliases* command should be executed each time the aliases file is changed for the change to take effect.

SEE ALSO

newaliases(1), dbm(3X), sendmail(8)
SENDMAIL Installation and Operation Guide.
SENDMAIL An Internetwork Mail Router.

BUGS

Because of restrictions in *dbm*(3X) a single alias cannot contain more than about 1000 bytes of information. You can get longer aliases by "chaining"; that is, make the last name in the alias be a dummy name which is a continuation alias.

NAME

ar − archive (library) file format

SYNOPSIS

#include <ar.h>

DESCRIPTION

The archive command *ar* combines several files into one. Archives are used mainly as libraries to be searched by the link-editor *ld.*

A file produced by *ar* has a magic string at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
#define ARMAG  "!<arch>\n"
#define SARMAG 8

#define ARFMAG "`\n"

struct ar_hdr {
        char        ar_name[16];
        char        ar_date[12];
        char        ar_uid[6];
        char        ar_gid[6];
        char        ar_mode[8];
        char        ar_size[10];
        char        ar_fmag[2];
};
```

The name is a blank-padded string. The *ar_fmag* field contains ARFMAG to help verify the presence of a header. The other fields are left-adjusted, blank-padded numbers. They are decimal except for *ar_mode*, which is octal. The date is the modification date of the file at the time of its insertion into the archive.

Each file begins on a even (0 mod 2) boundary; a new-line is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

There is no provision for empty areas in an archive file.

The encoding of the header is portable across machines. If an archive contains printable files, the archive itself is printable.

SEE ALSO

ar(1), ld(1), nm(1)

BUGS

File names lose trailing blanks. Most software dealing with archives takes even an included blank as a name terminator.

NAME

core — format of memory image file

SYNOPSIS

#include <machine/param.h>

DESCRIPTION

The UNIX System writes out a memory image of a terminated process when any of various errors occur. See *sigvec*(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The memory image is called 'core' and is written in the process's working directory (provided it can be; normal access controls apply).

The maximum size of a *core* file is limited by *setrlimit*(2). Files which would be larger than the limit are not created.

The core file consists of the *u.* area, whose size (in pages) is defined by the UPAGES manifest in the *<machine/param.h>* file. The *u.* area starts with a *user* structure as given in *<sys/user.h>*. The remainder of the core file consists first of the data pages and then the stack pages of the process image. The amount of data space image in the core file is given (in pages) by the variable *u_dsize* in the *u.* area. The amount of stack image in the core file is given (in pages) by the variable *u_ssize* in the *u.* area.

In general the debugger *adb*(1) is sufficient to deal with core images.

SEE ALSO

adb(1), dbx(1), sigvec(2), setrlimit(2)

NAME

dir — format of directories

SYNOPSIS

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

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry; see *fs*(5). The structure of a directory entry as given in the include file is:

```
/*
 * A directory consists of some number of blocks of DIRBLKSIZ
 * bytes, where DIRBLKSIZ is chosen such that it can be transferred
 * to disk in a single atomic operation (e.g. 512 bytes on most machines).
 *
 * Each DIRBLKSIZ byte block contains some number of directory entry
 * structures, which are of variable length. Each directory entry has
 * a struct direct at the front of it, containing its inode number,
 * the length of the entry, and the length of the name contained in
 * the entry. These are followed by the name padded to a 4 byte boundary
 * with null bytes. All names are guaranteed null terminated.
 * The maximum length of a name in a directory is MAXNAMLEN.
 *
 * The macro DIRSIZ(dp) gives the amount of space required to represent
 * a directory entry. Free space in a directory is represented by
 * entries which have dp->d_reclen > DIRSIZ(dp). All DIRBLKSIZ bytes
 * in a directory block are claimed by the directory entries. This
 * usually results in the last entry in a directory having a large
 * dp->d_reclen. When entries are deleted from a directory, the
 * space is returned to the previous entry in the same directory
 * block by increasing its dp->d_reclen. If the first entry of
 * a directory block is free, then its dp->d_ino is set to 0.
 * Entries other than the first in a directory do not normally have
 * dp->d_ino set to 0.
 */
#ifdef KERNEL
#define DIRBLKSIZ DEV_BSIZE
#else
#define DIRBLKSIZ 512
#endif

#define MAXNAMLEN 255

/*
 * The DIRSIZ macro gives the minimum record length which will hold
 * the directory entry. This requires the amount of space in struct direct
 * without the d_name field, plus enough space for the name with a terminating
 * null byte (dp->d_namlen+1), rounded up to a 4 byte boundary.
 */
#undef DIRSIZ
#define DIRSIZ(dp) \
   ((sizeof (struct direct) - (MAXNAMLEN+1)) + (((dp)->d_namlen+1 + 3) &~ 3))
```

```
struct  direct {
        u_long    d_ino;
        short     d_reclen;
        short     d_namlen;
        char      d_name[MAXNAMLEN + 1];
        /* typically shorter */
};

struct _dirdesc {
        int       dd_fd;
        long      dd_loc;
        long      dd_size;
        char      dd_buf[DIRBLKSIZ];
};
```

By convention, the first two entries in each directory are for '.' and '..'. The first is an entry for the directory itself. The second is for the parent directory. The meaning of '..' is modified for the root directory of the master file system ("/"), where '..' has the same meaning as '.'.

SEE ALSO

fs(5)

NAME

disktab — disk description file

SYNOPSIS

#include <disktab.h>

DESCRIPTION

Disktab is a simple date base which describes disk geometries and disk partition characteristics. The format is patterned after the *termcap*(5) terminal data base. Entries in *disktab* consist of a number of ':' separated fields. The first entry for each disk gives the names which are known for the disk, separated by '|' characters. The last name given should be a long name fully identifying the disk.

The following list indicates the normal values stored for each disk entry.

Name	**Type**	**Description**
ns	num	Number of sectors per track
nt	num	Number of tracks per cylinder
nc	num	Total number of cylinders on the disk
ba	num	Block size for partition 'a' (bytes)
bd	num	Block size for partition 'd' (bytes)
be	num	Block size for partition 'e' (bytes)
bf	num	Block size for partition 'f' (bytes)
bg	num	Block size for partition 'g' (bytes)
bh	num	Block size for partition 'h' (bytes)
fa	num	Fragment size for partition 'a' (bytes)
fd	num	Fragment size for partition 'd' (bytes)
fe	num	Fragment size for partition 'e' (bytes)
ff	num	Fragment size for partition 'f' (bytes)
fg	num	Fragment size for partition 'g' (bytes)
fh	num	Fragment size for partition 'h' (bytes)
pa	num	Size of partition 'a' in sectors
pb	num	Size of partition 'b' in sectors
pc	num	Size of partition 'c' in sectors
pd	num	Size of partition 'd' in sectors
pe	num	Size of partition 'e' in sectors
pf	num	Size of partition 'f' in sectors
pg	num	Size of partition 'g' in sectors
ph	num	Size of partition 'h' in sectors
se	num	Sector size in bytes
ty	str	Type of disk (e.g. removable, winchester)

Disktab entries may be automatically generated with the *diskpart* program.

FILES

/etc/disktab

SEE ALSO

newfs(8), diskpart(8)

BUGS

This file shouldn't exist, the information should be stored on each disk pack.

NAME

dump, dumpdates − incremental dump format

SYNOPSIS

#include <sys/types.h>
#include <sys/inode.h>
#include <dumprestor.h>

DESCRIPTION

Tapes used by *dump* and *restore*(8) contain:

a header record
two groups of bit map records
a group of records describing directories
a group of records describing files

The format of the header record and of the first record of each description as given in the include file <*dumprestor.h*> is:

```
#define NTREC       10
#define MLEN        16
#define MSIZ        4096

#define TS_TAPE     1
#define TS_INODE    2
#define TS_BITS     3
#define TS_ADDR     4
#define TS_END      5
#define TS_CLRI     6
#define MAGIC       (int) 60011
#define CHECKSUM    (int) 84446

struct  spcl {
        int         c_type;
        time_t      c_date;
        time_t      c_ddate;
        int         c_volume;
        daddr_t     c_tapea;
        ino_t       c_inumber;
        int         c_magic;
        int         c_checksum;
        struct      dinode        c_dinode;
        int         c_count;
        char        c_addr[BSIZE];
} spcl;

struct  idates {
        char        id_name[16];
        char        id_incno;
        time_t      id_ddate;
};

#define DUMPOUTFMT      "%-16s %c %s"        /* for printf */
                                      /* name, incno, ctime(date) */
#define DUMPINFMT  "%16s %c %[^\n]\n"    /* inverse for scanf */
```

NTREC is the number of 1024 byte records in a physical tape block. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the *c_type* field to indicate what sort of header this is. The types and their meanings are as follows:

TS_TAPE	Tape volume label
TS_INODE	A file or directory follows. The *c_dinode* field is a copy of the disk inode and contains bits telling what sort of file this is.
TS_BITS	A bit map follows. This bit map has a one bit for each inode that was dumped.
TS_ADDR	A subrecord of a file description. See *c_addr* below.
TS_END	End of tape record.
TS_CLRI	A bit map follows. This bit map contains a zero bit for all inodes that were empty on the file system when dumped.
MAGIC	All header records have this number in *c_magic*.
CHECKSUM	Header records checksum to this value.

The fields of the header structure are as follows:

c_type	The type of the header.
c_date	The date the dump was taken.
c_ddate	The date the file system was dumped from.
c_volume	The current volume number of the dump.
c_tapea	The current number of this (1024-byte) record.
c_inumber	The number of the inode being dumped if this is of type TS_INODE.
c_magic	This contains the value MAGIC above, truncated as needed.
c_checksum	This contains whatever value is needed to make the record sum to CHECKSUM.
c_dinode	This is a copy of the inode as it appears on the file system; see *fs*(5).
c_count	The count of characters in *c_addr*.
c_addr	An array of characters describing the blocks of the dumped file. A character is zero if the block associated with that character was not present on the file system, otherwise the character is non-zero. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, TS_ADDR records will be scattered through the file, each one picking up where the last left off.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a TS_END record and then the tapemark.

The structure *idates* describes an entry in the file */etc/dumpdates* where dump history is kept. The fields of the structure are:

id_name	The dumped filesystem is '/dev/ *id_nam*'.
id_incno	The level number of the dump tape; see *dump*(8).
id_ddate	The date of the incremental dump in system format see *types*(5).

FILES

/etc/dumpdates

SEE ALSO

dump(8), restore(8), fs(5), types(5)

NAME

fs, inode − format of file system volume

SYNOPSIS

#include <sys/types.h>
#include <sys/fs.h>
#include <sys/inode.h>

DESCRIPTION

Every file system storage volume (disk, nine-track tape, for instance) has a common format for certain vital information. Every such volume is divided into a certain number of blocks. The block size is a parameter of the file system. Sectors 0 to 15 on a file system are used to contain primary and secondary bootstrapping programs.

The actual file system begins at sector 16 with the *super block.* The layout of the super block as defined by the include file <*sys/fs.h*> is:

```
#define FS_MAGIC    0x011954
struct  fs {
        struct  fs *fs_link;            /* linked list of file systems */
        struct  fs *fs_rlink;           /*    used for incore super blocks */
        daddr_t fs_sblkno;              /* addr of super-block in filesys */
        daddr_t fs_cblkno;              /* offset of cyl-block in filesys */
        daddr_t fs_iblkno;              /* offset of inode-blocks in filesys */
        daddr_t fs_dblkno;              /* offset of first data after cg */
        long    fs_cgoffset;            /* cylinder group offset in cylinder */
        long    fs_cgmask;              /* used to calc mod fs_ntrak */
        time_t  fs_time;                /* last time written */
        long    fs_size;        /* number of blocks in fs */
        long    fs_dsize;               /* number of data blocks in fs */
        long    fs_ncg;                 /* number of cylinder groups */
        long    fs_bsize;               /* size of basic blocks in fs */
        long    fs_fsize;               /* size of frag blocks in fs */
        long    fs_frag;        /* number of frags in a block in fs */
/* these are configuration parameters */
        long    fs_minfree;             /* minimum percentage of free blocks */
        long    fs_rotdelay;            /* num of ms for optimal next block */
        long    fs_rps;                 /* disk revolutions per second */
/* these fields can be computed from the others */
        long    fs_bmask;               /* ``blkoff'' calc of blk offsets */
        long    fs_fmask;               /* ``fragoff'' calc of frag offsets */
        long    fs_bshift;              /* ``lblkno'' calc of logical blkno */
        long    fs_fshift;              /* ``numfrags'' calc number of frags */
/* these are configuration parameters */
        long    fs_maxcontig;           /* max number of contiguous blks */
        long    fs_maxbpg;              /* max number of blks per cyl group */
/* these fields can be computed from the others */
        long    fs_fragshift;           /* block to frag shift */
        long    fs_fsbtodb;             /* fsbtodb and dbtofsb shift constant */
        long    fs_sbsize;              /* actual size of super block */
        long    fs_csmask;              /* csum block offset */
        long    fs_csshift;             /* csum block number */
        long    fs_nindir;              /* value of NINDIR */
        long    fs_inopb;               /* value of INOPB */
        long    fs_nspf;        /* value of NSPF */
        long    fs_sparecon[6];         /* reserved for future constants */
/* sizes determined by number of cylinder groups and their sizes */
        daddr_t fs_csaddr;              /* blk addr of cyl grp summary area */
        long    fs_cssize;              /* size of cyl grp summary area */
        long    fs_cgsize;              /* cylinder group size */
/* these fields should be derived from the hardware */
        long    fs_ntrak;               /* tracks per cylinder */
        long    fs_nsect;               /* sectors per track */
        long    fs_spc;                 /* sectors per cylinder */
/* this comes from the disk driver partitioning */
        long    fs_ncyl;                /* cylinders in file system */
/* these fields can be computed from the others */
        long    fs_cpg;                 /* cylinders per group */
        long    fs_ipg;                 /* inodes per group */
        long    fs_fpg;                 /* blocks per group * fs_frag */
/* this data must be re-computed after crashes */
        struct  csum fs_cstotal;/* cylinder summary information */
/* these fields are cleared at mount time */
        char    fs_fmod;                /* super block modified flag */
        char    fs_clean;               /* file system is clean flag */
        char    fs_ronly;               /* mounted read-only flag */
        char    fs_flags;               /* currently unused flag */
        char    fs_fsmnt[MAXMNTLEN];            /* name mounted on */
/* these fields retain the current block allocation info */
        long    fs_cgrotor;             /* last cg searched */
        struct  csum *fs_csp[MAXCSBUFS];/* list of fs_cs info buffers */
        long    fs_cpc;                 /* cyl per cycle in postbl */
        short   fs_postbl[MAXCPG][NRPOS];/* head of blocks for each rotation */
        long    fs_magic;               /* magic number */
        u_char  fs_rotbl[1];            /* list of blocks for each rotation */
/* actually longer */
};
```

Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A file system is described by its super-block, which in turn describes the cylinder groups. The super-block is critical data and is replicated in each cylinder group to protect against catastrophic loss. This is done at file system creation time and the critical super-block data does not change, so the copies need not be referenced further unless disaster strikes.

Addresses stored in inodes are capable of addressing fragments of 'blocks'. File system blocks of at most size MAXBSIZE can be optionally broken into 2, 4, or 8 pieces, each of which is addressable; these pieces may be DEV_BSIZE, or some multiple of a DEV_BSIZE unit.

Large files consist of exclusively large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated as only as many fragments of a large block as are necessary. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determinable from information in the inode, using the "blksize(fs, ip, lbn)" macro.

The file system records space availability at the fragment level; to determine block availability, aligned fragments are examined.

The root inode is the root of the file system. Inode 0 can't be used for normal purposes and historically bad blocks were linked to inode 1, thus the root inode is 2 (inode 1 is no longer used for this purpose, however numerous dump tapes make this assumption, so we are stuck with it). The *lost+found* directory is given the next available inode when it is initially created by *mkfs*.

fs_minfree gives the minimum acceptable percentage of file system blocks which may be free. If the freelist drops below this level only the super-user may continue to allocate blocks. This may be set to 0 if no reserve of free blocks is deemed necessary, however severe performance degradations will be observed if the file system is run at greater than 90% full; thus the default value of *fs_minfree* is 10%.

Empirically the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 4, thus the default fragment size is a fourth of the block size.

Cylinder group related limits: Each cylinder keeps track of the availability of blocks at different rotational positions, so that sequential blocks can be laid out with minimum rotational latency. NRPOS is the number of rotational positions which are distinguished. With NRPOS 8 the resolution of the summary information is 2ms for a typical 3600 rpm drive.

fs_rotdelay gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for *fs_rotdelay* is 2ms.

Each file system has a statically allocated number of inodes. An inode is allocated for each NBPI bytes of disk space. The inode allocation strategy is extremely conservative.

MAXIPG bounds the number of inodes per cylinder group, and is needed only to keep the structure simpler by having the only a single variable size element (the free bit map).

N.B.: MAXIPG must be a multiple of INOPB(fs).

MINBSIZE is the smallest allowable block size. With a MINBSIZE of 4096 it is possible to create files of size 2^32 with only two levels of indirection. MINBSIZE must be big enough to hold a cylinder group block, thus changes to (struct cg) must keep its size within MINBSIZE. MAXCPG is limited only to dimension an array in (struct cg); it can be made larger as long as that structure's size remains within the bounds dictated by MINBSIZE. Note that super blocks are never more than size SBSIZE.

The path name on which the file system is mounted is maintained in *fs_fsmnt*. MAXMNTLEN defines the amount of space allocated in the super block for this name. The limit on the amount of summary information per file system is defined by MAXCSBUFS. It is currently parameterized for a maximum of two million cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from *fs_csaddr* (size *fs_cssize*) in addition to the super block.

N.B.: sizeof (struct csum) must be a power of two in order for the "fs_cs" macro to work.

Super block for a file system: MAXBPC bounds the size of the rotational layout tables and is limited by the fact that the super block is of size SBSIZE. The size of these tables is **inversely** proportional to the block size of the file system. The size of the tables is increased when sector sizes are not powers of two, as this increases the number of cylinders included before the rotational pattern repeats (*fs_cpc*). The size of the rotational layout tables is derived from the number of bytes remaining in (struct fs).

MAXBPG bounds the number of blocks of data per cylinder group, and is limited by the fact that cylinder groups are at most one block. The size of the free block table is derived from the size of blocks and the number of remaining bytes in the cylinder group structure (struct cg).

Inode: The inode is the focus of all file activity in the UNIX file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is 'named' by its device/i-number pair. For further information, see the include file <*sys/inode.h*>.

NAME

fstab − static information about the filesystems

SYNOPSIS

#include <fstab.h>

DESCRIPTION

The file */etc/fstab* contains descriptive information about the various file systems. */etc/fstab* is only *read* by programs, and not written; it is the duty of the system administrator to properly create and maintain this file. The order of records in */etc/fstab* is important because *fsck, mount,* and *umount* sequentially iterate through */etc/fstab* doing their thing.

The special file name is the **block** special file name, and not the character special file name. If a program needs the character special file name, the program must create it by appending a "r" after the last "/" in the special file name.

If *fs_type* is "rw" or "ro" then the file system whose name is given in the *fs_file* field is normally mounted read-write or read-only on the specified special file. If *fs_type* is "rq", then the file system is normally mounted read-write with disk quotas enabled. The *fs_freq* field is used for these file systems by the *dump*(8) command to determine which file systems need to be dumped. The *fs_passno* field is used by the *fsck*(8) program to determine the order in which file system checks are done at reboot time. The root file system should be specified with a *fs_passno* of 1, and other file systems should have larger numbers. File systems within a drive should have distinct numbers, but file systems on different drives can be checked on the same pass to utilize parallelism available in the hardware.

If *fs_type* is "sw" then the special file is made available as a piece of swap space by the *swapon*(8) command at the end of the system reboot procedure. The fields other than *fs_spec* and *fs_type* are not used in this case.

If *fs_type* is "rq" then at boot time the file system is automatically processed by the *quotacheck*(8) command and disk quotas are then enabled with *quotaon*(8). File system quotas are maintained in a file "quotas", which is located at the root of the associated file system.

If *fs_type* is specified as "xx" the entry is ignored. This is useful to show disk partitions which are currently not used.

```
#define FSTAB_RW    "rw"    /* read-write device */
#define FSTAB_RO    "ro"    /* read-only device */
#define FSTAB_RQ    "rq"    /* read-write with quotas */
#define FSTAB_SW    "sw"    /* swap device */
#define FSTAB_XX    "xx"    /* ignore totally */

struct fstab {
        char  *fs_spec;  /* block special device name */
        char  *fs_file;  /* file system path prefix */
        char  *fs_type;  /* rw,ro,sw or xx */
        int   fs_freq;   /* dump frequency, in days */
        int   fs_passno; /* pass number on parallel dump */
};
```

The proper way to read records from */etc/fstab* is to use the routines getfsent(), getfsspec(), getfstype(), and getfsfile().

FILES

/etc/fstab

SEE ALSO

getfsent(3X)

NAME

gettytab − terminal configuration data base

SYNOPSIS

/etc/gettytab

DESCRIPTION

Gettytab is a simplified version of the *termcap*(5) data base used to describe terminal lines. The initial terminal login process *getty*(8) accesses the *gettytab* file each time it starts, allowing simpler reconfiguration of terminal characteristics. Each entry in the data base is used to describe one class of terminals.

There is a default terminal class, *default*, that is used to set global defaults for all other classes. (That is, the *default* entry is read, then the entry for the class required is used to override particular settings.)

CAPABILITIES

Refer to *termcap*(5) for a description of the file layout. The *default* column below lists defaults obtained if there is no entry in the table obtained, nor one in the special *default* table.

Name	Type	Default	Description
ap	bool	false	terminal uses any parity
bd	num	0	backspace delay
bk	str	0377	alternate end of line character (input break)
cb	bool	false	use crt backspace mode
cd	num	0	carriage-return delay
ce	bool	false	use crt erase algorithm
ck	bool	false	use crt kill algorithm
cl	str	NULL	screen clear sequence
co	bool	false	console - add \n after login prompt
ds	str	^Y	delayed suspend character
ec	bool	false	leave echo OFF
ep	bool	false	terminal uses even parity
er	str	^?	erase character
et	str	^D	end of text (EOF) character
ev	str	NULL	initial enviroment
f0	num	unused	tty mode flags to write messages
f1	num	unused	tty mode flags to read login name
f2	num	unused	tty mode flags to leave terminal as
fd	num	0	form-feed (vertical motion) delay
fl	str	^O	output flush character
hc	bool	false	do NOT hangup line on last close
he	str	NULL	hostname editing string
hn	str	hostname	hostname
ht	bool	false	terminal has real tabs
ig	bool	false	ignore garbage characters in login name
im	str	NULL	initial (banner) message
in	str	^C	interrupt character
is	num	unused	input speed
kl	str	^U	kill character
lc	bool	false	terminal has lower case
lm	str	login:	login prompt
ln	str	^V	"literal next" character
lo	str	/bin/login	program to exec when name obtained
nd	num	0	newline (line-feed) delay

nl	bool	false	terminal has (or might have) a newline character
nx	str	default	next table (for auto speed selection)
op	bool	false	terminal uses odd parity
os	num	unused	output speed
pc	str	\0	pad character
pe	bool	false	use printer (hard copy) erase algorithm
ps	bool	false	line connected to a MICOM port selector
qu	str	^\	quit character
rp	str	^R	line retype character
rw	bool	false	do NOT use raw for input, use cbreak
sp	num	unused	line speed (input and output)
su	str	^Z	suspend character
tc	str	none	table continuation
to	num	0	timeout (seconds)
tt	str	NULL	terminal type (for enviroment)
ub	bool	false	do unbuffered output (of prompts etc)
uc	bool	false	terminal is known upper case only
we	str	^W	word erase character
xc	bool	false	do NOT echo control chars as ^X
xf	str	^S	XOFF (stop output) character
xn	str	^Q	XON (start output) character

If no line speed is specified, speed will not be altered from that which prevails when getty is entered. Specifying an input or output speed will override line speed for stated direction only.

Terminal modes to be used for the output of the message, for input of the login name, and to leave the terminal set as upon completion, are derived from the boolean flags specified. If the derivation should prove inadequate, any (or all) of these three may be overriden with one of the **f0**, **f1**, or **f2** numeric specifications, which can be used to specify (usually in octal, with a leading '0') the exact values of the flags. Local (new tty) flags are set in the top 16 bits of this (32 bit) value.

Should *getty* receive a null character (presumed to indicate a line break) it will restart using the table indicated by the **nx** entry. If there is none, it will re-use its original table.

Delays are specified in milliseconds, the nearest possible delay available in the tty driver will be used. Should greater certainty be desired, delays with values 0, 1, 2, and 3 are interpreted as choosing that particular delay algorithm from the driver.

The **cl** screen clear string may be preceded by a (decimal) number of milliseconds of delay required (a la termcap). This delay is simulated by repeated use of the pad character **pc**.

The initial message. and login message, **im** and **lm** may include the character sequence **%h** to obtain the hostname. (**%%** obtains a single '%' character.) The hostname is normally obtained from the system, but may be set by the **hn** table entry. In either case it may be edited with **he**. The **he** string is a sequence of characters, each character that is neither '@' nor '#' is copied into the final hostname. A '@' in the **he** string, causes one character from the real hostname to be copied to the final hostname. A '#' in the **he** string, causes the next character of the real hostname to be skipped. Surplus '@' and '#' characters are ignored.

When getty execs the login process, given in the **lo** string (usually "/bin/login"), it will have set the enviroment to include the terminal type, as indicated by the **tt** string (if it exists). The **ev** string, can be used to enter additional data into the environment. It is a list of comma separated strings, each of which will presumably be of the form *name=value*.

If a non-zero timeout is specified, with **to**, then getty will exit within the indicated number of seconds, either having received a login name and passed control to *login*, or having received an alarm signal, and exited. This may be useful to hangup dial in lines.

Output from *getty* is even parity unless **op** is specified. **Op** may be specified with **ap** to allow any parity on input, but generate odd parity output. Note: this only applies while getty is being run, terminal driver limitations prevent a more complete implementation. *Getty* does not check parity of input characters in *RAW* mode.

SEE ALSO

termcap(5), getty(8).

BUGS

Some ignorant peasants insist on changing the default special characters, so it is wise to always specify (at least) the erase, kill, and interrupt characters in the **default** table. In **all** cases, '#' or '^H' typed in a login name will be treated as an erase character, and '@' will be treated as a kill character.

The delay stuff is a real crock. Apart form its general lack of flexibility, some of the delay algorithms are not implemented. The terminal driver should support sane delay settings.

Currently *login*(1) stomps on the environment, so there is no point setting it in *gettytab*.

The **he** capability is stupid.

Termcap format is horrid, something more rational should have been chosen.

NAME

group − group file

DESCRIPTION

Group contains for each group the following information:

group name
encrypted password
numerical group ID
a comma separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; Each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

FILES

/etc/group

SEE ALSO

setgroups(2), initgroups(3X), crypt(3), passwd(1), passwd(5)

BUGS

The *passwd*(1) command won't change the passwords.

NAME

hosts — host name data base

DESCRIPTION

The *hosts* file contains information regarding the known hosts on the DARPA Internet. For each host a single line should be present with the following information:

official host name
Internet address
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts.

Network addresses are specified in the conventional "." notation using the *inet_addr*() routine from the Internet address manipulation library, *inet*(3N). Host names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/hosts

SEE ALSO

gethostent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

lda absolute load format

DESCRIPTION

The following is a description of the ".lda" object format: Typical record:

Byte 1	<1>	always 1
Byte 2	<0>	ignored by the loader
Byte 3	< >	low half of Nbytes
Byte 4	< >	high half of Nbytes
Byte 5	< >	low half of load point
Byte 6	< >	high half of load point
Byte 7	< >	first data byte
Byte Nbyte	< >	last data byte of record
Byte Nbyte+1	< >	Checksum

Notes: Nbytes -- The total number of bytes in the record NOT including the checksum byte. Checksum -- When added to the mod256 sum of all of the bytes in the record will result in a zero result. Execution Address Record:

Byte 1	<1>	always 1
Byte 2	<0>	ignored by the loader
Byte 3	< >	always 6 (Nbytes low)
Byte 4	< >	always 0 (Nbytes high)
Byte 5	< >	execution address low
Byte 6	< >	execution address high
Byte 7	< >	Checksum byte

SEE ALSO

ldachk(1), ldasav(1)

NAME

mtab − mounted file system table

SYNOPSIS

#include <fstab.h>
#include <mtab.h>

DESCRIPTION

Mtab resides in directory */etc* and contains a table of devices mounted by the *mount* command. *Umount* removes entries.

The table is a series of *mtab* structures, as defined in <mtab.h>. Each entry contains the null-padded name of the place where the special file is mounted, the null-padded name of the special file, and a type field, one of those defined in *<fstab.h>*. The special file has all its directories stripped away; that is, everything through the last '/' is thrown away. The type field indicates if the file system is mounted read-only, read-write, or read-write with disk quotas enabled.

This table is present only so people can look at it. It does not matter to *mount* if there are duplicated entries nor to *umount* if a name cannot be found.

FILES

/etc/mtab

SEE ALSO

mount(8)

NAME

networks – network name data base

DESCRIPTION

The *networks* file contains information regarding the known networks which comprise the DARPA Internet. For each network a single line should be present with the following information:

official network name
network number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network number may be specified in the conventional "." notation using the *inet_network*() routine from the Internet address manipulation library, *inet*(3N). Network names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/networks

SEE ALSO

getnetent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

newsrc – information file for readnews(1) and checknews(1)

DESCRIPTION

The *.newsrc* file contains the list of previously read articles and an optional options line for *readnews(1)* and *checknews(1)*. Each newsgroup that articles have been read from has a line of the form:

newsgroup: range

The *range* is a list of the articles read. It is basically a list of no.'s separated by commas with sequential no.'s collapsed with hyphens. For instance:

general: 1-78,80,85-90
fa.info-cpm: 1-7
net.news: 1
fa.info-vax! 1-5

If the : is replaced with an ! (as in info-vax above) the newsgroup is not subscribed to and will not be shown to the user.

An options line starts with the word **options** (left-justified). Then there are the list of options just as they would be on the command line. For instance:

options –n all !fa.sf-lovers !fa.human-nets –r
options –c –r

A string of lines beginning with a space or tab after the initial options line will be considered continuation lines.

FILES

~/.newsrc	Options and list of previously read articles

SEE ALSO

readnews(1), checknews(1)

NAME

passwd − password file

DESCRIPTION

Passwd contains for each user the following information:

name (login name, contains no upper case)
encrypted password
numerical user ID
numerical group ID
user's real name, office, extension, home phone.
initial working directory
program to use as Shell

The name may contain '&', meaning insert the login name. This information is set by the *chfn*(1) command and used by the *finger*(1) command.

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, then */bin/sh* is used.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user ID's to names.

Appropriate precautions must be taken to lock the file against changes if it is to be edited with a text editor; *vipw*(8) does the necessary locking.

FILES

/etc/passwd

SEE ALSO

getpwent(3), login(1), crypt(3), passwd(1), group(5), chfn(1), finger(1), vipw(8), adduser(8)

BUGS

A binary indexed file format should be available for fast access.

User information (name, office, etc.) should be stored elsewhere.

NAME

phones — remote host phone number data base

DESCRIPTION

The file /etc/phones contains the system-wide private phone numbers for the *tip*(1C) program. This file is normally unreadable, and so may contain privileged information. The format of the file is a series of lines of the form: <system-name>[\t]*<phone-number>. The system name is one of those defined in the *remote*(5) file and the phone number is constructed from [0123456789-=*%]. The "=" and "*" characters are indicators to the auto call units to pause and wait for a second dial tone (when going through an exchange). The "=" is required by the DF02-AC and the "*" is required by the BIZCOMP 1030.

Only one phone number per line is permitted. However, if more than one line in the file contains the same system name *tip*(1C) will attempt to dial each one in turn, until it establishes a connection.

FILES

/etc/phones

SEE ALSO

tip(1C), remote(5)

NAME

plot − graphics interface

DESCRIPTION

Files of this format are produced by routines described in *plot*(3X), and are interpreted for various devices by commands described in *plot*(1G). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the x and y values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the 'current point' for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in *plot*(3X).

m move: The next four bytes give a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes. See *plot*(1G).

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a newline.

a arc: The first four bytes give the center, the next four give the starting point, and the last four give the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

c circle: The first four bytes give the center of the circle, the next two the radius.

e erase: Start another frame of output.

f linemod: Take the following string, up to a newline, as the style for drawing further lines. The styles are 'dotted,' 'solid,' 'longdashed,' 'shortdashed,' and 'dotdashed.' Effective only in *plot 4014* and *plot ver.*

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *plot*(1G). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face isn't square.

4014	space(0, 0, 3120, 3120);
ver	space(0, 0, 2048, 2048);
300, 300s	space(0, 0, 4096, 4096);
450	space(0, 0, 4096, 4096);

SEE ALSO

plot(1G), plot(3X), graph(1G)

NAME
printcap — printer capability data base

SYNOPSIS
/etc/printcap

DESCRIPTION
Printcap is a simplified version of the *termcap*(5) data base used to describe line printers. The spooling system accesses the *printcap* file every time it is used, allowing dynamic addition and deletion of printers. Each entry in the data base is used to describe one printer. This data base may not be substituted for, as is possible for *termcap*, because it may allow accounting to be bypassed.

The default printer is normally *lp*, though the environment variable PRINTER may be used to override this. Each spooling utility supports an option, **−P***printer*, to allow explicit naming of a destination printer.

Refer to the *4.2BSD Line Printer Spooler Manual* for a complete discussion on how setup the database for a given printer.

CAPABILITIES
Refer to *termcap* for a description of the file layout.

Name	**Type**	**Default**	**Description**
af	str	NULL	name of accounting file
br	num	none	if lp is a tty, set the baud rate (ioctl call)
cf	str	NULL	cifplot data filter
df	str	NULL	tex data filter (DVI format)
fc	num	0	if lp is a tty, clear flag bits (sgtty.h)
ff	str	"\f"	string to send for a form feed
fo	bool	false	print a form feed when device is opened
fs	num	0	like 'fc' but set bits
gf	str	NULL	graph data filter (plot (3X) format)
ic	bool	false	driver supports (non standard) ioctl to indent printout
if	str	NULL	name of text filter which does accounting
lf	str	"/dev/console"	error logging file name
lo	str	"lock"	name of lock file
lp	str	"/dev/lp"	device name to open for output
mx	num	1000	maximum file size (in BUFSIZ blocks), zero = unlimited
nd	str	NULL	next directory for list of queues (unimplemented)
nf	str	NULL	ditroff data filter (device independent troff)
of	str	NULL	name of output filtering program
pl	num	66	page length (in lines)
pw	num	132	page width (in characters)
px	num	0	page width in pixels (horizontal)
py	num	0	page length in pixels (vertical)
rf	str	NULL	filter for printing FORTRAN style text files
rm	str	NULL	machine name for remote printer
rp	str	"lp"	remote printer name argument
rs	bool	false	restrict remote users to those with local accounts
rw	bool	false	open the printer device for reading and writing
sb	bool	false	short banner (one line only)
sc	bool	false	suppress multiple copies
sd	str	"/usr/spool/lpd"	spool directory
sf	bool	false	suppress form feeds
sh	bool	false	suppress printing of burst page header

st	str	"status"	status file name
tf	str	NULL	troff data filter (cat phototypesetter)
tr	str	NULL	trailer string to print when queue empties
vf	str	NULL	raster image filter
xc	num	0	if lp is a tty, clear local mode bits (tty (4))
xs	num	0	like 'xc' but set bits

Error messages sent to the console have a carriage return and a line feed appended to them, rather than just a line feed.

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

SEE ALSO

termcap(5), lpc(8), lpd(8), pac(8), lpr(1), lpq(1), lprm(1)
4.2BSD Line Printer Spooler Manual

NAME

protocols − protocol name data base

DESCRIPTION

The *protocols* file contains information regarding the known protocols used in the DARPA Internet. For each protocol a single line should be present with the following information:

official protocol name
protocol number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Protocol names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/protocols

SEE ALSO

getprotoent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

rcsfile -- format of RCS file

DESCRIPTION

An RCS file is an ASCII file. Its contents is described by the grammar below. The text is free format, i.e., spaces, tabs and new lines have no significance except in strings. Strings are enclosed by '@'. If a string contains a '@', it must be doubled.

The meta syntax uses the following conventions: '|' (bar) separates alternatives; '{' and '}' enclose optinal phrases; '{' and '}*' enclose phrases that may be repeated zero or more times; '{' and '}+' enclose phrases that must appear at least once and may be repeated; '<' and '>' enclose nonterminals.

```
<rcstext>      ::=    <admin> {<delta>}* <desc> {<deltatext>}*

<admin>        ::=    head          {<num>};
                      access        {<id>}*;
                      symbols       {<id> : <num>}*;
                      locks         {<id> : <num>}*;
                      comment       {<string>};

<delta>        ::=    <num>
                      date          <num>;
                      author        <id>;
                      state         {<id>};
                      branches      {<num>}*;
                      next          {<num>};

<desc>         ::=    desc          <string>

<deltatext>    ::=    <num>
                      log           <string>
                      text          <string>


<num>          ::=    {<digit>{.}}+

<digit>        ::=    0 | 1 | ... | 9

<id>           ::=    <letter>{<idchar>}*

<letter>       ::=    A | B | ... | Z | a | b | ... | z

<idchar>       ::=    Any printing ASCII character except space,
                      tab, carriage return, new line, and <special>.

<special>      ::=    ; | : | , | @

<string>       ::=    @{any ASCII character, with '@' doubled}*@
```

Identifiers are case sensitive. Keywords are in lower case only. The sets of keywords and identifiers may overlap.

The <delta> nodes form a tree. All nodes whose numbers consist of a single pair (e.g., 2.3, 2.1, 1.3, etc.) are on the "trunk", and are linked through the "next" field in order of decreasing numbers. The "head" field in the <admin> node points to the head of that sequence (i.e., contains the highest pair).

All <delta> nodes whose numbers consist of 2n fields (n≥2) (e.g., 3.1.1.1, 2.1.2.2, etc.) are linked as follows. All nodes whose first (2n)-1 number fields are identical are linked through the "next" field in order of increasing numbers. For each such sequence, the <delta> node whose number is identical to the first 2(n-1) number fields of the deltas on that sequence is called the branchpoint. The "branches" field of a node contains a list of the numbers of the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers.

Example:

```
                             Head
                              |
                              |
                              v
                          ---------
    / \          / \      |       |      / \           / \
   /   \        /   \     |  2.1  |     /   \         /   \
  /     \      /     \    |       |    /     \       /     \
 /1.2.1.3\    /1.3.1.1\   |       |   /1.2.2.2\     /1.2.2.1.1.1\
 ---------    ---------   ---------   ---------     -------------
     ↑            ↑           |           ↑               ↑
     |            |           |           |               |
     |            |           v           |               |
    / \           |       ---------      / \              |
   /   \          |       \  1.3  /     /   \             |
  /     \         ---------\     /     /     \-------------
 /1.2.1.1\                  \   /     /1.2.2.1\
 ---------                   \ /      ---------
     ↑                        |           ↑
     |                        |           |
     |                        v           |
     |                    ---------       |
     |                    \  1.2  /       |
     ----------------------\     /---------
                            \   /
                             \ /
                              |
                              |
                              v
                          ---------
                          \  1.1  /
                           \     /
                            \   /
                             \ /
```

Fig. 1: A revision tree

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.0 ; Release Date: 82/11/18 .

SEE ALSO

ci (1), co (1), ident (1), rcs (1), rcsdiff (1), rcsintro (1), rcsmerge (1), rlog (1), sccstorcs (8).

NAME

remote — remote host description file

DESCRIPTION

The systems known by *tip*(1C) and their attributes are stored in an ASCII file which is structured somewhat like the *termcap*(5) file. Each line in the file provides a description for a single *system*. Fields are separated by a colon (":"). Lines ending in a \ character with an immediately following newline are continued on the next line.

The first entry is the name(s) of the host system. If there is more than one name for a system, the names are separated by vertical bars. After the name of the system comes the fields of the description. A field name followed by an '=' sign indicates a string value follows. A field name followed by a '#' sign indicates a following numeric value.

Entries named "tip*" and "cu*" are used as default entries by *tip*, and the *cu* interface to *tip*, as follows. When *tip* is invoked with only a phone number, it looks for an entry of the form "tip300", where 300 is the baud rate with which the connection is to be made. When the *cu* interface is used, entries of the form "cu300" are used.

CAPABILITIES

Capabilities are either strings (str), numbers (num), or boolean flags (bool). A string capability is specified by *capability=value*; e.g. "dv=/dev/harris". A numeric capability is specified by *capability#value*; e.g. "xa#99". A boolean capability is specified by simply listing the capability.

at	(str) Auto call unit type.
br	(num) The baud rate used in establishing a connection to the remote host. This is a decimal number. The default baud rate is 300 baud.
cm	(str) An initial connection message to be sent to the remote host. For example, if a host is reached through port selector, this might be set to the appropriate sequence required to switch to the host.
cu	(str) Call unit if making a phone call. Default is the same as the 'dv' field.
di	(str) Disconnect message sent to the host when a disconnect is requested by the user.
du	(bool) This host is on a dial-up line.
dv	(str) UNIX device(s) to open to establish a connection. If this file refers to a terminal line, *tip*(1C) attempts to perform an exclusive open on the device to insure only one user at a time has access to the port.
el	(str) Characters marking an end-of-line. The default is NULL. '~' escapes are only recognized by *tip* after one of the characters in 'el', or after a carriage-return.
fs	(str) Frame size for transfers. The default frame size is equal to BUFSIZ.
hd	(bool) The host uses half-duplex communication, local echo should be performed.
ie	(str) Input end-of-file marks. The default is NULL.
oe	(str) Output end-of-file string. The default is NULL. When *tip* is transferring a file, this string is sent at end-of-file.
pa	(str) The type of parity to use when sending data to the host. This may be one of "even", "odd", "none", "zero" (always set bit 8 to zero), "one" (always set bit 8 to 1). The default is even parity.
pn	(str) Telephone number(s) for this host. If the telephone number field contains an @ sign, *tip* searches the file */etc/phones* file for a list of telephone numbers; c.f. *phones*(5).
tc	(str) Indicates that the list of capabilities is continued in the named description. This is

used primarily to share common capability information.

Here is a short example showing the use of the capability continuation feature:

```
UNIX-1200:\
        :dv=/dev/cau0:el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:oe=^D:br#1200:
arpavax|ax:\
        :pn=7654321%:tc=UNIX-1200
```

FILES

/etc/remote

SEE ALSO

tip(1C), phones(5)

NAME

services – service name data base

DESCRIPTION

The *services* file contains information regarding the known services available in the DARPA Internet. For each service a single line should be present with the following information:

official service name
port number
protocol name
aliases

Items are separated by any number of blanks and/or tab characters. The port number and protocol name are considered a single *item*; a "/" is used to separate the port and protocol (e.g. "512/tcp"). A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Service names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/services

SEE ALSO

getservent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

stab – symbol table types

SYNOPSIS

#include <stab.h>

DESCRIPTION

Stab.h defines some values of the n_type field of the symbol table of a.out files. These are the types for permanent symbols (i.e. not local labels, etc.) used by the old debugger *sdb* and the Berkeley Pascal compiler *pc*(1). Symbol table entries can be produced by the *.stabs* assembler directive. This allows one to specify a double-quote delimited name, a symbol type, one char and one short of information about the symbol, and an unsigned long (usually an address). To avoid having to produce an explicit label for the address field, the *.stabd* directive can be used to implicitly address the current location. If no name is needed, symbol table entries can be generated using the *.stabn* directive. The loader promises to preserve the order of symbol table entries produced by *.stab* directives. As described in *a.out*(5), an element of the symbol table consists of the following structure:

```
/*
 * Format of a symbol table entry.
 */
struct nlist {
        union {
                char  *n_name;  /* for use when in-core */
                long  n_strx;   /* index into file string table */
        } n_un;
        unsigned char  n_type;   /* type flag */
        char           n_other;  /* unused */
        short          n_desc;   /* see struct desc, below */
        unsigned n_value;        /* address or offset or line */
};
```

The low bits of the n_type field are used to place a symbol into at most one segment, according to the following masks, defined in <*a.out.h*>. A symbol can be in none of these segments by having none of these segment bits set.

```
/*
 * Simple values for n_type.
 */
#define N_UNDF   0x0  /* undefined */
#define N_ABS    0x2  /* absolute */
#define N_TEXT   0x4  /* text */
#define N_DATA   0x6  /* data */
#define N_BSS    0x8  /* bss */

#define N_EXT    01   /* external bit, or'ed in */
```

The n_value field of a symbol is relocated by the linker, *ld*(1) as an address within the appropriate segment. N_value fields of symbols not in any segment are unchanged by the linker. In addition, the linker will discard certain symbols, according to rules of its own, unless the n_type field has one of the following bits set:

```
/*
 * Other permanent symbol table entries have some of the N_STAB bits set.
 * These are given in <stab.h>
 */
#define N_STAB          0xe0/* if any of these bits set, don't discard */
```

This allows up to 112 (7 * 16) symbol types, split between the various segments. Some of these have already been claimed. The old symbolic debugger, *sdb*, uses the following n_type values:

```
#define N_GSYM   0x20 /* global symbol: name,,0,type,0 */
#define N_FNAME  0x22 /* procedure name (f77 kludge): name,,0 */
#define N_FUN    0x24 /* procedure: name,,0,linenumber,address */
#define N_STSYM  0x26 /* static symbol: name,,0,type,address */
#define N_LCSYM  0x28 /* .lcomm symbol: name,,0,type,address */
#define N_RSYM   0x40 /* register sym: name,,0,type,register */
#define N_SLINE  0x44 /* src line: 0,,0,linenumber,address */
#define N_SSYM   0x60 /* structure elt: name,,0,type,struct_offset */
#define N_SO     0x64 /* source file name: name,,0,0,address */
#define N_LSYM   0x80 /* local sym: name,,0,type,offset */
#define N_SOL    0x84 /* #included file name: name,,0,0,address */
#define N_PSYM   0xa0 /* parameter: name,,0,type,offset */
#define N_ENTRY  0xa4 /* alternate entry: name,linenumber,address */
#define N_LBRAC  0xc0 /* left bracket: 0,,0,nesting level,address */
#define N_RBRAC  0xe0 /* right bracket: 0,,0,nesting level,address */
#define N_BCOMM  0xe2 /* begin common: name,, */
#define N_ECOMM  0xe4 /* end common: name,, */
#define N_ECOML  0xe8 /* end common (local name): ,,address */
#define N_LENG   0xfe /* second stab entry with length information */
```

where the comments give *sdb* conventional use for *.stabs* and the n_name, n_other, n_desc, and n_value fields of the given n_type. *Sdb* uses the n_desc field to hold a type specifier in the form used by the Portable C Compiler, *cc*(1), in which a base type is qualified in the following structure:

```
struct desc {
        short  q6:2,
               q5:2,
               q4:2,
               q3:2,
               q2:2,
               q1:2,
               basic:4;
};
```

There are four qualifications, with q1 the most significant and q6 the least significant:

0	none
1	pointer
2	function
3	array

The sixteen basic types are assigned as follows:

0	undefined
1	function argument
2	character
3	short
4	int
5	long
6	float
7	double
8	structure
9	union

10	enumeration
11	member of enumeration
12	unsigned character
13	unsigned short
14	unsigned int
15	unsigned long

The Berkeley Pascal compiler, *pc*(1), uses the following n_type value:

```
#define N_PC    0x30    /* global pascal symbol: name,,0,subtype,line */
```

and uses the following subtypes to do type checking across separately compiled files:

1	source file name
2	included file name
3	global label
4	global constant
5	global type
6	global variable
7	global function
8	global procedure
9	external function
10	external procedure
11	library variable
12	library routine

SEE ALSO

as(1), ld(1), dbx(1), a.out(5)

BUGS

Sdb assumes that a symbol of type N_GSYM with name *name* is located at address _*name*.

More basic types are needed.

NAME

tar — tape archive file format

DESCRIPTION

Tar, (the tape archive command) dumps several files into one, in a medium suitable for transportation.

A "tar tape" or file is a series of blocks. Each block is of size TBLOCK. A file on the tape is represented by a header block which describes the file, followed by zero or more blocks which give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an end-of-file indicator.

The blocks are grouped for physical I/O operations. Each group of *n* blocks (where *n* is set by the **b** keyletter on the *tar*(1) command line — default is 20 blocks) is written with a single system call; on nine-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads.

The header block looks like:

```
#define TBLOCK      512
#define NAMSIZ      100

union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMSIZ];
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char linkflag;
                char linkname[NAMSIZ];
        } dbuf;
};
```

Name is a null-terminated string. The other fields are zero-filled octal numbers in ASCII. Each field (of width w) contains w-2 digits, a space, and a null, except *size* and *mtime*, which do not contain the trailing null. *Name* is the name of the file, as specified on the *tar* command line. Files dumped because they were in a directory which was named in the command line have the directory name as prefix and */filename* as suffix. *Mode* is the file mode, with the top bit masked off. *Uid* and *gid* are the user and group numbers which own the file. *Size* is the size of the file in bytes. Links and symbolic links are dumped with this field specified as zero. *Mtime* is the modification time of the file at the time it was dumped. *Chksum* is a decimal ASCII value which represents the sum of all the bytes in the header block. When calculating the checksum, the *chksum* field is treated as if it were all blanks. *Linkflag* is ASCII '0' if the file is "normal" or a special file, ASCII '1' if it is an hard link, and ASCII '2' if it is a symbolic link. The name linked-to, if any, is in *linkname,* with a trailing null. Unused fields of the header are binary zeros (and are included in the checksum).

The first time a given i-node number is dumped, it is dumped as a regular file. The second and subsequent times, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved, but not the file it was linked to, an error message is printed and the tape must be manually rescanned to retrieve the linked-to file.

The encoding of the header is designed to be portable across machines.

SEE ALSO

tar(1)

BUGS

Names or linknames longer than NAMSIZ produce error reports and cannot be dumped.

NAME

termcap — terminal capability data base

SYNOPSIS

/etc/termcap

DESCRIPTION

Termcap is a data base describing terminals, used, *e.g.*, by *vi*(1) and *curses*(3X). Terminals are described in *termcap* by giving a set of capabilities which they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in *termcap.*

Entries in *termcap* consist of a number of ':' separated fields. The first entry for each terminal gives the names which are known for the terminal, separated by '|' characters. The first name is always 2 characters long and is used by older version 6 systems which store the terminal type in a 16 bit word in a systemwide data base. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks; the last name may well contain blanks for readability.

CAPABILITIES

(P) indicates padding may be specified
(P*) indicates that padding may be based on no. lines affected

Name	**Type**	**Pad?**	**Description**
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not ^H
bs	bool		Terminal can backspace with ^H
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
ch	str	(P)	Like cm but horizontal motion only, line stays same
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
cr	str	(P*)	Carriage return, (default ^M)
cs	str	(P)	Change scrolling region (vt100), like cm
cv	str	(P)	Like ch but vertical only.
da	bool		Display may be retained above
dB	num		Number of millisec of bs delay needed
db	bool		Display may be retained below
dC	num		Number of millisec of cr delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of nl delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode

ei	str		End insert mode; give ":ei=:" if **ic**
eo	str		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default ^L)
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
ho	str		Home cursor (if no **cm**)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	bool		Insert mode (enter); give ":im=:" if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by "other" function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of "keypad transmit" mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of "other" keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in "keypad transmit" mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on "other" function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map, used by vi version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor.
ms	bool		Safe to move while in standout and underline mode
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default **\n**)
ns	bool		Terminal is a CRT but doesn't scroll.
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
pt	bool		Has hardware tabs (may need to be set with **is**)
se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by so or se
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use **cm**
ti	str		String to begin programs that use **cm**
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by us or ue

ul	bool	Terminal underlines even though it doesn't overstrike
up	str	Upline (cursor up)
us	str	Start underscore mode
vb	str	Visible bell (may not move cursor)
ve	str	Sequence to end open/visual mode
vs	str	Sequence to start open/visual mode
xb	bool	Beehive (f1=escape, f2=ctrl C)
xn	bool	A newline is ignored after a wrap (Concept)
xr	bool	Return acts like **ce** \r \n (Delta Data)
xs	bool	Standout not erased by writing over it (HP 264?)
xt	bool	Tabs are destructive, magic so char (Teleray 1061)

A Sample Entry

The following entry, which describes the Concept−100, is among the more complex entries in the *termcap* file as of this writing. (This particular concept entry is outdated, and is used as an example only.)

```
c1|c100|concept100:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200:\
        :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:cm=\Ea%+ %+ :co#80:\
        :dc=16\E^A:dl=3*\E^B:ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
        :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and that empty fields may be included for readability (here between the last field on a line and the first field on the next). Capabilities in *termcap* are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two letter codes. For instance, the fact that the Concept has "automatic margins" (i.e. an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **co** which indicates the number of columns the terminal has gives the value '80' for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either a integer, e.g. '20', or an integer followed by an '*', i.e. '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n** **\r** **\t** **\b** **\f** give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. If it is necessary to place a : in a capability it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability it must be encoded as **\200**. The routines which deal with *termcap* use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *termcap* and to build up a description gradually, using partial descriptions with *ex* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *termcap* file to describe it or bugs in *ex*. To easily test a new terminal description you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in */etc/termcap*. TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor. (This only works on version 7 systems.)

Basic capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, then this is given by the **cl** string capability. If the terminal can backspace, then it should have the **bs** capability, unless a backspace is accomplished by a character other than ^H (ugh) in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability.

A very important point here is that the local cursor motions encoded in *termcap* are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the *termcap* file usually assumes that this is on, i.e. **am**.

These capabilities suffice to describe hardcopy and "glass-tty" terminals. Thus the model 33 teletype is described as

```
t3|33|tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as

```
cl|adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor addressing

Cursor addressing in the terminal is described by a **cm** string capability, with *printf*(3S) like escapes **%x** in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, then its arguments are the line and then the column to which motion is desired, and the % encodings have the following meanings:

%d	as in *printf*, 0 origin
%2	like %2d
%3	like %3d
%.	like %c
%+x	adds *x* to value, then %.
%>xy	if value > x adds y, no output.
%r	reverses order of line and column, no output
%i	increments line/column (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output.
%D	Reverse coding (x-2*(x%16)), no output. (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is "cm=6\E&%r%2c%2Y". The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, "cm=^T%.%.". Terminals which use "%." need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t**, **\n** **^D** and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus "cm=\E=%+ %+ ".

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, this should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen) then this can be given as **ho**; similarly a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

Area clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **ce**. If the terminal can clear from the current position to the end of the display, then this should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **al**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl**; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as **sb**, but just **al** suffices. If the terminal can retain display memory above then the **da** capability should be given; if display memory can be retained below then **db** should be given. These let the editor understand that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using *termcap.* The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type "abc   def" using local cursor motions (not spaces) between the "abc" and the "def". Then position the cursor before the "abc" and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the "abc" shifts over to the "def" which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for "insert null". If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines. We have seen no terminals which have an insert mode not not falling into one of these two classes.

The editor can handle both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give as **im** the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Give as **ei** the sequence to leave insert mode (give this, with an empty value also if you gave **im** so). Now give as **ic** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ic**, terminals which send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g. if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode these can be given as **so** and **se** respectively. If there are several flavors of standout mode (such as inverse video, blinking, or underlining — half bright is not usually an acceptable "standout" mode unless the terminal is in inverse video mode constantly) the preferred mode is inverse video by itself. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **ug** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **us** and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

Many terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of *ex*, this can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change, e.g., from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl, kr, ku, kd,** and **kh** respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0, l1, ..., l9**. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the *termcap* 2 letter codes can be given in the **ko** capability, for example, ":ko=cl,ll,sf,sb:", which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete but still in use in version 2 of vi, which must be run on some minicomputers due to memory limitations. This field is redundant with **kl, kr, ku, kd,** and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, the mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than ^I to tab, then this can be given as **ta**.

Hazeltine terminals, which don't allow '~' characters to be printed should indicate **hz**. Datamedia terminals, which echo carriage-return linefeed for carriage return and then ignore a following linefeed should indicate **nc**. Early Concept terminals, which ignore a linefeed immediately after an **am** wrap, should indicate **xn**. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt**. Other specific terminal problems may be corrected by adding more capabilities of the form x*x*.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** will be printed before **if**. This is useful where **if** is */usr/lib/tabset/std* but **is** clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be *last* and the combined length of the two entries must not exceed 1024. Since *termlib* routines search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be canceled with **xx@** where xx is the capability. For example, the entry

```
hn|2621nl:ks@:ke@:tc=2621:
```

defines a 2621nl that does not have the **ks** or **ke** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

/etc/termcap file containing terminal descriptions

SEE ALSO

ex(1), curses(3X), termcap(3X), tset(1), vi(1), ul(1), more(1)

AUTHOR

William Joy
Mark Horton added underlining and keypad support

BUGS

Ex allows only 256 characters for string capabilities, and the routines in *termcap*(3X) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The **ma**, vs, and ve entries are specific to the *vi* program.

Not all programs support all entries. There are entries that are not supported by any program.

NAME

tp − DEC/mag tape formats

DESCRIPTION

Tp dumps files to and extracts files from DECtape and magtape. The formats of these tapes are the same except that magtapes have larger directories.

Block zero contains a copy of a stand-alone bootstrap program. See *reboot*(8).

Blocks 1 through 24 for DECtape (1 through 62 for magtape) contain a directory of the tape. There are 192 (resp. 496) entries in the directory; 8 entries per block; 64 bytes per entry. Each entry has the following format:

```
struct {
        char            pathname[32];
        unsigned short  mode;
        char            uid;
        char            gid;
        char            unused1;
        char            size[3];
        long            modtime;
        unsigned short  tapeaddr;
        char            unused2[16];
        unsigned short  checksum;
};
```

The path name entry is the path name of the file when put on the tape. If the pathname starts with a zero word, the entry is empty. It is at most 32 bytes long and ends in a null byte. Mode, uid, gid, size and time modified are the same as described under i-nodes (see file system *fs*(5)). The tape address is the tape block number of the start of the contents of the file. Every file starts on a block boundary. The file occupies (size+511)/512 blocks of continuous tape. The checksum entry has a value such that the sum of the 32 words of the directory entry is zero.

Blocks above 25 (resp. 63) are available for file storage.

A fake entry has a size of zero.

SEE ALSO

fs(5), tp(

BUGS

The *pathname, uid, gid,* and *size* fields are too small.

NAME

ttys − terminal initialization data

DESCRIPTION

The *ttys* file is read by the *init* program and specifies which terminal special files are to have a process created for them so that people can log in. There is one line in the *ttys* file per special file.

The first character of a line in the *ttys* file is either '0' or '1'. If the first character on the line is a '0', the *init* program ignores that line. If the first character on the line is a '1', the *init* program creates a login process for that line. The second character on each line is used as an argument to *getty*(8), which performs such tasks as baud-rate recognition, reading the login name, and calling *login*. For normal lines, the character is '0'; other characters can be used, for example, with hard-wired terminals where speed recognition is unnecessary or which have special characteristics. (*Getty* will have to be fixed in such cases.) The remainder of the line is the terminal's entry in the device directory, /dev.

FILES

/etc/ttys

SEE ALSO

gettytab(5), init(8), getty(8), login(1)

NAME

ttytype − data base of terminal types by port

SYNOPSIS

/etc/ttytype

DESCRIPTION

Ttytype is a database containing, for each tty port on the system, the kind of terminal that is attached to it. There is one line per port, containing the terminal kind (as a name listed in termcap (5)), a space, and the name of the tty, minus /dev/.

This information is read by *tset*(1) and by *login*(1) to initialize the TERM variable at login time.

SEE ALSO

tset(1), login(1)

BUGS

Some lines are merely known as "dialup" or "plugboard".

NAME

types − primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

The data types defined in the include file are used in UNIX system code; some data of these types are accessible to user code:

```
/*      types.h    6.1     83/07/29*/

/*
 * Basic system types and major/minor device constructing/busting macros.
 */

/* major part of a device */
#define major(x)   ((int)(((unsigned)(x)>>8)&0377))

/* minor part of a device */
#define minor(x)   ((int)((x)&0377))

/* make a device number */
#define makedev(x,y)      ((dev_t)(((x)<<8) | (y)))

typedef unsigned char     u_char;
typedef unsigned short    u_short;
typedef unsigned int      u_int;
typedef unsigned long     u_long;
typedef unsigned short    ushort;/* sys III compat */

#ifdef vax
typedef struct      _physadr { int r[1]; } *physadr;
typedef struct      label_t {
        int         val[14];
} label_t;
#endif
typedef struct      _quad { long val[2]; } quad;
typedef long        daddr_t;
typedef char *      caddr_t;
typedef u_long      ino_t;
typedef long        swblk_t;
typedef int         size_t;
typedef int         time_t;
typedef short       dev_t;
typedef int         off_t;

typedef struct      fd_set { int fds_bits[1]; } fd_set;
```

The form *daddr_t* is used for disk addresses except in an i-node on disk, see *fs*(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

SEE ALSO

fs(5), time(3), lseek(2), adb(1)

NAME

utmp, wtmp -- login records

SYNOPSIS

#include <utmp.h>

DESCRIPTION

The *utmp* file records information about who is currently using the system. The file is a sequence of entries with the following structure declared in the include file:

```
/*	utmp.h	4.2	83/05/22	*/

/*
 * Structure of utmp and wtmp files.
 *
 * Assuming the number 8 is unwise.
 */
struct utmp {
	char	ut_line[8];		/* tty name */
	char	ut_name[8];		/* user id */
	char	ut_host[16];		/* host name, if remote */
	long	ut_time;	/* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of *time*(3C).

The *wtmp* file records all logins and logouts. A null user name indicates a logout on the associated terminal. Furthermore, the terminal namindicathat system was rebooted at the indicated time; the adjacent pair of entries with terminal names '|' and '{' indicate the system-maintained time just before and just after a *date* command has changed the system's idea of the time.

Wtmp is maintained by *login*(1) and *init*(8). Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by *ac*(8).

FILES

/etc/utmp
/usr/adm/wtmp

SEE ALSO

login(1), init(8), who(1), ac(8)

NAME

uuencode − format of an encoded uuencode file

DESCRIPTION

Files output by *uuencode(1C)* consist of a header line, followed by a number of body lines, and a trailer line. *Uudecode(1C)* will ignore any lines preceding the header or following the trailer. Lines preceding a header must not, of course, look like a header.

The header line is distinguished by having the first 6 characters "begin ". The word *begin* is followed by a mode (in octal), and a string which names the remote file. A space separates the three items in the header line.

The body consists of a number of lines, each at most 62 characters long (including the trailing newline). These consist of a character count, followed by encoded characters, followed by a newline. The character count is a single printing character, and represents an integer, the number of bytes the rest of the line represents. Such integers are always in the range from 0 to 63 and can be determined by subtracting the character space (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a space to make the characters printing. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra garbage will be included to make the character count a multiple of 4. The body is terminated by a line with a count of zero. This line consists of one ASCII space.

The trailer line consists of "end" on a line by itself.

SEE ALSO

uuencode(1C), uudecode(1C), uusend(1C), uucp(1C), mail(1)

NAME

vfont − font formats for the Benson-Varian or Versatec

SYNOPSIS

/usr/lib/vfont/*

DESCRIPTION

The fonts for the printer/plotters have the following format. Each file contains a header, an array of 256 character description structures, and then the bit maps for the characters themselves. The header has the following format:

```
struct header {
        short           magic;
        unsigned short  size;
        short           maxx;
        short           maxy;
        short           xtnd;
} header;
```

The *magic* number is 0436 (octal). The *maxx, maxy,* and *xtnd* fields are not used at the current time. *Maxx* and *maxy* are intended to be the maximum horizontal and vertical size of any glyph in the font, in raster lines. The *size* is the size of the bit maps for the characters in bytes. Before the maps for the characters is an array of 256 structures for each of the possible characters in the font. Each element of the array has the form:

```
struct dispatch {
        unsigned short  addr;
        short           nbytes;
        char            up;
        char            down;
        char            left;
        char            right;
        short           width;
};
```

The *nbytes* field is nonzero for characters which actually exist. For such characters, the *addr* field is an offset into the rest of the file where the data for that character begins. There are *up+down* rows of data for each character, each of which has *left+right* bits, rounded up to a number of bytes. The *width* field is not used by vcat, although it is to make width tables for *troff.* It represents the logical width of the glyph, in raster lines, and shows where the base point of the next glyph would be.

FILES

/usr/lib/vfont/*

SEE ALSO

troff(1), pti(1), vpr(1), vtroff(1), vfontinfo(1)

NAME

vgrindefs — vgrind's language definition data base

SYNOPSIS

/usr/lib/vgrindefs

DESCRIPTION

Vgrindefs contains all language definitions for vgrind. The data base is very similar to *termcap*(5).

FIELDS

The following table names and describes each field.

Name	**Type**	**Description**
pb	str	regular expression for start of a procedure
bb	str	regular expression for start of a lexical block
be	str	regular expression for the end of a lexical block
cb	str	regular expression for the start of a comment
ce	str	regular expression for the end of a comment
sb	str	regular expression for the start of a string
se	str	regular expression for the end of a string
lb	str	regular expression for the start of a character constant
le	str	regular expression for the end of a character constant
tl	bool	present means procedures are only defined at the top lexical level
oc	bool	present means upper and lower case are equivalent
kw	str	a list of keywords separated by spaces

Example

The following entry, which describes the C language, is typical of a language entry.

```
C|c:    :pb=^\d?*?\d?\p\d??):bb={:be=}:cb=/*:ce=*/:sb=":se=\e":\
        :lb=':le=\e':tl:\
        :kw=asm auto break case char continue default do double else enum\
        extern float for fortran goto if int long register return short\
        sizeof static struct switch typedef union unsigned while #define\
        #else #endif #if #ifdef #ifndef #include #undef # define else endif\
        if ifdef ifndef include undef:
```

Note that the first field is just the language name (and any variants of it). Thus the C language could be specified to *vgrind*(1) as "c" or "C".

Entries may continue onto multiple lines by giving a \ as the last character of a line. Capabilities in *vgrindefs* are of two types: Boolean capabilities which indicate that the language has some particular feature and string capabilities which give a regular expression or keyword list.

REGULAR EXPRESSIONS

Vgrindefs uses regular expression which are very similar to those of *ex*(1) and *lex*(1). The characters '^', '$', ':' and '\' are reserved characters and must be "quoted" with a preceding \ if they are to be included as normal characters. The metasymbols and their meanings are:

$	the end of a line
^	the beginning of a line
\d	a delimiter (space, tab, newline, start of line)
\a	matches any string of symbols (like .* in lex)
\p	matches any alphanumeric name. In a procedure definition (pb) the string that matches

this symbol is used as the procedure name.

()	grouping
|	alternation
?	last item is optional
\e	preceding any string means that the string will not match an input string if the input string is preceded by an escape character (\). This is typically used for languages (like C) which can include the string delimiter in a string b escaping it.

Unlike other regular expressions in the system, these match words and not characters. Hence something like "(tramp|steamer)flies?" would match "tramp", "steamer", "trampflies", or "steamerflies".

KEYWORD LIST

The keyword list is just a list of keywords in the language separated by spaces. If the "oc" boolean is specified, indicating that upper and lower case are equivalent, then all the keywords should be specified in lower case.

FILES

/usr/lib/vgrindefs file containing terminal descriptions

SEE ALSO

vgrind(1), troff(1)

AUTHOR

Dave Presotto

BUGS

NAME

aardvark − yet another exploration game

SYNOPSIS

/usr/games/aardvark

DESCRIPTION

Aardvark is yet another computer fantasy simulation game of the adventure/zork genre. This one is written in DDL (Dungeon Definition Language) and is intended primarily as an example of how to write a dungeon in DDL.

FILES

/usr/games/lib/ddlrun ddl interpreter
/usr/games/lib/aardvarkinternal form of aardvark dungeon

AUTHOR

Mike Urban, UCLA

NAME
adventure — an exploration game

SYNOPSIS
/usr/games/adventure

DESCRIPTION
The object of the game is to locate and explore Colossal Cave, find the treasures hidden there, and bring them back to the building with you. The program is self-describing to a point, but part of the game is to discover its rules.

To terminate a game, type 'quit'; to save a game for later resumption, type 'suspend'.

BUGS
Saving a game creates a large executable file instead of just the information needed to resume the game.

NAME

arithmetic − provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [**+−x/**] [range]

DESCRIPTION

Arithmetic types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; **+−x/** respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is **+−**

Range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

NAME

backgammon − the game

SYNOPSIS

/usr/games/backgammon

DESCRIPTION

This program does what you expect. It will ask whether you need instructions.

NAME

banner − print large banner on printer

SYNOPSIS

/usr/games/banner [**−w***n*] message ...

DESCRIPTION

Banner prints a large, high quality banner on the standard output. If the message is omitted, it prompts for and reads one line of its standard input. If **−w** is given, the output is scrunched down from a width of 132 to *n* , suitable for a narrow terminal. If *n* is omitted, it defaults to 80.

The output should be printed on a hard-copy device, up to 132 columns wide, with no breaks between the pages. The volume is enough that you want a printer or a fast hardcopy terminal, but if you are patient, a decwriter or other 300 baud terminal will do.

BUGS

Several ASCII characters are not defined, notably <, >, [,], \, ^, _, {, }, |, and ~. Also, the characters ", ', and & are funny looking (but in a useful way.)

The **−w** option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

AUTHOR

Mark Horton

NAME

bcd − convert to antique media

SYNOPSIS

/usr/games/bcd text

DESCRIPTION

Bcd converts the literal *text* into a form familiar to old-timers.

SEE ALSO

dd(1)

NAME

boggle − play the game of boggle

SYNOPSIS

/usr/games/boggle [+] [++]

DESCRIPTION

This program is intended for people wishing to sharpen their skills at Boggle (TM Parker Bros.). If you invoke the program with 4 arguments of 4 letters each, (*e.g.* "**boggle appl epie moth erhd**") the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If you invoke the program without arguments, it will generate a board for you, let you enter words for 3 minutes, and then tell you how well you did relative to **/usr/dict/words**.

The object of Boggle is to find, within 3 minutes, as many words as possible in a 4 by 4 grid of letters. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found.

In interactive play, enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting 'break'. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Advanced players may wish to invoke the program with 1 or 2 +'s as the first argument. The first + removes the restriction that positions can only be used once in each word. The second + causes a position to be considered adjacent to itself as well as its (up to) 8 neighbors.

NAME

canfield, cfscores — the solitaire card game canfield

SYNOPSIS

/usr/games/canfield
/usr/games/cfscores

DESCRIPTION

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In Canfield, tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be able to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types 'ht' for his move. Foundation base cards are also automatically moved to the foundation when they become available.

The command 'c' causes *canfield* to maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase ones chances of winning.

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as is possible without moving any cards from your hand to the talon. (the initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute.

With no arguments, the program *cfscores* prints out the current status of your canfield account. If a user name is specified, it prints out the status of their canfield account. If the **−a** flag is specified, it prints out the canfield accounts for all users that have played the game since the database was set up.

FILES

/usr/games/canfield	the game itself
/usr/games/cfscores	the database printer
/usr/games/lib/cfscores	the database of scores

BUGS

It is impossible to cheat.

AUTHORS

Originally written: Steve Levine
Further random hacking by: Steve Feldman, Kirk McKusick, Mikey Olson, and Eric Allman.

NAME

chase − Try to escape to killer robots

SYNOPSIS

/usr/games/chase [*nrobots*] [*nfences*]

DESCRIPTION

The object of the game chase is to move around inside of the box on the screen without getting eaten by the robots chasing and without running into anything.

If a robot runs into another robot while chasing you, they crash and leave a junk heap. If a robot runs into a fence, it is destroyed.

If you can survive until all the robots are destroyed, you have won!

If you do not specify either *nrobots* or *nfences,* chase will prompt you for them.

NAME

chess — the game of chess

SYNOPSIS

/usr/games/chess

DESCRIPTION

Chess is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol '+' is used to specify check; 'o-o' and 'o-o-o' specify castling. To play black, type 'first'; to print the board, type an empty line.

Each move is echoed in the appropriate notation followed by the program's reply.

FILES

/usr/lib/chess	binary image to run in compatibility mode

DIAGNOSTICS

The most cryptic diagnostic is 'eh?' which means that the input was syntactically incorrect.

BUGS

Pawns may be promoted only to queens.

NAME

ching − the book of changes and other cookies

SYNOPSIS

/usr/games/ching [hexagram]

DESCRIPTION

The *I Ching* or *Book of Changes* is an ancient Chinese oracle that has been in use for centuries as a source of wisdom and advice.

The text of the *oracle* (as it is sometimes known) consists of sixty-four *hexagrams*, each symbolized by a particular arrangement of six straight (−−−) and broken (− −) lines. These lines have values ranging from six through nine, with the even values indicating the broken lines.

Each hexagram consists of two major sections. The **Judgement** relates specifically to the matter at hand (E.g., "It furthers one to have somewhere to go.") while the **Image** describes the general attributes of the hexagram and how they apply to one's own life ("Thus the superior man makes himself strong and untiring.").

When any of the lines have the values six or nine, they are moving lines; for each there is an appended judgement which becomes significant. Furthermore, the moving lines are inherently unstable and change into their opposites; a second hexagram (and thus an additional judgement) is formed.

Normally, one consults the oracle by fixing the desired question firmly in mind and then casting a set of changes (lines) using yarrow−stalks or tossed coins. The resulting hexagram will be the answer to the question.

Using an algorithm suggested by S. C. Johnson, the UNIX *oracle* simply reads a question from the standard input (up to an EOF) and hashes the individual characters in combination with the time of day, process id and any other magic numbers which happen to be lying around the system. The resulting value is used as the seed of a random number generator which drives a simulated coin−toss divination. The answer is then piped through **nroff** for formatting and will appear on the standard output.

For those who wish to remain steadfast in the old traditions, the oracle will also accept the results of a personal divination using, for example, coins. To do this, cast the change and then type the resulting line values as an argument.

The impatient modern may prefer to settle for Chinese cookies; try *fortune*(6).

SEE ALSO

It furthers one to see the great man.

DIAGNOSTICS

The great prince issues commands,
Founds states, vests families with fiefs.
Inferior people should not be employed.

BUGS

Waiting in the mud
Brings about the arrival of the enemy.

If one is not extremely careful,
Somebody may come up from behind and strike him.
Misfortune.

NAME

cribbage − the card game cribbage

SYNOPSIS

/usr/games/cribbage [−req] *name ...*

DESCRIPTION

Cribbage plays the card game cribbage, with the program playing one hand and the user the other. The program will initially ask the user if the rules of the game are needed − if so, it will print out the appropriate section from *According to Hoyle* with *more (1).*

Cribbage options include:

−e When the player makes a mistakes scoring his hand or crib, provide an explanation of the correct score. (This is especially useful for beginning players.)

−q Print a shorter form of all messages − this is only recommended for users who have played the game without specifying this option.

−r Instead of asking the player to cut the deck, the program will randomly cut the deck.

Cribbage first asks the player whether he wishes to play a short game ("once around", to 61) or a long game ("twice around", to 121). A response of 's' will result in a short game, any other response will play a long game.

At the start of the first game, the program asks the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, the program first prints the player's hand, whose crib it is, and then asks the player to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, the program cuts the deck (if it is the player's crib) or asks the player to cut the deck (if it's its crib); in the later case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the person who doesn't have the crib) leading the first card. Play continues, as per cribbage, until all cards are exhausted. The program keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. The program requests the player to score his hand (and the crib, if it is his) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

Cards are specified as rank followed by suit. The ranks may be specified as one of: 'a', '2', '3', '4', '5', '6', '7', '8', '9', 't', 'j', 'q', and 'k', or alternatively, one of: "ace", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten", "jack", "queen", and "king". Suits may be specified as: 's', 'h', 'd', and 'c', or alternatively as: "spades", "hearts", "diamonds", and "clubs". A card may be specified as: <rank> " " <suit>, or: <rank> " of " <suit>. If the single letter rank and suit designations are used, the space separating the suit and rank may be left out. Also, if only one card of the desired rank is playable, typing the rank is sufficient. For example, if your hand was "2H, 4D, 5C, 6H, JC, KD" and it was desired to discard the king of diamonds, any of the following could be typed: "k", "king", "kd", "k d", "k of d", "king d", "king of d", "k diamonds", "k of diamonds", "king diamonds", or "king of diamonds".

FILES

/usr/games/cribbage

AUTHORS

Earl T. Cohen wrote the logic. Ken Arnold added the screen oriented interface.

NAME

doctor — interact with a psychoanalyst

SYNOPSIS

/usr/games/doctor

DESCRIPTION

Doctor is a lisp-language version of the legendary ELIZA program of Joseph Weizenbaum. This script "simulates" a Rogerian psychoanalyst. Type in lower case, and when you get tired or bored, type your interrupt character (either control-C or Rubout). Remember to type two carriage returns when you want it to answer.

In order to run this you must have a Franz Lisp system in /usr/ucb/lisp.

AUTHORS

Adapted for Lisp by Jon L White, moved to Franz by John Foderaro, from an original script by Joseph Weizenbaum.

NAME

fish — play "Go Fish"

SYNOPSIS

/usr/games/fish

DESCRIPTION

Fish plays the game of "Go Fish", a childrens' card game. The Object is to accumulate 'books' of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the 'pool' of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing a, 2, 3, 4, 5, 6, 7, 8, 9, 10, j, q, or k when asked. Hitting return gives you information about the size of my hand and the pool, and tells you about my books. Saying 'p' as a first guess puts you into 'pro' level; The default is pretty dumb.

NAME

fortune − print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune [−] [−wslao]

DESCRIPTION

Fortune with no arguments prints out a random adage. The flags mean:

−w Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.

−s Short messages only.

−l Long messages only.

−o Choose from an alternate list of adages, often used for potentially offensive ones.

−a Choose from either list of adages.

FILES

/usr/games/lib/fortunes.dat

AUTHOR

Ken Arnold

NAME

hangman — Computer version of the game hangman

SYNOPSIS

/usr/games/hangman

DESCRIPTION

In *hangman,* the computer picks a word from the on-line word list and you must try to guess it. The computer keeps track of which letters have been guessed and how many wrong guesses you have made on the screen in a graphic fashion.

FILES

/usr/dict/words	On-line word list

AUTHOR

Ken Arnold

NAME
mille − play Mille Bournes

SYNOPSIS
/usr/games/mille [file]

DESCRIPTION
Mille plays a two-handed game reminiscent of the Parker Brother's game of Mille Bournes with you. The rules are described below. If a file name is given on the command line, the game saved in that file is started.

When a game is started up, the bottom of the score window will contain a list of commands. They are:

P	Pick a card from the deck. This card is placed in the 'P' slot in your hand.
D	Discard a card from your hand. To indicate which card, type the number of the card in the hand (or "P" for the just-picked card) followed by a <RETURN> or <SPACE>. The <RETURN or <SPACE> is required to allow recovery from typos which can be very expensive, like discarding safeties.
U	Use a card. The card is again indicated by its number, followed by a <RETURN> or <SPACE>.
O	Toggle ordering the hand. By default off, if turned on it will sort the cards in your hand appropriately. This is not recommended for the impatient on slow terminals.
Q	Quit the game. This will ask for confirmation, just to be sure. Hitting <DELETE> (or <RUBOUT>) is equivalent.
S	Save the game in a file. If the game was started from a file, you will be given an opportunity to save it on the same file. If you don't wish to, or you did not start from a file, you will be asked for the file name. If you type a <RETURN> without a name, the save will be terminated and the game resumed.
R	Redraw the screen from scratch. The command ^L (control 'L') will also work.
W	Toggle window type. This switches the score window between the startup window (with all the command names) and the end-of-game window. Using the end-of-game window saves time by eliminating the switch at the end of the game to show the final score. Recommended for hackers and other miscreants.

If you make a mistake, an error message will be printed on the last line of the score window, and a bell will beep.

At the end of each hand or game, you will be asked if you wish to play another. If not, it will ask you if you want to save the game. If you do, and the save is unsuccessful, play will be resumed as if you had said you wanted to play another hand/game. This allows you to use the "S" command to reattempt the save.

AUTHOR
Ken Arnold
(The game itself is a product of Parker Brothers, Inc.)

SEE ALSO
curses(3X), *Screen Updating and Cursor Movement Optimization: A Library Package*, Ken Arnold

CARDS
Here is some useful information. The number in parentheses after the card name is the number of that card in the deck:

Hazard	Repair	Safety
Out of Gas (2)	Gasoline (6)	Extra Tank (1)
Flat Tire (2)	Spare Tire (6)	Puncture Proof (1)
Accident (2)	Repairs (6)	Driving Ace (1)
Stop (4)	Go (14)	Right of Way (1)
Speed Limit (3)	End of Limit (6)	

25 – (10), 50 – (10), 75 – (10), 100 – (12), 200 – (4)

RULES

Object: The point of game is to get a total of 5000 points in several hands. Each hand is a race to put down exactly 700 miles before your opponent does. Beyond the points gained by putting down milestones, there are several other ways of making points.

Overview: The game is played with a deck of 101 cards. *Distance* cards represent a number of miles traveled. They come in denominations of 25, 50, 75, 100, and 200. When one is played, it adds that many miles to the player's trip so far this hand. *Hazard* cards are used to prevent your opponent from putting down Distance cards. They can only be played if your opponent has a *Go* card on top of the Battle pile. The cards are *Out of Gas*, *Accident*, *Flat Tire*, *Speed Limit*, and *Stop*. *Remedy* cards fix problems caused by Hazard cards played on you by your opponent. The cards are *Gasoline*, *Repairs*, *Spare Tire*, *End of Limit*, and *Go*. *Safety* cards prevent your opponent from putting specific Hazard cards on you in the first place. They are *Extra Tank*, *Driving Ace*, *Puncture Proof*, and *Right of Way*, and there are only one of each in the deck.

Board Layout: The board is split into several areas. From top to bottom, they are: **SAFETY AREA** (unlabeled): This is where the safeties will be placed as they are played. **HAND**: These are the cards in your hand. **BATTLE**: This is the Battle pile. All the Hazard and Remedy Cards are played here, except the *Speed Limit* and *End of Limit* cards. Only the top card is displayed, as it is the only effective one. **SPEED**: The Speed pile. The *Speed Limit* and *End of Limit* cards are played here to control the speed at which the player is allowed to put down miles. **MILEAGE**: Miles are placed here. The total of the numbers shown here is the distance traveled so far.

Play: The first pick alternates between the two players. Each turn usually starts with a pick from the deck. The player then plays a card, or if this is not possible or desirable, discards one. Normally, a play or discard of a single card constitutes a turn. If the card played is a safety, however, the same player takes another turn immediately.

This repeats until one of the players reaches 700 points or the deck runs out. If someone reaces 700, they have the option of going for an *Extension*, which means that the play continues until someone reaches 1000 miles.

Hazard and Remedy Cards: Hazard Cards are played on your opponent's Battle and Speed piles. Remedy Cards are used for undoing the effects of your opponent's nastyness.

Go (Green Light) must be the top card on your Battle pile for you to play any mileage, unless you have played the *Right of Way* card (see below).

Stop is played on your opponent's *Go* card to prevent them from playing mileage until they play a *Go* card.

Speed Limit is played on your opponent's Speed pile. Until they play an *End of Limit* they can only play 25 or 50 mile cards, presuming their *Go* card allows them to do even that.

End of Limit is played on your Speed pile to nullify a *Speed Limit* played by your opponent.

Out of Gas is played on your opponent's *Go* card. They must then play a *Gasoline* card, and then a *Go* card before they can play any more mileage.

Flat Tire is played on your opponent's *Go* card. They must then play a *Spare Tire* card, and then a *Go* card before they can play any more mileage.

Accident is played on your opponent's *Go* card. They must then play a *Repairs* card, and then a *Go* card before they can play any more mileage.

Safety Cards: Safety cards prevent your opponent from playing the corresponding Hazard cards on you for the rest of the hand. It cancels an attack in progress, and *always entitles the player to an extra turn*.

Right of Way prevents your opponent from playing both *Stop* and *Speed Limit* cards on you. It also acts as a permanent *Go* card for the rest of the hand, so you can play mileage as long as there is not a Hazard card on top of your Battle pile. In this case only, your opponent can play Hazard cards directly on a Remedy card besides a Go card.

Extra Tank When played, your opponent cannot play an *Out of Gas* on your Battle Pile.

Puncture Proof When played, your opponent cannot play a *Flat Tire* on your Battle Pile.

Driving Ace When played, your opponent cannot play an *Accident* on your Battle Pile.

Distance Cards: Distance cards are played when you have a *Go* card on your Battle pile, or a Right of Way in your Safety area and are not stopped by a Hazard Card. They can be played in any combination that totals exactly 700 miles, except that *you cannot play more than two 200 mile cards in one hand.* A hand ends whenever one player gets exactly 700 miles or the deck runs out. In that case, play continues until neither someone reaches 700, or neither player can use any cards in their hand. If the trip is completed after the deck runs out, this is called *Delayed Action*.

Coup Fourré: This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In Mille Bournes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, even *before* you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points (see "Scoring" below).

Scoring: Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window have the following meanings:

Milestones Played: Each player scores as many miles as they played before the trip ended.
Each Safety: 100 points for each safety in the Safety area.
All 4 Safeties: 300 points if all four safeties are played.
Each Coup Fouré: 300 points for each Coup Fouré accomplished.

The following bonus scores can apply only to the winning player.

Trip Completed: 400 points bonus for completing the trip to 700 or 1000.
Safe Trip: 300 points bonus for completing the trip without using any 200 mile cards.
Delayed Action: 300 points bonus for finishing after the deck was exhausted.
Extension: 200 points bonus for completing a 1000 mile trip.
Shut-Out: 500 points bonus for completing the trip before your opponent played any mileage cards.

Running totals are also kept for the current score for each player for the hand (**Hand Total**), the game (**Overall Total**), and number of games won (**Games**).

NAME

monop − Monopoly game

SYNOPSIS

/usr/games/monop [file]

DESCRIPTION

Monop is reminiscent of the Parker Brother's game Monopoly, and monitors a game between 1 to 9 users. It is assumed that the rules of Monopoly are known. The game follows the standard rules, with the exception that, if a property would go up for auction and there are only two solvent players, no auction is held and the property remains unowned.

The game, in effect, lends the player money, so it is possible to buy something which you cannot afford. However, as soon as a person goes into debt, he must "fix the problem", *i.e.*, make himself solvent, before play can continue. If this is not possible, the player's property reverts to his debtee, either a player or the bank. A player can resign at any time to any person or the bank, which puts the property back on the board, unowned.

Any time that the response to a question is a *string*, e.g., a name, place or person, you can type '?' to get a list of valid answers. It is not possible to input a negative number, nor is it ever necessary.

A Summary of Commands:

quit: quit game: This allows you to quit the game. It asks you if you're sure.

print: print board: This prints out the current board. The columns have the following meanings (column headings are the same for the **where**, **own holdings**, and **holdings** commands):

Name	The first ten characters of the name of the square
Own	The *number* of the owner of the property.
Price	The cost of the property (if any)
Mg	This field has a '*' in it if the property is mortgaged
#	If the property is a Utility or Railroad, this is the number of such owned by the owner. If the property is land, this is the number of houses on it.
Rent	Current rent on the property. If it is not owned, there is no rent.

where: where players are: Tells you where all the players are. A '*' indicates the current player.

own holdings:
List your own holdings, *i.e.*, money, get-out-of-jail-free cards, and property.

holdings: holdings list: Look at anyone's holdings. It will ask you whose holdings you wish to look at. When you are finished, type "done".

shell: shell escape: Escape to a shell. When the shell dies, the program continues where you left off.

mortgage: mortgage property: Sets up a list of mortgageable property, and asks which you wish to mortgage.

unmortgage:
unmortgage property: Unmortgage mortgaged property.

buy: buy houses: Sets up a list of monopolies on which you can buy houses. If there is more than one, it asks you which you want to buy for. It then asks you how many for each piece of property, giving the current amount in parentheses after the property name. If you build in an unbalanced manner (a disparity of more than one house within the same monopoly), it asks you to re-input things.

sell: sell houses: Sets up a list of monopolies from which you can sell houses. it operates in an analogous manner to *buy*

card: card for jail: Use a get-out-of-jail-free card to get out of jail. If you're not in jail, or you don't have one, it tells you so.

pay: pay for jail: Pay $50 to get out of jail, from whence you are put on Just Visiting. Difficult to do if you're not there.

trade: This allows you to trade with another player. It asks you whom you wish to trade with, and then asks you what each wishes to give up. You can get a summary at the end, and, in all cases, it asks for confirmation of the trade before doing it.

resign: Resign to another player or the bank. If you resign to the bank, all property reverts to its virgin state, and get-out-of-jail free cards revert to the deck.

save: save game: Save the current game in a file for later play. You can continue play after saving, either by adding the file in which you saved the game after the *monop* command, or by using the *restore* command (see below). It will ask you which file you wish to save it in, and, if the file exists, confirm that you wish to overwrite it.

restore: restore game: Read in a previously saved game from a file. It leaves the file intact.

roll: Roll the dice and move forward to your new location. If you simply hit the <RETURN> key instead of a command, it is the same as typing *roll*.

AUTHOR

Ken Arnold

FILES

/usr/games/lib/cards.pck	Chance and Community Chest cards

BUGS

No command can be given an argument instead of a response to a query.

NAME

number − convert Arabic numerals to English

SYNOPSIS

/usr/games/number

DESCRIPTION

Number copies the standard input to the standard output, changing each decimal number to a fully spelled out version.

NAME

quiz — test your knowledge

SYNOPSIS

/usr/games/quiz [**−i** file] [**−t**] [category1 category2]

DESCRIPTION

Quiz gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2.* If no categories are specified, *quiz* gives instructions and lists the available categories.

Quiz tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, *quiz* reports a score and terminates.

The **−t** flag specifies 'tutorial' mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **−i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line      = category newline | category ':' line
category  = alternate | category '|' alternate
alternate = empty | alternate primary
primary   = character | '[' category ']' | option
option    = '{' category '}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash '\' is used as with *sh*(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, *quiz* will refrain from asking it.

FILES

/usr/games/quiz.k/*

BUGS

The construct 'a|ab' doesn't work in an information file. Use 'a{b}'.

NAME

rain – animated raindrops display

SYNOPSIS

/usr/games/rain

DESCRIPTION

Rain's display is modeled after the VAX/VMS program of the same name. The terminal has to be set for 9600 baud to obtain the proper effect.

As with all programs that use *termcap*, the TERM environment variable must be set (and exported) to the type of the terminal being used.

FILES

/etc/termcap

AUTHOR

Eric P. Scott

NAME

rogue − Exploring The Dungeons of Doom

SYNOPSIS

/usr/games/rogue [−r] [*save_file*] **[−s] [−d]**

DESCRIPTION

Rogue is a computer fantasy game with a new twist. It is crt oriented and the object of the game is to survive the attacks of various monsters and get a lot of gold, rather than the puzzle solving orientation of most computer fantasy games.

To get started you really only need to know two commands. The command ? will give you a list of the available commands and the command / will identify the things you see on the screen.

To win the game (as opposed to merely playing to beat other people high scores) you must locate the Amulet of Yendor which is somewhere below the 20th level of the dungeon and get it out. Nobody has achieved this yet and if somebody does, they will probably go down in history as a hero among heros.

When the game ends, either by your death, when you quit, or if you (by some miracle) manage to win, *rogue* will give you alist of the top-ten scorers. The scoring is based entirely upon how much gold you get. There is a 10% penalty for getting yourself killed.

If *save_file* is specified, rogue will be restored from the specified saved game file. If the **−r** option is used, the save game file is presumed to be the default.

The **−s** option will print out the list of scores.

The **−d** option will kill you and try to add you to the score file.

For more detailed directions, read the document *A Guide to the Dungeons of Doom.*

AUTHORS

Michael C. Toy, Kenneth C. R. C. Arnold, Glenn Wichman

FILES

/usr/games/lib/rogue_roll	Score file
˜/rogue.save	Default save file

SEE ALSO

Michael C. Toy and Kenneth C. R. C. Arnold, *A guide to the Dungeons of Doom*

BUGS

Probably infinite. However, that Floating Eyes sometimes transfix you permanently is *not* a bug. It's a feature.

NAME

snake, snscore — display chase game

SYNOPSIS

/usr/games/snake [**−w***n*] [**−l***n*]
/usr/games/snscore

DESCRIPTION

Snake is a display-based game which must be played on a CRT terminal from among those supported by vi(1). The object of the game is to make as much money as possible without getting eaten by the snake. The **−l** and **−w** options allow you to specify the length and width of the field. By default the entire screen (except for the last column) is used.

You are represented on the screen by an I. The snake is 6 squares long and is represented by S's. The money is $, and an exit is #. Your score is posted in the upper left hand corner.

You can move around using the same conventions as vi(1), the h, j, k, and l keys work, as do the arrow keys. Other possibilities include:

sefc	These keys are like hjkl but form a directed pad around the d key.
HJKL	These keys move you all the way in the indicated direction to the same row or column as the money. This does *not* let you jump away from the snake, but rather saves you from having to type a key repeatedly. The snake still gets all his turns.
SEFC	Likewise for the upper case versions on the left.
ATPB	These keys move you to the four edges of the screen. Their position on the keyboard is the mnemonic, e.g. P is at the far right of the keyboard.
x	This lets you quit the game at any time.
p	Points in a direction you might want to go.
w	Space warp to get out of tight squeezes, at a price.
!	Shell escape
^Z	Suspend the snake game, on systems which support it. Otherwise an interactive shell is started up.

To earn money, move to the same square the money is on. A new $ will appear when you earn the current one. As you get richer, the snake gets hungrier. To leave the game, move to the exit (#).

A record is kept of the personal best score of each player. Scores are only counted if you leave at the exit, getting eaten by the snake is worth nothing.

As in pinball, matching the last digit of your score to the number which appears after the game is worth a bonus.

To see who wastes time playing snake, run */usr/games/snscore* .

FILES

/usr/games/lib/snakerawscores	database of personal bests
/usr/games/lib/snake.log	log of games played
/usr/games/busy	program to determine if system too busy

BUGS

When playing on a small screen, it's hard to tell when you hit the edge of the screen.

The scoring function takes into account the size of the screen. A perfect function to do this equitably has not been devised.

NAME

trek — trekkie game

SYNOPSIS

/usr/games/trek [[**−a**] file]

DESCRIPTION

Trek is a game of space glory and war. Below is a summary of commands. For complete documentation, see *Trek* by Eric Allman.

If a filename is given, a log of the game is written onto that file. If the **−a** flag is given before the filename, that file is appended to, not truncated.

The game will ask you what length game you would like. Valid responses are "short", "medium", and "long". You may also type "restart", which restarts a previously saved game. You will then be prompted for the skill, to which you must respond "novice", "fair", "good", "expert", "commadore", or "impossible". You should normally start out with a novice and work up.

In general, throughout the game, if you forget what is appropriate the game will tell you what it expects if you just type in a question mark.

AUTHOR

Eric Allman

SEE ALSO

/usr/doc/trek

COMMAND SUMMARY

abandon	**capture**
cloak up/down	
computer request; ...	damages
destruct	**dock**
help	impulse course distance
lrscan	move course distance
phasers automatic amount	
phasers manual amt1 course1 spread1 ...	
torpedo course [yes] angle/no	
ram course distance	rest time
shell	shields up/down
srscan [yes/no]	
status	**terminate** yes/no
undock	visual course
warp warp_factor	

NAME

worm − Play the growing worm game

SYNOPSIS

/usr/games/worm [*size*]

DESCRIPTION

In *worm,* you are a little worm, your body is the "o"'s on the screen and your head is the "@". You move with the hjkl keys (as in the game snake). If you don't press any keys, you continue in the direction you last moved. The upper case HJKL keys move you as if you had pressed several (9 for HL and 5 for JK) of the corresponding lower case key (unless you run into a digit, then it stops).

On the screen you will see a digit, if your worm eats the digit is will grow longer, the actual amount longer depends on which digit it was that you ate. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into either the sides of the screen, or itself. The current score (how much the worm has grown) is kept in the upper left corner of the screen.

The optional argument, if present, is the initial length of the worm.

BUGS

If the initial length of the worm is set to less than one or more than 75, various strange things happen.

NAME

worms − animate worms on a display terminal

SYNOPSIS

/usr/games/worms [−field] [−length #] [−number #] [−trail]

DESCRIPTION

Brian Horn (cithep!bdh) showed me a *TOPS-20* program on the DEC-2136 machine called *WORM*, and suggested that I write a similar program that would run under *Unix*. I did, and no apologies.

−field makes a "field" for the worm(s) to eat; **−trail** causes each worm to leave a trail behind it. You can figure out the rest by yourself.

FILES

/etc/termcap

AUTHOR

Eric P. Scott

SEE ALSO

Snails, by Karl Heuer

BUGS

The lower-right-hand character position will not be updated properly on a terminal that wraps at the right margin.

Terminal initialization is not performed.

NAME

wump — the game of hunt-the-wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

Wump plays the game of 'Hunt the Wumpus.' A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in *People's Computer Company, 2,* 2 (November 1973).

NAME

zork — the game of dungeon

SYNOPSIS

/usr/games/zork

DESCRIPTION

Dungeon is a computer fantasy simulation based on Adventure and on Dungeons & Dragons, originally written by Lebling, Blank, and Anderson of MIT. In it you explore a dungeon made up of various rooms, caves, rivers, and so on. The object of the game is to collect as much treasure as possible and stow it safely in the trophy case (and, of course, to stay alive.)

Figuring out the rules is part of the game, but if you are stuck, you should start off with "open mailbox", "take leaflet", and then "read leaflet". Additional useful commands that are not documented include:

quit	(to end the game)
!cmd	(the usual shell escape convention)
>	(to save a game)
<	(to restore a game)

FILES

/usr/games/lib/d*

NAME

miscellaneous − miscellaneous useful information pages

DESCRIPTION

This section contains miscellaneous documentation, mostly in the area of text processing macro packages for *troff*(1).

ascii	map of ASCII character set
environ	user environment
eqnchar	special character definitions for eqn
hier	file system hierarchy
mailaddr	mail addressing description
man	macros to typeset manual pages
me	macros for formatting papers
ms	macros for formatting manuscripts
term	conventional names for terminals

NAME

ascii − map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

Ascii is a map of the ASCII character set, 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 *execve*(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=:/usr/ucb:/bin:/usr/bin.
HOME	A user's login directory, set by *login*(1) from the password file *passwd*(5).
TERM	The kind of terminal for which output is to be prepared. This information is used by commands, such as *nroff* or *plot*(1G), which may exploit special terminal capabilities. See */etc/termcap* (*termcap*(5)) for a list of terminal types.
SHELL	The file name of the users login shell.
TERMCAP	The string describing the terminal in TERM, or the name of the termcap file, see *termcap*(5),*termcap*(3X).
EXINIT	A startup list of commands read by *ex*(1), *edit*(1), and *vi*(1).
USER	The login name of the user.
PRINTER	The name of the default printer to be used by *lpr*(1), *lpq*(1), and *lprm*(1).

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 *execve*(2). It is unwise to conflict with certain *sh*(1) variables that are frequently exported by '.profile' files: MAIL, PS1, PS2, IFS.

SEE ALSO
csh(1), ex(1), login(1), sh(1), execve(2), system(3), termcap(3X), termcap(5)

NAME

eqnchar − special character definitions for eqn

SYNOPSIS

eqn /usr/pub/eqnchar [files] **|** **troff** [options]

neqn /usr/pub/eqnchar [files] **|** **nroff** [options]

DESCRIPTION

Eqnchar contains *troff* and *nroff* character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with *eqn* and *neqn*. It contains definitions for the following characters

ciplus	⊕	||	‖	*square*	□
citimes	⊗	*langle*	⟨	*circle*	○
wig	∼	*rangle*	⟩	*blot*	■
-wig	≃	*hbar*	ℏ	*bullet*	●
>wig	≳	*ppd*	⊥	*prop*	∝
<wig	≲	*<->*	↔	*empty*	∅
=wig	≅	*<=>*	⇔	*member*	∈
star	*	|<	≮	*nomem*	∉
bigstar	✱	|>	≯	*cup*	∪
=dot	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	∟	*incl*	⊑
andsign	∧	*3dot*	⋮	*subset*	⊂
=del	≜	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊆
oppE	∃	*3quarter*	¾	*!supset*	⊇
angstrom	Å	*degree*	°		

FILES

/usr/pub/eqnchar

SEE ALSO

troff(1), eqn(1)

NAME

hier − file system hierarchy

DESCRIPTION

The following outline gives a quick tour through a representative directory hierarchy.

```
/        root
/vmunix
         the kernel binary (UNIX itself)
/lost+found
         directory for connecting detached files for fsck(8)
/dev/    devices (4)
         MAKEDEV
                  shell script to create special files
         MAKEDEV.local
                  site specific part of MAKEDEV
         console
                  main console, tty(4)
         tty*     terminals, tty(4)
         hp*      disks, hp(4)
         rhp*     raw disks, hp(4)
         up*      UNIBUS disks up(4)
         ...
/bin/    utility programs, cf /usr/bin/ (1)
         as       assembler
         cc       C compiler executive, cf /lib/ccom, /lib/cpp, /lib/c2
         csh      C shell
         ...
/lib/    object libraries and other stuff, cf /usr/lib/
         libc.a   system calls, standard I/O, etc. (2,3,3S)
         ...
         ccom     C compiler proper
         cpp      C preprocessor
         c2       C code improver
         ...
/etc/    essential data and maintenance utilities; sect (8)
         dump     dump program dump(8)
         passwd   password file, passwd(5)
         group    group file, group(5)
         motd     message of the day, login(1)
         termcap
                  description of terminal capabilities, termcap(5)
         ttytype  table of what kind of terminal is on each port, ttytype(5)
         mtab     mounted file table, mtab(5)
         dumpdates
                  dump history, dump(8)
         fstab    file system configuration table fstab(5)
         disktab  disk characteristics and partition tables, disktab(5)
         hosts    host name to network address mapping file, hosts(5)
         networks
                  network name to network number mapping file, networks(5)
         protocols
                  protocol name to protocol number mapping file, protocols(5)
         services
```

```
                network services definition file, services(5)
        remote  names and description of remote hosts for tip(1C), remote(5)
        phones  private phone numbers for remote hosts, as described in phones(5)
        ttys    properties of terminals, ttys(5)
        getty   part of login, getty(8)
        init    the parent of all processes, init(8)
        rc      shell program to bring the system up
        rc.local site dependent portion of rc
        cron    the clock daemon, cron(8)
        mount   mount(8)
        ...
/sys/   system source
        h/      header (include) files
                acct.h  acct(5)
                stat.h  stat(2)
                ...
        sys/    machine independent system source
                init_main.c
                uipc_socket.c
                ufs_syscalls.c
                ...
        conf/   site configuration files
                GENERIC
                ...
        net/    general network source
        netinet/
                DARPA Internet network source
        netimp/
                network code related to use of an IMP
                if_imp.c
                if_imphost.c
                if_imphost.h
                ...
        vax/    source specific to the VAX
                locore.s
                machdep.c
                ...
        vaxuba/
                device drivers for hardware which resides on the UNIBUS
                uba.c
                dh.c
                up.c
                ...
        vaxmba/
                device drivers for hardware which resides on the MASBUS
                mba.c
                hp.c
                ht.c
                ...
        vaxif   network interface drivers for the VAX
                if_en.c
                if_ec.c
                if_vv.c
                ...
/tmp/   temporary files, usually on a fast device, cf /usr/tmp/
        e*      used by ed(1)
        ctm*    used by cc(1)
        ...
/usr/   general-pupose directory, usually a mounted file system
        adm/    administrative information
                wtmp    login history, utmp(5)
                messages
                        hardware error messages
                tracct  phototypesetter accounting, troff(1)
                lpacct  line printer accounting lpr(1)
                vaacct, vpacct
                        varian and versatec accounting vpr(1), vtroff(1), pac(8)
/usr    /bin
        utility programs, to keep /bin/ small
        tmp/    temporaries, to keep /tmp/ small
                stm*    used by sort(1)
                raster  used by plot(1G)
        dict/   word lists, etc.
                words   principal word list, used by look(1)
                spellhist
                        history file for spell(1)
        games/
                hangman
                lib/    library of stuff for the games
                        quiz.k/ what quiz(6) knows
                                index   category index
                                africa  countries and capitals
                                ...
                        ...
                ...
        include/
                standard #include files
                a.out.h object file layout, a.out(5)
                stdio.h standard I/O, intro(3S)
                math.h  (3M)
                ...
                sys/    system-defined layouts, cf /sys/h
                net/    symbolic link to sys/net
                machine/
                        symbolic link to sys/machine
                ...
        lib/    object libraries and stuff, to keep /lib/ small
                atrun   scheduler for at(1)
                lint/   utility files for lint
                        lint[12]
                                subprocesses for lint(1)
                        llib-lc dummy declarations for /lib/libc.a, used by lint(1)
                        llib-lm dummy declarations for /lib/libc.m
                        ...
                struct/ passes of struct(1)
                ...
                tmac/   macros for troff(1)
                        tmac.an
                                macros for man(7)
                        tmac.s  macros for ms(7)
                        ...
                font/   fonts for troff(1)
                        ftR     Times Roman
                        ftB     Times Bold
                        ...
                uucp/   programs and data for uucp(1C)
                        L.sys   remote system names and numbers
                        uucico  the real copy program
                        ...
                units   conversion tables for units(1)
                eign    list of English words to be ignored by ptx(1)
/usr/   man/
        volume 1 of this manual, man(1)
                man0/   general
                        intro   introduction to volume 1, ms(7) format
                        xx      template for manual page
                man1/   chapter 1
                        as.1
                        mount.1m
                        ...
                ...
                cat1/   preformatted pages for section 1
                ...
        msgs/   messages, cf msgs(1)
                bounds highest and lowest message
        new/    binaries of new versions of programs
        preserve/
                editor temporaries preserved here after crashes/hangups
        public/ binaries of user programs - write permission to everyone
        spool/  delayed execution files
                at/     used by at(1)
                lpd/    used by lpr(1)
                        lock    present when line printer is active
                        cf*     copy of file to be printed, if necessary
                        df*     daemon control file, lpd(8)
                        tf*     transient control file, while lpr is working
                uucp/   work files and staging area for uucp(1C)
                        LOGFILE
                                summary log
                        LOG.*   log file for one transaction
                mail/   mailboxes for mail(1)
                        name    mail file for user name
                        name.lock
                                lock file while name is receiving mail
                secretmail/
                        like mail/
                uucp/   work files and staging area for uucp(1C)
                        LOGFILE
                                summary log
                        LOG.*   log file for one transaction
                        mqueue/
                                mail queue for sendmail(8)
        wd      initial working directory of a user, typically wd is the user's login name
                .profile set environment for sh(1), environ(7)
                .project
                        what you are doing (used by ( finger(1) )
                .cshrc  startup file for csh(1)
                .exrc   startup file for ex(1)
                .plan   what your short-term plans are (used by finger(1) )
                .netrc  startup file for various network programs
                .msgsrc
                        startup file for msgs(1)
                .mailrc startup file for mail(1)
                calendar
                        user's datebook for calendar(1)
        doc/    papers, mostly in volume 2 of this manual, typically in ms(7) format
                as/     assembler manual
                c       C manual
                ...
/usr/   src/
        source programs for utilities, etc.
        bin/    source of commands in /bin
                as/     assembler
                ar.c    source for ar(1)
                ...
        usr.bin/
                source for commands in /usr/bin
                troff/  source for nroff and troff(1)
                        font/   source for font tables, /usr/lib/font/
                                ftR.c   Roman
                                ...
                        term/   terminal characteristics tables, /usr/lib/term/
                                tab300.c
                                        DASI 300
                                ...
                        ...
        ucb     source for programs in /usr/ucb
        games/  source for /usr/games
        lib/    source for programs and archives in /lib
                libc/   C runtime library
                        csu/    startup and wrapup routines needed with every C program
                                crt0.s  regular startup
                                mcrt0.s modified startup for cc −p
                        sys/    system calls (2)
                                access.s
                                brk.s
                                ...
                stdio/  standard I/O functions (3S)
```

```
                    fgets.c
                    fopen.c
                    ...
              gen/  other functions in (3)
                    abs.c
                    ...
              net/  network functions in (3N)
                    gethostbyname.c
                    ...
        local/  source which isn't normally distributed
        new/    source for new versions of commands and library routines
        old/    source for old versions of commands and library routines
        ucb/    binaries of programs developed at UCB
              ...
              edit  editor for beginners
              ex    command editor for experienced users
              ...
              mail  mail reading/sending subsystem
              man   on line documentation
              ...
              pi    Pascal translator
              px    Pascal interpreter
              ...
              vi    visual editor
```

SEE ALSO

ls(1), apropos(1), whatis(1), whereis(1), finger(1), which(1), ncheck(8), find(1), grep(1)

BUGS

The position of files is subject to change without notice.

NAME

mailaddr – mail addressing description

DESCRIPTION

Mail addresses are based on the ARPANET protocol listed at the end of this manual page. These addresses are in the general format

user@domain

where a domain is a hierarchical dot separated list of subdomains. For example, the address

eric@monet.Berkeley.ARPA

is normally interpreted from right to left: the message should go to the ARPA name tables (which do not correspond exactly to the physical ARPANET), then to the Berkeley gateway, after which it should go to the local host monet. When the message reaches monet it is delivered to the user "eric".

Unlike some other forms of addressing, this does not imply any routing. Thus, although this address is specified as an ARPA address, it might travel by an alternate route if that was more convenient or efficient. For example, at Berkeley the associated message would probably go directly to monet over the Ethernet rather than going via the Berkeley ARPANET gateway.

Abbreviation. Under certain circumstances it may not be necessary to type the entire domain name. In general anything following the first dot may be omitted if it is the same as the domain from which you are sending the message. For example, a user on "calder.Berkeley.ARPA" could send to "eric@monet" without adding the ".Berkeley.ARPA" since it is the same on both sending and receiving hosts.

Certain other abbreviations may be permitted as special cases. For example, at Berkeley ARPANET hosts can be referenced without adding the ".ARPA" as long as their names do not conflict with a local host name.

Compatibility. Certain old address formats are converted to the new format to provide compatibility with the previous mail system. In particular,

host:user

is converted to

user@host

to be consistent with the *rcp*(1C) command.

Also, the syntax:

host!user

is converted to:

user@host.UUCP

This is normally converted back to the "host!user" form before being sent on for compatibility with older UUCP hosts.

The current implementation is not able to route messages automatically through the UUCP network. Until that time you must explicitly tell the mail system which hosts to send your message through to get to your final destination.

Case Distinctions. Domain names (i.e., anything after the "@" sign) may be given in any mixture of upper and lower case with the exception of UUCP hostnames. Most hosts accept any mixture of case in user names, with the notable exception of MULTICS sites.

Differences with ARPA Protocols. Although the UNIX addressing scheme is based on the ARPA mail addressing protocols, there are some significant differences.

At the time of this writing the only "top level" domain defined by ARPA is the ".ARPA" domain itself. This is further restricted to having only one level of host specifier. That is, the only addresses that ARPA accepts at this time must be in the format "user@host.ARPA" (where "host" is one word). In particular, addresses such as:

eric@monet.Berkeley.ARPA

are not currently legal under the ARPA protocols. For this reason, these addresses are converted to a different format on output to the ARPANET, typically:

eric%monet@Berkeley.ARPA

Route-addrs. Under some circumstances it may be necessary to route a message through several hosts to get it to the final destination. Normally this routing is done automatically, but sometimes it is desirable to route the message manually. An address that shows these relays are termed "route-addrs." These use the syntax:

<@hosta,@hostb:user@hostc>

This specifies that the message should be sent to hosta, from there to hostb, and finally to hostc. This path is forced even if there is a more efficient path to hostc.

Route-addrs occur frequently on return addresses, since these are generally augmented by the software at each host. It is generally possible to ignore all but the "user@host" part of the address to determine the actual sender.

Postmaster. Every site is required to have a user or user alias designated "postmaster" to which problems with the mail system may be addressed.

CSNET. Messages to CSNET sites can be sent to "user.host@UDel-Relay".

BERKELEY

The following comments apply only to the Berkeley environment.

Host Names. Many of the old familiar host names are being phased out. In particular, single character names as used in Berknet are incompatible with the larger world of which Berkeley is now a member. For this reason the following names are being obsoleted. You should notify any correspondents of your new address as soon as possible.

OLD	NEW	j ingvax	ucbingres
p	ucbcad	r arpavax	ucbarpa
v csvax	ucbernie		
n	ucbkim	y	ucbcory

The old addresses will be rejected as unknown hosts sometime in the near future.

What's My Address? If you are on a local machine, say monet, your address is

yourname@monet.Berkeley.ARPA

However, since most of the world does not have the new software in place yet, you will have to give correspondents slightly different addresses. From the ARPANET, your address would be:

yourname%monet@Berkeley.ARPA

From UUCP, your address would be:

ucbvax!yourname%monet

Computer Center. The Berkeley Computer Center is in a subdomain of Berkeley. Messages to the computer center should be addressed to:

user%host.CC@Berkeley.ARPA

The alternate syntax:

user@host.CC

may be used if the message is sent from inside Berkeley.

For the time being Computer Center hosts are known within the Berkeley domain, i.e., the ".CC" is optional. However, it is likely that this situation will change with time as both the Computer Science department and the Computer Center grow.

Bitnet. Hosts on bitnet may be accessed using:

user@host.BITNET

SEE ALSO

mail(1), sendmail(8); Crocker, D. H., *Standard for the Format of Arpa Internet Text Messages,* RFC822.

NAME

man − macros to typeset manual

SYNOPSIS

nroff −man file ...

troff −man file ...

DESCRIPTION

These macros are used to lay out pages of this manual. A skeleton page may be found in the file /usr/man/man0/xx.

Any text argument *t* may be zero to six words. Quotes may be used to include blanks in a 'word'. If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way .I may be used to italicize a whole line, or .SM followed by .B to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by **−man**:

*R	'®', '(Reg)' in *nroff*.
*S	Change to default type size.

FILES

/usr/lib/tmac/tmac.an
/usr/man/man0/xx

SEE ALSO

troff(1), man(1)

BUGS

Relative indents don't nest.

REQUESTS

Request	Cause Break	If no Argument	Explanation
.B *t*	no	*t*=n.t.l.*	Text *t* is bold.
.BI *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and italic.
.BR *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and Roman.
.DT	no	.5i 1i...	Restore default tabs.
.HP *i*	yes	*i*=p.i.*	Set prevailing indent to *i*. Begin paragraph with hanging indent.
.I *t*	no	*t*=n.t.l.	Text *t* is italic.
.IB *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and bold.
.IP *x i*	yes	*x*=""	Same as .TP with tag *x*.
.IR *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and Roman.
.LP	yes	-	Same as .PP.
.PD *d*	no	*d*=.4v	Interparagraph distance is *d*.
.PP	yes	-	Begin paragraph. Set prevailing indent to .5i.
.RE	yes	-	End of relative indent. Set prevailing indent to amount of starting .RS.
.RB *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and bold.
.RI *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and italic.
.RS *i*	yes	*i*=p.i.	Start relative indent, move left margin in distance *i*. Set prevailing indent to .5i for nested indents.
.SH *t*	yes	*t*=n.t.l.	Subhead.

.SM *t*	no	*t*=n.t.l.	Text *t* is small.
.TH *n c x v m*	yes	-	Begin page named *n* of chapter *c; x* is extra commentary, e.g. 'local', for page foot center; *v* alters page foot left, e.g. '4th Berkeley Distribution'; *m* alters page head center, e.g. 'Brand X Programmer's Manual'. Set prevailing indent and tabs to .5i.
.TP *i*	yes	*i*=p.i.	Set prevailing indent to *i*. Begin indented paragraph with hanging tag given by next text line. If tag doesn't fit, place it on separate line.

* n.t.l. = next text line; p.i. = prevailing indent

NAME

me − macros for formatting papers

SYNOPSIS

nroff −me [options] file ...
troff −me [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col*(1).

The macro requests are defined below. Many *nroff* and *troff* requests are unsafe in conjunction with this package, however these requests may be used with impunity after the first .pp:

.bp	begin new page
.br	break output line here
.sp n	insert n spacing lines
.ls n	(line spacing) n=1 single, n=2 double space
.na	no alignment of right margin
.ce n	center next n lines
.ul n	underline next n lines
.sz +n	add n to point size

Output of the *eqn, neqn, refer,* and *tbl*(1) preprocessors for equations and tables is acceptable as input.

FILES

/usr/lib/tmac/tmac.e
/usr/lib/me/*

SEE ALSO

eqn(1), troff(1), refer(1), tbl(1)
−me Reference Manual, Eric P. Allman
Writing Papers with Nroff Using −me

REQUESTS

In the following list, "initialization" refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete; see *The −me Reference Manual* for interesting details.

Request	Initial Value	Cause Break	Explanation
.(c	-	yes	Begin centered block
.(d	-	no	Begin delayed text
.(f	-	no	Begin footnote
.(l	-	yes	Begin list
.(q	-	yes	Begin major quote
.(x *x*	-	no	Begin indexed item in index *x*
.(z	-	no	Begin floating keep
.)c	-	yes	End centered block
.)d	-	yes	End delayed text
.)f	-	yes	End footnote
.)l	-	yes	End list
.)q	-	yes	End major quote
.)x	-	yes	End index item
.)z	-	yes	End floating keep
.++ *m H*	-	no	Define paper section. *m* defines the part of the paper, and can be **C** (chapter), **A** (appendix), **P** (preliminary, e.g., abstract, table of contents, etc.), **B**

			(bibliography), **RC** (chapters renumbered from page one each chapter), or **RA** (appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.
.EN	-	yes	Space after equation produced by *eqn* or *neqn*.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.ac *A N*	-	no	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.ba +*n*	0	yes	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column
.bi *x*	no	no	Print *x* in bold italics (nofill only)
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef *'x'y'z'*	""""	no	Set even footer to x y z
.eh *'x'y'z'*	""""	no	Set even header to x y z
.fo *'x'y'z'*	""""	no	Set footer to x y z
.hx	-	no	Suppress headers and footers on next page.
.he *'x'y'z'*	""""	no	Set header to x y z
.hl	-	yes	Draw a horizontal line
.i *x*	no	no	Italicize *x*; if *x* missing, italic text follows.
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form .•*x*. Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of *'x'y'z'*	""""	no	Set odd footer to x y z
.oh *'x'y'z'*	""""	no	Set odd header to x y z
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.
.re	-	no	Reset tabs to default values.
.sc	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sz +*n*	10p	no	Augment the point size by *n* points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in *troff*). (Nofill only).
.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x*.

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.

FILES

/usr/lib/tmac/tmac.x
/usr/lib/ms/x.???

SEE ALSO

eqn(1), refer(1), tbl(1), troff(1)

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
.AM	−	n	better accent mark definitions
.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	UC Berkeley thesis mode
.TS *x*	–	y,y	begin table; if *x* = H table has multi-page header
.UL *x*	–	n	underline *x*, even in *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:

Name	Register Controls	Takes Effect	Default
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)

When using the extended accent mark definitions available with .AM, these strings should come after, rather than before, the letter to be accented.

BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

NAME

term − conventional names for terminals

DESCRIPTION

Certain commands use these terminal names. They are maintained as part of the shell environment (see *sh*(1),*environ*(7)).

adm3a	Lear Seigler Adm-3a
2621	Hewlett-Packard HP262? series terminals
hp	Hewlett-Packard HP264? series terminals
c100	Human Designed Systems Concept 100
h19	Heathkit H19
mime	Microterm mime in enhanced ACT IV mode
1620	DIABLO 1620 (and others using HyType II)
300	DASI/DTC/GSI 300 (and others using HyType I)
33	TELETYPE® Model 33
37	TELETYPE Model 37
43	TELETYPE Model 43
735	Texas Instruments TI735 (and TI725)
745	Texas Instruments TI745
dumb	terminals with no special features
dialup	a terminal on a phone line with no known characteristics
network	a terminal on a network connection with no known characteristics
4014	Tektronix 4014
vt52	Digital Equipment Corp. VT52

The list goes on and on. Consult /etc/termcap (see *termcap*(5)) for an up-to-date and locally correct list.

Commands whose behavior may depend on the terminal either consult TERM in the environment, or accept arguments of the form **−Tterm**, where *term* is one of the names given above.

SEE ALSO

stty(1), tabs(1), plot(1G), sh(1), environ(7) ex(1), clear(1), more(1), ul(1), tset(1), termcap(5), termcap(3X), ttytype(5)
troff(1) for *nroff*

BUGS

The programs that ought to adhere to this nomenclature do so only fitfully.

NAME

intro – introduction to system maintenance and operation commands

DESCRIPTION

This section contains information related to system operation and maintenance. In particular, commands used to create new file systems, *newfs*, *mkfs*, and verify the integrity of the file systems, *fsck*, *icheck*, *dcheck*, and *ncheck* are described here. The section *format* should be consulted when formatting disk packs. The section *crash* should be consulted in understanding how to interpret system crash dumps.

LIST OF PROGRAMS

Program	*Appears on Page*	*Description*
ac	ac.8	login accounting
accton	sa.8	system accounting
adduser	adduser.8	procedure for adding new users
analyze	analyze.8	Virtual UNIX postmortem crash analyzer
arcv	arcv.8	convert archives to new format
arff	arff.8v	archiver and copier for floppy
bad144	bad144.8	read/write dec standard 144 bad sector information
badsect	badsect.8	create files to contain bad sectors
bugfiler	bugfiler.8	file bug reports in folders automatically
catman	catman.8	create the cat files for the manual
chown	chown.8	change owner
clri	clri.8	clear i-node
comsat	comsat.8c	biff server
config	config.8	build system configuration files
crash	crash.8v	what happens when the system crashes
cron	cron.8	clock daemon
dcheck	dcheck.8	file system directory consistency check
diskpart	diskpart.8	calculate default disk partition sizes
dmesg	dmesg.8	collect system diagnostic messages to form error log
drtest	drtest.8	standalone disk test program
dump	dump.8	incremental file system dump
dumpfs	dumpfs.8	dump file system information
edquota	edquota.8	edit user quotas
fastboot	fastboot.8	reboot/halt the system without checking the disks
fasthalt	fastboot.8	reboot/halt the system without checking the disks
flcopy	arff.8v	archiver and copier for floppy
format	format.8v	how to format disk packs
fsck	fsck.8	file system consistency check and interactive repair
ftpd	ftpd.8c	DARPA Internet File Transfer Protocol server
gettable	gettable.8c	get NIC format host tables from a host
getty	getty.8	set terminal mode
halt	halt.8	stop the processor
htable	htable.8	convert NIC standard format host tables
icheck	icheck.8	file system storage consistency check
ifconfig	ifconfig.8c	configure network interface parameters
implog	implog.8c	IMP log interpreter
implogd	implogd.8c	IMP logger process
init	init.8	process control initialization
kgmon	kgmon.8	generate a dump of the operating systems profile buffers
lpc	lpc.8	line printer control program
lpd	lpd.8	line printer daemon

makedev	makedev.8	make system special files
makekey	makekey.8	generate encryption key
mkfs	mkfs.8	construct a file system
mklost+found	mklost+found.8	make a lost+found directory for fsck
mknod	mknod.8	build special file
mkproto	mkproto.8	construct a prototype file system
mount	mount.8	mount and dismount file system
ncheck	ncheck.8	generate names from i-numbers
newfs	newfs.8	construct a new file system
pac	pac.8	printer/ploter accounting information
pstat	pstat.8	print system facts
quot	quot.8	summarize file system ownership
quotacheck	quotacheck.8	file system quota consistency checker
quotaoff	quotaon.8	turn file system quotas on and off
quotaon	quotaon.8	turn file system quotas on and off
rc	rc.8	command script for auto-reboot and daemons
rdump	rdump.8c	file system dump across the network
reboot	reboot.8	UNIX bootstrapping procedures
renice	renice.8	alter priority of running processes
repquota	repquota.8	summarize quotas for a file system
restore	restore.8	incremental file system restore
rexecd	rexecd.8c	remote execution server
rlogind	rlogind.8c	remote login server
rmt	rmt.8c	remote magtape protocol module
route	route.8c	manually manipulate the routing tables
routed	routed.8c	network routing daemon
rrestore	rrestore.8c	restore a file system dump across the network
rshd	rshd.8c	remote shell server
rwhod	rwhod.8c	system status server
rxformat	rxformat.8v	format floppy disks
sa	sa.8	system accounting
savecore	savecore.8	save a core dump of the operating system
sendmail	sendmail.8	send mail over the internet
shutdown	shutdown.8	close down the system at a given time
sticky	sticky.8	executable files with persistent text
swapon	swapon.8	specify additional device for paging and swapping
sync	sync.8	update the super block
syslog	syslog.8	log systems messages
telnetd	telnetd.8c	DARPA TELNET protocol server
tftpd	tftpd.8c	DARPA Trivial File Transfer Protocol server
trpt	trpt.8c	transliterate protocol trace
tunefs	tunefs.8	tune up an existing file system
umount	mount.8	mount and dismount file system
update	update.8	periodically update the super block
uuclean	uuclean.8c	uucp spool directory clean-up
uusnap	uusnap.8c	show snapshot of the UUCP system
vipw	vipw.8	edit the password file

NAME

750rom – details of Vax-11/750 boot ROMs

SYNOPSIS

```
↑P
>>> B/1 loading xx(0,0)boot
Boot
: xx(0,0)750rom
```

then re-boot the CPU

DESCRIPTION

The Vax-11/750 has a four-position rotary switch on its front panel labeled "Boot Device". This switch is used to select one of four bootstrap loaders. Each loader is contained in a ROM on the memory controller board (board L0011 on earlier machines, L0016 on later ones.)

One way to discover what bootstrap ROMs are present is to use *750rom*, a stand-alone program (i.e., you must shutdown Unix before using it.) There are no options; just load it and it will report the ROM configuration.

CHECKING BY HAND

```
↑P
>>> E/P F20400      to check "A" ROM
>>> E/P F20500      to check "B" ROM
>>> E/P F20600      to check "C" ROM
>>> E/P F20700      to check "D" ROM
>>> C       to continue CPU, if desired
```

PROCEDURE

If you don't want to have to reboot Unix, or the *750rom* program is not present on your root device, you can still find out what ROMs are present.

To find out what bootstrap ROMs are loaded, follow the procedure described above to look at the contents of the ROMs (which appear in the physical address space.) The low-order eight bits (2 hexadecimal digits) should tell you the boot device type:

42	DEC Massbus (CMI) disks (RP06/RP07)
44	TU58 console cassette (Dectape-II)
4C	RL02 cartridge disk
4D	RK07 cartridge disk
53	Systems Industries CMI interface disks (Eagle)
55	DEC UDA50 disks (RA25/RA60/RA80/RA81)
FF	No ROM installed in this position

Normally, the machines seem to arrive with the TU58 ROM at position "A", the RK07 ROM at position "B", and the RL02 ROM at position "C". Position "D" is usually empty unless the system is shipped by DEC with a disk other than RK07 or RL02.

If you have foreign vendor disks (such as SI Eagles) installed, make sure the installer installs a ROM for you. You should mark the memory controller board so that if it is replaced by Field Service, they transfer the ROMs.

Other, more complicated ways to determine what ROMs are installed are to run the *ECKAM* diagnostic (memory diagnostic, supposedly on cassette #5), or, if you have the Remote Diagnostic Option installed, run the microdiagnostics *ECKAB* and *ECKAC* (supposedly on cassettes #1 and

#2). Either of these methods will display the ROM configuration at some point during the test. (For more details, see *Vax-11/750 Installation and Acceptance Manual* for *ECKAM*, or *KC750 Microdiagnostics and Technical Manual*, for *ECKAB/ECKAC*.)

BUGS

There may be other possible ROM types that are not listed here.

The names of the diagnostics are subject to change.

NAME

ac − login accounting

SYNOPSIS

/etc/ac [**−w** wtmp] [**−p**] [**−d**] [people] ...

DESCRIPTION

Ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced. **−w** is used to specify an alternate *wtmp* file. **−p** prints individual totals; without this option, only totals are printed. **−d** causes a printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, */usr/adm/wtmp* is used.

The accounting file */usr/adm/wtmp* is maintained by *init* and *login.* Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

FILES

/usr/adm/wtmp

SEE ALSO

init(8), sa(8), login(1), utmp(5).

NAME

adduser – procedure for adding new users

DESCRIPTION

A new user must choose a login name, which must not already appear in */etc/passwd.* An account can be added by editing a line into the passwd file; this must be done with the password file locked e.g. by using *vipw*(8).

A new user is given a group and user id. User id's should be distinct across a system, since they are used to control access to files. Typically, users working on similar projects will be put in the same group. Thus at UCB we have groups for system staff, faculty, graduate students, and a few special groups for large projects. System staff is group "10" for historical reasons, and the super-user is in this group.

A skeletal account for a new user "ernie" would look like:

```
ernie::235:20:& Kovacs,508E,7925,6428202:/mnt/grad/ernie:/bin/csh
```

The first field is the login name "ernie". The next field is the encrypted password which is not given and must be initialized using *passwd*(1). The next two fields are the user and group id's. Traditionally, users in group 20 are graduate students and have account names with numbers in the 200's. The next field gives information about ernie's real name, office and office phone and home phone. This information is used by the *finger*(1) program. From this information we can tell that ernie's real name is "Ernie Kovacs" (the & here serves to repeat "ernie" with appropriate capitalization), that his office is 508 Evans Hall, his extension is x2-7925, and this his home phone number is 642-8202. You can modify the *finger*(1) program if necessary to allow different information to be encoded in this field. The UCB version of finger knows several things particular to Berkeley – that phone extensions start "2–", that offices ending in "E" are in Evans Hall and that offices ending in "C" are in Cory Hall.

The final two fields give a login directory and a login shell name. Traditionally, user files live on a file system which has the machines single letter *net*(1) address as the first of two characters. Thus on the Berkeley CS Department VAX, whose Berknet address is "csvax" abbreviated "v" the user file systems are mounted on "/va", "/vb", etc. On each such filesystem there are subdirectories there for each group of users, i.e.: "/va/staff" and "/vb/prof". This is not strictly necessary but keeps the number of files in the top level directories reasonably small.

The login shell will default to "/bin/sh" if none is given. Most users at Berkeley choose "/bin/csh" so this is usually specified here.

It is useful to give new users some help in getting started, supplying them with a few skeletal files such as *.profile* if they use "/bin/sh", or *.cshrc* and *.login* if they use "/bin/csh". The directory "/usr/skel" contains skeletal definitions of such files. New users should be given copies of these files which, for instance, arrange to use *tset*(1) automatically at each login.

FILES

/etc/passwd	password file
/usr/skel	skeletal login directory

SEE ALSO

passwd(1), finger(1), chsh(1), chfn(1), passwd(5), vipw(8)

BUGS

User information should be stored in its own data base separate from the password file.

NAME

analyze − Virtual UNIX postmortem crash analyzer

SYNOPSIS

/etc/analyze [**−s** swapfile] [**−f**] [**−m**] [**−d**] [**−D**] [**−v**] corefile [system]

DESCRIPTION

Analyze is the post-mortem analyzer for the state of the paging system. In order to use *analyze* you must arrange to get a image of the memory (and possibly the paging area) of the system after it crashes (see *crash*(8V)).

The *analyze* program reads the relevant system data structures from the core image file and indexing information from **/vmunix** (or the specified file) to determine the state of the paging subsystem at the point of crash. It looks at each process in the system, and the resources each is using in an attempt to determine inconsistencies in the paging system state. Normally, the output consists of a sequence of lines showing each active process, its state (whether swapped in or not), its *p0br*, and the number and location of its page table pages. Any pages which are locked while raw i/o is in progress, or which are locked because they are *intransit* are also printed. (Intransit text pages often diagnose as duplicated; you will have to weed these out by hand.)

The program checks that any pages in core which are marked as not modified are, in fact, identical to the swap space copies. It also checks for non-overlap of the swap space, and that the core map entries correspond to the page tables. The state of the free list is also checked.

Options to *analyze*:

−D	causes the diskmap for each process to be printed.
−d	causes the (sorted) paging area usage to be printed.
−f	which causes the free list to be dumped.
−m	causes the entire coremap state to be dumped.
−v	(long unused) which causes a hugely verbose output format to be used.

In general, the output from this program can be confused by processes which were forking, swapping, or exiting or happened to be in unusual states when the crash occurred. You should examine the flags fields of relevant processes in the output of a *pstat*(8) to weed out such processes.

It is possible to look at the core dump with *adb* if you do

```
adb −k /vmunix /vmcore
```

FILES

/vmunix	default system namelist

SEE ALSO

adb(1), ps(1), crash(8V), pstat(8)

AUTHORS

Ozalp Babaoglu and William Joy

DIAGNOSTICS

Various diagnostics about overlaps in swap mappings, missing swap mappings, page table entries inconsistent with the core map, incore pages which are marked clean but differ from disk-image copies, pages which are locked or intransit, and inconsistencies in the free list.

It would be nice if this program analyzed the system in general, rather than just the paging system in particular.

NAME

arcv − convert archives to new format

SYNOPSIS

/etc/arcv file ...

DESCRIPTION

Arcv converts archive files (see *ar*(1), *ar*(5)) from 32v and Third Berkeley editions to a new portable format. The conversion is done in place, and the command refuses to alter a file not in old archive format.

Old archives are marked with a magic number of 0177545 at the start; new archives have a first line "!<arch>".

FILES

/tmp/v*, temporary copy

SEE ALSO

ar(1), ar(5)

NAME

arff, flcopy − archiver and copier for floppy

SYNOPSIS

/etc/arff [key] [name ...]
/etc/flcopy [**−h**] [**−t***n*]

DESCRIPTION

Arff saves and restores files on the console floppy disk. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file names specifying which files are to be dumped or restored.

Files names have restrictions, because of radix50 considerations. They must be in the form 1-6 alphanumerics followed by "." followed by 0-3 alphanumerics. Case distinctions are lost. Only the trailing component of a pathname is used.

The function portion of the key is specified by one of the following letters:

r The named files are replaced where found on the floppy, or added taking up the minimal possible portion of the first empty spot on the floppy.

x The named files are extracted from the floppy.

d The named files are deleted from the floppy. Arff will combine contiguous deleted files into one empty entry in the rt-11 directory.

t The names of the specified files are listed each time they occur on the floppy. If no file argument is given, all of the names on the floppy are listed.

The following characters may be used in addition to the letter which selects the function desired.

v The **v** (verbose) option, when used with the **t** function gives more information about the floppy entries than just the name.

f causes *arff* to use the next argument as the name of the archive instead of /dev/floppy.

m causes *arff* not to use the mapping algorithm employed in interleaving sectors around a floppy disk. In conjunction with the **f** option it may be used for extracting files from rt11 formatted cartridge disks, for example. It may also be used to speed up reading from and writing to rx02 floppy disks, by using the 'c' device instead of the 'b' device.

c causes *arff* to create a new directory on the floppy, effectively deleting all previously existing files.

Flcopy copies the console floppy disk (opened as '/dev/floppy') to a file created in the current directory, named "floppy", then prints the message "Change Floppy, hit return when done". Then *flcopy* copies the local file back out to the floppy disk.

The **−h** option to *flcopy* causes it to open a file named "floppy" in the current directory and copy it to */dev/floppy;* the **−t** option causes only the first *n* tracks to participate in a copy.

Arff may also be used with the console TU58 cassettes on the 11/730. To do so, the **m** key must be specified. Normally, the **f** key is also used.

FILES

/dev/floppy or /dev/rrx??
floppy (in current directory)

SEE ALSO

fl(4), rx(4), rxformat(8V)

AUTHORS

Keith Sklower, Richard Tuck

BUGS

Floppy errors are handled ungracefully; *Arff* does not handle multi-segment rt11 directories.

NAME

arp − address resolution display and control

SYNOPSIS

arp *hostname*
arp -a [*vmunix*] [*kmem*]
arp -d *hostname*
arp -s *hostname ether_addr* [**temp**] [**pub**]
arp -f *filename*

DESCRIPTION

The *arp* program displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol (*arp*(4p)).

With no flags, the program displays the current ARP entry for *hostname*. With the **-a** flag, the program displays all of the current ARP entries by reading the table from the file *kmem* (default /dev/kmem) based on the kernel file *vmunix* (default /vmunix).

With the **-d** flag, a super-user may delete an entry for the host called *hostname*.

The **-s** flag is given to create an ARP entry for the host called *hostname* with the Ethernet address *ether_addr*. The Ethernet address is given as six hex bytes separated by colons. The entry will be permanent unless the word **temp** is given in the command. If the word **pub** is given, the entry will be "published", e.g., this system will respond to ARP requests for *hostname* even though the hostname is not its own.

The **-f** flag causes the file *filename* to be read and multiple entries to be set in the ARP tables. Entries in the file should be of the form

hostname ether_addr [**temp**] [**pub**]

with argument meanings as given above.

SEE ALSO

arp(4p), ifconfig(8c)

NAME
bad144 − read/write dec standard 144 bad sector information

SYNOPSIS
/etc/bad144 [**−f**] disktype disk [sno [bad ...]]

DESCRIPTION
Bad144 can be used to inspect the information stored on a disk that is used by the disk drivers to implement bad sector forwarding. The format of the information is specified by DEC standard 144, as follows.

The bad sector information is located in the first 5 even numbered sectors of the last track of the disk pack. There are five identical copies of the information, described by the *dkbad* structure.

Replacement sectors are allocated starting with the first sector before the bad sector information and working backwards towards the beginning of the disk. A maximum of 126 bad sectors are supported. The position of the bad sector in the bad sector table determines which replacement sector it corresponds to.

The bad sector information and replacement sectors are conventionally only accessible through the "c" file system partition of the disk. If that partition is used for a file system, the user is responsible for making sure that it does not overlap the bad sector information or any replacement sectors.

The bad sector structure is as follows:

```
struct dkbad {
          long      bt_csn;                       /* cartridge serial number */
          u_short   bt_mbz;                       /* unused; should be 0 */
          u_short   bt_flag;                      /* -1 => alignment cartridge */
          struct bt_bad {
                    u_short bt_cyl;               /* cylinder number of bad sector */
                    u_short bt_trksec;            /* track and sector number */
          } bt_bad[126];
};
```

Unused slots in the *bt_bad* array are filled with all bits set, a putatively illegal value.

Bad144 is invoked by giving a device type (e.g. rk07, rm03, rm05, etc.), and a device name (e.g. hk0, hp1, etc.). It reads the first sector of the last track of the corresponding disk and prints out the bad sector information. It may also be invoked giving a serial number for the pack and a list of bad sectors, and will then write the supplied information onto the same location. Note, however, that *bad144* does not arrange for the specified sectors to be marked bad in this case. This option should only be used to restore known bad sector information which was destroyed.

If the disk is an RP06, Fujitsu Eagle, or Ampex Capricorn on a Massbus, the **−f** option may be used to mark the bad sectors as "bad". **NOTE: this can only be done safely when there is no other disk activity,** preferably while running single-user. Otherwise, new bad sectors can be added only by running a formatter. Note that the order in which the sectors are listed determines which sectors used for replacements; if new sectors are being added to the list on a drive that is in use, care should be taken that replacements for existing bad sectors have the correct contents.

SEE ALSO
badsect(8), format(8V)

BUGS

It should be possible to format disks on-line under UNIX.

It should be possible to mark bad sectors on drives of all type.

On an 11/750, the standard bootstrap drivers used to boot the system do not understand bad sectors, handle ECC errors, or the special SSE (skip sector) errors of RM80 type disks. This means that none of these errors can occur when reading the file /vmunix to boot. Sectors 0-15 of the disk drive must also not have any of these errors.

The drivers which write a system core image on disk after a crash do not handle errors; thus the crash dump area must be free of errors and bad sectors.

NAME

badsect − create files to contain bad sectors

SYNOPSIS

/etc/badsect bbdir sector ...

DESCRIPTION

Badsect makes a file to contain a bad sector. Normally, bad sectors are made inaccessible by the standard formatter, which provides a forwarding table for bad sectors to the driver; see *bad144*(8) for details. If a driver supports the bad blocking standard it is much preferable to use that method to isolate bad blocks, since the bad block forwarding makes the pack appear perfect, and such packs can then be copied with *dd*(1). The technique used by this program is also less general than bad block forwarding, as *badsect* can't make amends for bad blocks in the i-list of file systems or in swap areas.

On some disks, adding a sector which is suddenly bad to the bad sector table currently requires the running of the standard DEC formatter. Thus to deal with a newly bad block or on disks where the drivers do not support the bad-blocking standard *badsect* may be used to good effect.

Badsect is used on a quiet file system in the following way: First mount the file system, and change to its root directory. Make a directory BAD there. Run *badsect* giving as argument the BAD directory followed by all the bad sectors you wish to add. (The sector numbers must be relative to the beginning of the file system, but this is not hard as the system reports relative sector numbers in its console error messages.) Then change back to the root directory, unmount the file system and run *fsck*(8) on the file system. The bad sectors should show up in two files or in the bad sector files and the free list. Have *fsck* remove files containing the offending bad sectors, but **do not** have it remove the BAD/*nnnnn* files. This will leave the bad sectors in only the BAD files.

Badsect works by giving the specified sector numbers in a *mknod*(2) system call, creating an illegal file whose first block address is the block containing bad sector and whose name is the bad sector number. When it is discovered by *fsck* it will ask "HOLD BAD BLOCK"? A positive response will cause *fsck* to convert the inode to a regular file containing the bad block.

SEE ALSO

bad144(8), fsck(8), format(8V)

DIAGNOSTICS

Badsect refuses to attach a block that resides in a critical area or is out of range of the file system. A warning is issued if the block is already in use.

BUGS

If more than one sector which comprise a file system fragment are bad, you should specify only one of them to *badsect,* as the blocks in the bad sector files actually cover all the sectors in a file system fragment.

NAME

breathlife – breath-of-life server for bootloading 3mb Altos

SYNOPSIS

/etc/pup/breathlife

DESCRIPTION

This is a simple server that blindly sends out an Alto "breath of life" packet on the 3mb ethernet every few seconds. Use it only if you have an Alto on the same network as your system.

FILES

/etc/pup/pupnettab	for list of networks

AUTHOR

Jeffrey Mogul

NAME

bugfiler — file bug reports in folders automatically

SYNOPSIS

bugfiler [mail directory]

DESCRIPTION

Bugfiler is a program to automatically intercept bug reports, summarize them and store them in the appropriate sub directories of the mail directory specified on the command line or the (system dependent) default. It is designed to be compatible with the Rand MH mail system. *Bugfiler* is normally invoked by the mail delivery program through *aliases*(5) with a line such as the following in /usr/lib/aliases.

```
bugs:"|bugfiler /usr/bugs/mail"
```

It reads the message from the standard input or the named file, checks the format and returns mail acknowledging receipt or a message indicating the proper format. Valid reports are then summarized and filed in the appropriate folder. Users can then log onto the system and check the summary file for bugs that pertain to them. Bug reports are submitted in RFC822 format and must contain the following header lines:

```
Date: <date the report is received>
From: <valid return address>
Subject: <short summary of the problem>
Index: <source directory>/<source file> <version> [Fix]
```

In addition, the body of the message must contain a line which begins with "Description:" followed by zero or more lines describing the problem in detail and a line beginning with "Repeat-By:" followed by zero or more lines describing how to repeat the problem. If the keyword 'Fix' is specified in the 'Index' line, then there must also be a line beginning with "Fix:" followed by a diff of the old and new source files or a description of what was done to fix the problem.

The 'Index' line is the key to the filing mechanism. The source directory name must match one of the folder names in the mail directory. The message is then filed in this folder and a line appended to the summary file in the following format:

```
<folder name>/<message number>  <Index info>
                                <Subject info>
```

FILES

/usr/new/lib/mh/deliver	mail delivery program
/usr/new/lib/mh/unixtomh	converts unix mail format to mh format
maildir/.ack	the message sent in acknowledgement
maildir/.format	the message sent when format errors are detected
maildir/summary	the summary file
maildir/Bf??????	temporary copy of the input message
maildir/Rp??????	temporary file for the reply message.

SEE ALSO

mh(1), newaliases(1), aliases(5)

BUGS

Since mail can be forwarded in a number of different ways, *bugfiler* does not recognize forwarded mail and will reply/complain to the forwarder instead of the original sender unless there is a 'Reply-To' field in the header.

Duplicate messages should be discarded or recognized and put somewhere else.

NAME

buildnetdir – build binary-format Pup Network Directory

SYNOPSIS

buildnetdir [–p] [–d] [–s] [–] [infile [outfile]]

DESCRIPTION

Buildnetdir parses the human-readable version of Pup Network directory and produces a binary-formatted version. The default input file is /etc/pup/Pup-Network.txt; the default output file is /etc/pup/Pup-Network.Dir; you cannot specify the output file unless you also specify the input file.

Options are:

-p	Parse Only – the input file is parsed, and a new version number is found, but the output file is not changed.
-s	Storage statistics are printed.
-d	Debugging information is printed.
-	The standard input is read for input.

PROPER WAY OF USING THIS PROGRAM

In order to reduce confusion, we (at Stanford) have established a procedure for updating the directory.

The "official" copy of the human-readable directory lives on [Navajo]<System>Pup-Network.txt. Retrieve this file to your local /etc/pup/Pup-Network.txt, edit it, and run *buildnetdir*; you might first want to use it with the –p flag, to make sure that the file is syntactically correct. If it is ok, run *buildnetdir* withouth the –p flag, which will "install" the new binary directory. The miscserver on the local machine will notice this within a minute or so, and propagate the new directory to all other machines.

Meanwhile, you should store the new human-readable directory back at Navajo and at the IFS; before doing this, you should probably check to see if anyone else has modified it in parallel with you. (If this has happened, the two of you will have to agree on a correct version, then proceed again from the beginning.)

If any old-fashioned miscservers (those that do not use the network directory update protocol) still exist on your network, please copy the new human-readable directory to those machines.

FILES

/etc/pup/Pup-Network.txt	default input file
/etc/pup/Pup-Network.Dir	default output file

SEE ALSO

miscserver(8)

DIAGNOSTICS

More or less self-explanatory. If the program cannot determine a new version number (i.e., no server on the net provide the current version number), you will be asked for one.

AUTHOR

Jeffrey Mogul

BUGS

Doesn't understand "attributes" attached to addresses (understands attributes attached to entries, though.)

NAME

catman − create the cat files for the manual

SYNOPSIS

/etc/catman [−p] [−n] [−w] [sections]

DESCRIPTION

Catman creates the preformatted versions of the on-line manual from the nroff input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, *catman* will recreate the **/usr/lib/whatis** database.

If there is one parameter not starting with a '−', it is take to be a list of manual sections to look in. For example

catman 123

will cause the updating to only happen to manual sections 1, 2, and 3.

Options:

−n	prevents creations of **/usr/lib/whatis.**
−p	prints what would be done instead of doing it.
−w	causes only the **/usr/lib/whatis** database to be created. No manual reformatting is done.

FILES

/usr/man/man?/*.*	raw (nroff input) manual sections
/usr/man/cat?/*.*	preformatted manual pages
/usr/lib/makewhatis	commands to make whatis database

SEE ALSO

man(1)

BUGS

Acts oddly on nights with full moons.

NAME

chown – change owner

SYNOPSIS

/etc/chown [**–f**] owner file ...

DESCRIPTION

Chown changes the owner of the *files* to *owner*. The owner may be either a decimal UID or a login name found in the password file.

Only the super-user can change owner, in order to simplify accounting procedures. No errors are reported when the **–f** (force) option is given.

FILES

/etc/passwd

SEE ALSO

chgrp(1), chown(2), passwd(5), group(5)

NAME

clri − clear i-node

SYNOPSIS

/etc/clri filesystem i-number ...

DESCRIPTION

N.B.: *Clri* is obsoleted for normal file system repair work by *fsck*(8).

Clri writes zeros on the i-nodes with the decimal *i-numbers* on the *filesystem.* After *clri,* any blocks in the affected file will show up as 'missing' in an *icheck*(8) of the *filesystem.*

Read and write permission is required on the specified file system device. The i-node becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to zap an i-node which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated i-node, so the whole cycle is likely to be repeated again and again.

SEE ALSO

icheck(8)

BUGS

If the file is open, *clri* is likely to be ineffective.

NAME

comsat − biff server

SYNOPSIS

/etc/comsat

DESCRIPTION

Comsat is the server process which listens for reports of incoming mail and notifies users if they have requested this service. *Comsat* listens on a datagram port associated with the "biff" service specification (see *services*(5)) for one line messages of the form

    user@mailbox-offset

If the *user* specified is logged in to the system and the associated terminal has the owner execute bit turned on (by a "biff y"), the *offset* is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the "From", "To", "Date", or "Subject" lines are not included in the displayed message.

FILES

/etc/utmp	to find out who's logged on and on what terminals

SEE ALSO

biff(1)

BUGS

The message header filtering is prone to error.

Users should be notified of mail which arrives on other machines than the one they are currently logged in to.

The notification should appear in a separate window so it does not mess up the screen.

NAME

config − build system configuration files

SYNOPSIS

/etc/config [−p] *config_file*

DESCRIPTION

Config builds a set of system configuration files from a short file which describes the sort of system that is being configured. It also takes as input a file which tells *config* what files are needed to generate a system. This can be augmented by a configuration specific set of files that give alternate files for a specific machine. (see the FILES section below) If the **−p** option is supplied, *config* will configure a system for profiling; c.f. *kgmon*(8), *gprof*(1).

Config should be run from the **conf** subdirectory of the system source (usually /sys/conf). *Config* assumes that there is already a directory *../config_file* created and it places all its output files in there. The output of *config* consists of a number files: **ioconf.c** contains a description of what i/o devices are attached to the system,; **ubglue.s** contains a set of interrupt service routines for devices attached to the UNIBUS; **makefile** is a file used by *make*(1) in building the system; a set of header files which contain the number of various devices that will be compiled into the system; and a set of swap configuration files which contain definitions for the disk areas to be used for swapping, the root file system, argument processing, and system dumps.

After running *config*, it is necessary to run "make depend" in the directory where the new makefile was created. *Config* reminds you of this when it completes.

If you get any other error messages from *config*, you should fix the problems in your configuration file and try again. If you try to compile a system that had configuration errors, you will likely meet with failure.

FILES

/sys/conf/makefile.vax	generic makefile for the VAX
/sys/conf/files	list of common files system is built from
/sys/conf/files.vax	list of VAX specific files
/sys/conf/devices.vax	name to major device mapping file for the VAX
/sys/conf/files.ERNIE	list of files specific to ERNIE system

SEE ALSO

"Building 4.2BSD UNIX System with Config"
The SYNOPSIS portion of each device in section 4.

BUGS

The line numbers reported in error messages are usually off by one.

NAME

crash − what happens when the system crashes

DESCRIPTION

This section explains what happens when the system crashes and how you can analyze crash dumps.

When the system crashes voluntarily it prints a message of the form

> panic: why i gave up the ghost

on the console, takes a dump on a mass storage peripheral, and then invokes an automatic reboot procedure as described in *reboot*(8). (If auto-reboot is disabled on the front panel of the machine the system will simply halt at this point.) Unless some unexpected inconsistency is encountered in the state of the file systems due to hardware or software failure the system will then resume multi-user operations.

The system has a large number of internal consistency checks; if one of these fails, then it will panic with a very short message indicating which one failed.

The most common cause of system failures is hardware failure, which can reflect itself in different ways. Here are the messages which you are likely to encounter, with some hints as to causes. Left unstated in all cases is the possibility that hardware or software error produced the message in some unexpected way.

IO err in push
hard IO err in swap

The system encountered an error trying to write to the paging device or an error in reading critical information from a disk drive. You should fix your disk if it is broken or unreliable.

timeout table overflow

This really shouldn't be a panic, but until we fix up the data structure involved, running out of entries causes a crash. If this happens, you should make the timeout table bigger.

KSP not valid
SBI fault
CHM? in kernel

These indicate either a serious bug in the system or, more often, a glitch or failing hardware. If SBI faults recur, check out the hardware or call field service. If the other faults recur, there is likely a bug somewhere in the system, although these can be caused by a flakey processor. Run processor microdiagnostics.

machine check %x:
description

machine dependent machine-check information

We should describe machine checks, and will someday. For now, ask someone who knows (like your friendly field service people).

trap type %d, code=%d, pc=%x

A unexpected trap has occurred within the system; the trap types are:

0	reserved addressing fault
1	privileged instruction fault
2	reserved operand fault
3	bpt instruction fault
4	xfc instruction fault
5	system call trap

6	arithmetic trap
7	ast delivery trap
8	segmentation fault
9	protection fault
10	trace trap
11	compatibility mode fault
12	page fault
13	page table fault

The favorite trap types in system crashes are trap types 8 and 9, indicating a wild reference. The code is the referenced address, and the pc at the time of the fault is printed. These problems tend to be easy to track down if they are kernel bugs since the processor stops cold, but random flakiness seems to cause this sometimes.

init died

The system initialization process has exited. This is bad news, as no new users will then be able to log in. Rebooting is the only fix, so the system just does it right away.

That completes the list of panic types you are likely to see.

When the system crashes it writes (or at least attempts to write) an image of memory into the back end of the primary swap area. After the system is rebooted, the program *savecore*(8) runs and preserves a copy of this core image and the current system in a specified directory for later perusal. See *savecore*(8) for details.

To analyze a dump you should begin by running *adb*(1) with the **−k** flag on the core dump. Normally the command "*(intstack-4)$c" will provide a stack trace from the point of the crash and this will provide a clue as to what went wrong. A more complete discussion of system debugging is impossible here. See, however, "Using ADB to Debug the UNIX Kernel".

SEE ALSO

adb(1), analyze(8), reboot(8)
VAX 11/780 System Maintenance Guide for more information about machine checks.
Using ADB to Debug the UNIX Kernel

NAME

cron — clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times according to the instructions in the file /usr/lib/crontab. Since *cron* never exits, it should only be executed once. This is best done by running *cron* from the initialization process through the file /etc/rc; see *init*(8).

Crontab consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns to specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (1-7 with 1=Monday). Each of these patterns may contain a number in the range above; two numbers separated by a minus meaning a range inclusive; a list of numbers separated by commas meaning any of the numbers; or an asterisk meaning all legal values. The sixth field is a string that is executed by the Shell at the specified times. A percent character in this field is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the Shell. The other lines are made available to the command as standard input.

Crontab is examined by *cron* every minute.

FILES

/usr/lib/crontab

NAME

dcheck – file system directory consistency check

SYNOPSIS

/etc/dcheck [**−i** numbers] [filesystem]

DESCRIPTION

N.B.: *Dcheck* is obsoleted for normal consistency checking by *fsck*(8).

Dcheck reads the directories in a file system and compares the link-count in each i-node with the number of directory entries by which it is referenced. If the file system is not specified, a set of default file systems is checked.

The **−i** flag is followed by a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

The program is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), icheck(8), fs(5), clri(8), ncheck(8)

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

BUGS

Since *dcheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

Dcheck is obsoleted by *fsck* and remains for historical reasons.

NAME
ddacct – Dump Dover Accounting

SYNOPSIS
ddacct [-cpuP]

DESCRIPTION
Dumps dover accounting information which consists of the number of pages and files printed for each user and the percentage of total dover usage represented by that user (only for the current machine, of course).

Normally the listing is sorted by pages printed, the -P switch causes it to be sorted by user id and -u causes it to be sorted by user login name.

The -c switch causes the accounting information to be cleaned out after the listing is made. It resets all counters to zero.

FILES
/usr/adm/dover-accnt   accounting information.

SEE ALSO
cz (1)

DIAGNOSTICS
"There is no accounting file to dump": a system administrator can enable dover accounting by creating an empty accouting file.

BUGS
Only accounts for stuff sent to the Dover with czarina (*cz*(1)) and *dtroff*(1). The accounting information should really be kept by *dpr*(1), since all files printed go through that program.

The accounting database cannot separately account for multiple printers.

HISTORY
06-Mar-81 James Gosling (jag) at Carnegie-Mellon University
Created.

NAME

diskpart − calculate default disk partition sizes

SYNOPSIS

/etc/diskpart [**−p**] [**−d**] disk-type

DESCRIPTION

Diskpart is used to calculate the disk partition sizes based on the default rules used at Berkeley. If the **−p** option is supplied, tables suitable for inclusion in a device driver are produced. If the **−d** option is supplied, an entry suitable for inclusion in the disk description file */etc/disktab* is generated; c.f. *disktab*(5). Space is always left in the last partition on the disk for a bad sector forwarding table. The space reserved is one track for the replicated copies of the table and sufficient tracks to hold a pool of 126 sectors to which bad sectors are mapped. For more information, see *bad144*(8).

The disk partition sizes are based on the total amount of space on the disk as give in the table below (all values are supplied in units of 512 byte sectors). The 'c' partition is, by convention, used to access the entire physical disk, including the space reserved for the bad sector forwarding table. In normal operation, either the 'g' partition is used, or the 'd', 'e', and 'f' partitions are used. The 'g' and 'f' partitions are variable sized, occupying whatever space remains after allocation of the fixed sized partitions. If the disk is smaller than 20 Megabytes, then *diskpart* aborts with the message "disk too small, calculate by hand".

Partition	20-60 MB	61-205 MB	206-355 MB	356+ MB
a	15884	15884	15884	15884
b	10032	33440	33440	66880
d	15884	15884	15884	15884
e	unused	55936	55936	307200
h	unused	unused	291346	291346

If an unknown disk type is specified, *diskpart* will prompt for the required disk geometry information.

SEE ALSO

disktab(5), bad144(8)

BUGS

Certain default partition sizes are based on historical artifacts (e.g. RP06), and may result in unsatisfactory layouts.

When using the −d flag, alternate disk names are not included in the output.

Does not understand how to handle drives attached to a UDA50.

NAME

dmesg — collect system diagnostic messages to form error log

SYNOPSIS

/etc/dmesg [—]

DESCRIPTION

Dmesg looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed by the system when device (hardware) errors occur and (occasionally) when system tables overflow non-fatally. If the — flag is given, then *dmesg* computes (incrementally) the new messages since the last time it was run and places these on the standard output. This is typically used with *cron*(8) to produce the error log */usr/adm/messages* by running the command

```
/etc/dmesg — >> /usr/adm/messages
```

every 10 minutes.

FILES

/usr/adm/messages	error log (conventional location)
/usr/adm/msgbuf	scratch file for memory of — option

BUGS

The system error message buffer is of small finite size. As *dmesg* is run only every few minutes, not all error messages are guaranteed to be logged. This can be construed as a blessing rather than a curse.

Error diagnostics generated immediately before a system crash will never get logged.

NAME

drtest − standalone disk test program

DESCRIPTION

Drtest is a standalone program used to read a disk track by track. It was primarily intended as a test program for new standalone drivers, but has shown useful in other contexts as well, such as verifying disks and running speed tests. For example, when a disk has been formatted (by format(8)), you can check that hard errors has been taken care of by running *drtest.* No hard errors should be found, but in many cases quite a few soft ECC errors will be reported.

While *drtest* is running, the cylinder number is printed on the console for every 10th cylinder read.

EXAMPLE

A sample run of *drtest* is shown below. In this example (using a 750), *drtest* is loaded from the root file system; usually it will be loaded from the machine's console storage device. Boldface means user input. As usual, "#" and "@" may be used to edit input.

```
>>>B/3
%%
loading hk(0,0)boot
Boot
: hk(0,0)drtest
Test program for stand-alone up and hp driver

Debugging level (1=bse, 2=ecc, 3=bse+ecc)?
Enter disk name [type(adapter,unit), e.g. hp(1,3)]? hp(0,0)
Device data: #cylinders=1024, #tracks=16, #sectors=32
Testing hp(0,0), chunk size is 16384 bytes.
(chunk size is the number of bytes read per disk access)
Start ...Make sure hp(0,0) is online
 ...
(errors are reported as they occur)
 ...
(...program restarts to allow checking other disks)
(...to abort halt machine with ^P)
```

DIAGNOSTICS

The diagnostics are intended to be self explanatory. Note, however, that the device number in the diagnostic messages is identified as *typeX* instead of *type(a,u)* where *X* = a*8+u, e.g., hp(1,3) becomes hp11.

SEE ALSO

format(8), bad144(8)

AUTHOR

Helge Skrivervik

NAME

dump − incremental file system dump

SYNOPSIS

/etc/dump [key [*argument* ...] filesystem]

DESCRIPTION

Dump copies to magnetic tape all files changed after a certain date in the *filesystem.* The *key* specifies the date and other options about the dump. *Key* consists of characters from the set **0123456789fusdWn.**

0−9 This number is the 'dump level'. All files modified since the last date stored in the file */etc/dumpdates* for the same filesystem at lesser levels will be dumped. If no date is determined by the level, the beginning of time is assumed; thus the option **0** causes the entire filesystem to be dumped.

f Place the dump on the next *argument* file instead of the tape. If the name of the file is "−", *dump* writes to standard output.

u If the dump completes successfully, write the date of the beginning of the dump on file */etc/dumpdates.* This file records a separate date for each filesystem and each dump level. The format of */etc/dumpdates* is readable by people, consisting of one free format record per line: filesystem name, increment level and *ctime(3)* format dump date. */etc/dumpdates* may be edited to change any of the fields, if necessary.

s The size of the dump tape is specified in feet. The number of feet is taken from the next *argument.* When the specified size is reached, *dump* will wait for reels to be changed. The default tape size is 2300 feet.

d The density of the tape, expressed in BPI, is taken from the next *argument.* This is used in calculating the amount of tape used per reel. The default is 1600.

W *Dump* tells the operator what file systems need to be dumped. This information is gleaned from the files */etc/dumpdates* and */etc/fstab.* The **W** option causes *dump* to print out, for each file system in */etc/dumpdates* the most recent dump date and level, and highlights those file systems that should be dumped. If the **W** option is set, all other options are ignored, and *dump* exits immediately.

w Is like W, but prints only those filesystems which need to be dumped.

n Whenever *dump* requires operator attention, notify by means similar to a *wall*(1) all of the operators in the group "operator".

If no arguments are given, the *key* is assumed to be **9u** and a default file system is dumped to the default tape.

Dump requires operator intervention on these conditions: end of tape, end of dump, tape write error, tape open error or disk read error (if there are more than a threshold of 32). In addition to alerting all operators implied by the **n** key, *dump* interacts with the operator on *dump's* control terminal at times when *dump* can no longer proceed, or if something is grossly wrong. All questions *dump* poses **must** be answered by typing "yes" or "no", appropriately.

Since making a dump involves a lot of time and effort for full dumps, *dump* checkpoints itself at the start of each tape volume. If writing that volume fails for some reason, *dump* will, with operator permission, restart itself from the checkpoint after the old tape has been rewound and removed, and a new tape has been mounted.

Dump tells the operator what is going on at periodic intervals, including usually low estimates of the number of blocks to write, the number of tapes it will take, the time to completion, and the time to the tape change. The output is verbose, so that others know that the terminal controlling *dump* is busy, and will be for some time.

Now a short suggestion on how to perform dumps. Start with a full level 0 dump

```
dump 0un
```

Next, dumps of active file systems are taken on a daily basis, using a modified Tower of Hanoi algorithm, with this sequence of dump levels:

3 2 5 4 7 6 9 8 9 9 ...

For the daily dumps, a set of 10 tapes per dumped file system is used on a cyclical basis. Each week, a level 1 dump is taken, and the daily Hanoi sequence repeats with 3. For weekly dumps, a set of 5 tapes per dumped file system is used, also on a cyclical basis. Each month, a level 0 dump is taken on a set of fresh tapes that is saved forever.

FILES

/dev/rrp1g	default filesystem to dump from
/dev/rmt8	default tape unit to dump to
/etc/ddate	old format dump date record (obsolete after −J option)
/etc/dumpdates	new format dump date record
/etc/fstab	dump table: file systems and frequency
/etc/group	to find group *operator*

SEE ALSO

restore(8), dump(5), fstab(5)

DIAGNOSTICS

Many, and verbose.

BUGS

Sizes are based on 1600 BPI blocked tape; the raw magtape device has to be used to approach these densities. Fewer than 32 read errors on the filesystem are ignored. Each reel requires a new process, so parent processes for reels already written just hang around until the entire tape is written.

It would be nice if *dump* knew about the dump sequence, kept track of the tapes scribbled on, told the operator which tape to mount when, and provided more assistance for the operator running *restore*.

NAME

dumpfs — dump file system information

SYNOPSIS

dumpfs *filesys*|*device*

DESCRIPTION

Dumpfs prints out the super block and cylinder group information for the file system or special device specified. The listing is very long and detailed. This command is useful mostly for finding out certain file system information such as the file system block size and minimum free space percentage.

SEE ALSO

fs(5), disktab(5), tunefs(8), newfs(8), fsck(8)

NAME

edquota − edit user quotas

SYNOPSIS

edquota [**−p** *proto-user*] *users...*

DESCRIPTION

Edquota is a quota editor. One or more users may be specified on the command line. For each user a temporary file is created with an ASCII representation of the current disc quotas for that user and an editor is then invoked on the file. The quotas may then be modified, new quotas added, etc. Upon leaving the editor, *edquota* reads the temporary file and modifies the binary quota files to reflect the changes made.

If the **−p** option is specified, *edquota* will duplicate the quotas of the prototypical user specified for each user specified. This is the normal mechanism used to initialize quotas for groups of users.

The editor invoked is *vi*(1) unless the environment variable EDITOR specifies otherwise.

Only the super-user may edit quotas.

FILES

quotas	at the root of each file system with quotas
/etc/fstab	to find file system names and locations

SEE ALSO

quota(1), quota(2), quotacheck(8), quotaon(8), repquota(8)

DIAGNOSTICS

Various messages about inaccessible files; self-explanatory.

BUGS

The format of the temporary file is inscrutable.

NAME

enstat – print enet (packet filter) information

SYNOPSIS

/etc/enstat [–cdfkpqsv01234567] [system-image [corefile]]

DESCRIPTION

Enstat interprets the data structures of the Ethernet packet filter driver *enet*(4). If *system-image* is given, the required namelist is taken from there; otherwise, it is taken from */vmunix.* If *corefile* is given, the data structures are sought there, otherwise in */dev/kmem.* (If *corefile* is a core dump, then the –k option must be given.)

OPTIONS

c	(Counts): give various counts (per ethernet unit) including number of packets sent and received.
d	(Descriptors): show OpenDescriptors for each minor device.
f	(Filters): show packet filters for each minor device.
k	The corefile is a crash dump, not a running system's /dev/kmem.
p	(Parameters): give device parameters including device type, header and address lengths, MTU, and interface and broadcast addresses.
q	(QueueElements): show the QueueElements.
s	(Scavenger): show the FreeQueue and Scavenger statistics.
v	(Verbose): show information for minor devices not actually in use, and complete queue information, only if this flag is given.
<digit>	Limit output to information about specified units; if no digits are given, then all units are displayed.

If no options are given, then all are assumed (except for **–v** [Verbose]).

OUTPUT FORMAT

"AllDescriptors"		
	Minor device number for open descriptor; followed by "K" if opened by kernel.	
LOC	Descriptor location	
USED	"yes" or "no"	
LINK-QUEUE	Forward and Backward links to other descriptors; three leading digits suppressed	
STATE	Blank, or one of:	
	wait	waiting for input, indefinite wait
	timed	waiting for input, timed wait
	tout	has timed out
	anything else	shouldn't happen
WAIT-QUEUE	Forward and Backward links to waiting packets	
NQ'D	number of packets queued for input (maximum for this queue shown in parentheses)	
TOUT	timeout duration in clock ticks (if the **–v** [Verbose] option is not given, then times may be expressed as minutes [with a trailing "m"], hours [with a trailing "h"], or simply "long", to keep the columns lined up.)	
SIGN	signal number to be delivered when a packet arrives	
PROC	process to be signaled when a packet arrives	
(PID)	process id which enabled the signal	

"Filters"	
LOC	location of descriptor
PRI	priority of filter
LEN	length of filter (in shortwords)
FILTER	see *enet*(4) for interpretation of Ethernet packet filters
"QueueElts"	
LOC	Location of queue element
LINK-QUEUE	Forward and backward links
FUNC	Address of completion function (used by kernel-mode access)
COUNT	packet size
REF	Reference count for queue element.

FILES

/vmunix	namelist
/dev/kmem	default source of tables

SEE ALSO

etherport(1), netstat(1), enet(4), pstat(8)
K. Thompson, *UNIX Implementation*

AUTHOR

Jeffrey Mogul at Stanford, after work done by Mike Accetta at CMU, based on *pstat*(8).

BUGS

Some of the output is a bit cramped so as to fit on an 80-character line. It should be possible to get a less verbose but more readable listing.

Since things happen pretty fast, it's not likely that *enstat* will provide a consistent view of a running system. It is mostly useful for analyzing static problems, not transient ones.

NAME

expire – remove outdated news articles

SYNOPSIS

/usr/lib/news/expire [**–n** *newsgroups*] [**–i**] [**–I**] [**–v** [*level*]] [**–e***days*] [**–a**]

DESCRIPTION

Expire is normally started up by *cron*(8) every night to remove all expired news. If no newsgroups are specified, the default is to expire **all**.

Articles whose specified expiration date has already passed are considered expirable. The **–a** option causes expire to archive articles in /usr/spool/oldnews. Otherwise, the articles are unlinked.

The **–v** option causes expire to be more verbose. It can be given a verbosity level (default 1) as in **–v3** for even more output. This is useful if articles aren't being expired and you want to know why.

The **–e** flag gives the number of days to use for a default expiration date. If not given, an installation dependent default (often 2 weeks) is used.

The **–i** and **–I** flags tell **expire** to ignore any expiration date explicitly given on articles. This can be used when disk space is really tight. The **–I** flag will always ignore expiration dates, while the **–i** flag will only ignore the date if ignoring it would expire the article sooner. *WARNING:* If you have articles archived by giving them expiration dates far into the future, these options might remove these files anyway.

SEE ALSO

checknews(1), inews(1), readnews(1), recnews(8), sendnews(8), uurec(8)

NAME

fastboot, fasthalt — reboot/halt the system without checking the disks

SYNOPSIS

/etc/fastboot [*boot-options*]
/etc/fasthalt [*halt-options*]

DESCRIPTION

Fastboot and *fasthalt* are shell scripts which reboot and halt the system without checking the file systems. This is done by creating a file */fastboot*, then invoking the *reboot* program. The system startup script, */etc/rc*, looks for this file and, if present, skips the normal invocation of *fsck*(8).

SEE ALSO

halt(8), reboot(8), rc(8)

NAME

filetime – tell minutes since file (access, modification) time

SYNOPSIS

filetime [-c | -m | -a] filename

DESCRIPTION

On the standard output, print the number of minutes since the file specified in the argument was { created, modified, accessed }. An optional argument -X where "X" is one of { c, m, a } selects which file time to use; the default is the modification time.

This program was written for .login or .profile files as part of the login startup script; here is an example for /bin/csh; the version for /bin/sh would be quite similar:

```
if (`filetime .last_done` > 20 ) then
        << commands that should be executed periodically>>
        << .... >>
endif
touch .last_done
```

The purpose is to avoid checking for news, mail on other systems, etc, if you just logged in 5 minutes ago and did it before. Thus the user can adjust the granularity of performing these tasks and thus speed up her/his login.

AUTHOR

Steve Hartwell. Manual page by Brian Reid.

NAME

fingd – network finger server

SYNOPSIS

/etc/fingd

DESCRIPTION

Fingd listens on TCP port 79 (decimal) for connections. When a connection is made, it reads the string (assumed to be a user name or mail alias) and executes *fing*(1) with that argument, sending the output back on the connection.

Connection history is logged on standard error.

AUTHOR

Christopher A. Kent

SEE ALSO

fing(l), finger(1)

NOTES

Since the name being sent to the server may be a forwarded request (due to a user having his mail forwarded to one central machine via a *delivermail*(8) alias), the *fing* command will be invoked with the −m flag to force matches on login names only. Unfortunately, this doesn't allow people to be *finger*ed by last name unless there is a *delivermail* alias to handle it. The alternative can cause excessive return information if a user has a gecos string that textually overlaps someone's login id.

NAME

format − how to format disk packs

DESCRIPTION

There are two ways to format disk packs. The simplest is to use the *format* program. The alternative is to use the DEC standard formatting software which operates under the DEC diagnostic supervisor. This manual page describes the operation of *format*, then concludes with some remarks about using the DEC formatter.

Format is a standalone program used to format and check disks prior to constructing file systems. In addition to the formatting operation, *format* records any bad sectors encountered according to DEC standard 144. Formatting is performed one track at a time by writing the appropriate headers and a test pattern and then checking the sector by reading and verifying the pattern, using the controller's ECC for error detection. A sector is marked bad if an unrecoverable media error is detected, or if a correctable ECC error greater than 5 bits in length is detected (such errors are indicated as "ECC" in the summary printed upon completing the format operation). After the entire disk has been formatted and checked, the total number of errors are reported, any bad sectors and skip sectors are marked, and a bad sector forwarding table is written to the disk in the first five even numbered sectors of the last track. *Format* may be used on any UNIBUS or MASSBUS drive supported by the *up* and *hp* device drivers which uses 4-byte headers (everything except RP's).

The test pattern used during the media check may be selected from one of: 0xf00f (RH750 worst case), 0xec6d (media worst case), and 0xa5a5 (alternating 1's and 0's). Normally the media worst case pattern is used.

Format also has an option to perform an extended "severe burnin," which makes 46 passes using different patterns. Using this option, sectors with any errors of any size are marked bad. This test runs for many hours, depending on the disk and processor.

Each time *format* is run a completely new bad sector table is generated based on errors encountered while formatting. The device driver, however, will always attempt to read any existing bad sector table when the device is first opened. Thus, if a disk pack has never previously been formatted, or has been formatted with different sectoring, five error messages will be printed when the driver attempts to read the bad sector table; these diagnostics should be ignored.

Formatting a 400 megabyte disk on a MASSBUS disk controller usually takes about 20 minutes. Formatting on a UNIBUS disk controller takes significantly longer. For every hundredth cylinder formatted *format* prints a message indicating the current cylinder being formatted. (This message is just to reassure people that nothing is is amiss.)

Format uses the standard notation of the standalone i/o library in identifying a drive to be formatted. A drive is specified as *zz(x,y)*, where *zz* refers to the controller type (either *hp* or *up*), *x* is the unit number of the drive; 8 times the UNIBUS or MASSBUS adaptor number plus the MASSBUS drive number or UNIBUS drive unit number; and *y* is the file system partition on drive *x* (this should always be 0). For example, "hp(1,0)" indicates that drive 1 on MASSBUS adaptor 0 should be formatted; while "up(10,0)" indicates UNIBUS drive 2 on UNIBUS adaptor 1 should be formatted.

Before each formatting attempt, *format* prompts the user in case debugging should be enabled in the appropriate device driver. A carriage return disables debugging information.

Format should be used prior to building file systems (with *newfs*(8)) to insure all sectors with uncorrectable media errors are remapped. If a drive develops uncorrectable defects after formatting, the program *badsect*(8) must be used.

EXAMPLE

A sample run of *format* is shown below. In this example (using a VAX-11/780), *format* is loaded from the console floppy; on an 11/750 *format* will be loaded from the root file system.

Boldface means user input. As usual, "#" and "@" may be used to edit input.

```
>>>L FORMAT
                        LOAD DONE, 00004400 BYTES LOADED
>>>S 2
Disk format/check utility

Enable debugging (0=none, 1=bse, 2=ecc, 3=bse+ecc)? 0
Device to format? hp(8,0)
(error messages may occur as old bad sector table is read)
Formatting drive hp0 on adaptor 1: verify (yes/no)? yes
Device data: #cylinders=842, #tracks=20, #sectors=48
Available test patterns are:
                        1 - (f00f) rh750 worst case
                        2 - (ec6d) media worst case
                        3 - (a5a5) alternating 1's and 0's
                        4 - (ffff) Severe burnin (takes several hours)
Pattern (one of the above, other to restart)? 2
Start formatting...make sure the drive is online
 ...
(soft ecc's and other errors are reported as they occur)
 ...
(if 4 write check errors were found, the program terminates like this...)
 ...
Errors:
Write check: 4
Bad sector: 0
ECC: 0
Skip sector: 0
Total of 4 hard errors found.
Writing bad sector table at block 808271
(808271 is the block # of the first block in the bad sector table)
Done
(...program restarts to allow formatting other disks)
(...to abort halt machine with ^P)
```

DIAGNOSTICS

The diagnostics are intended to be self explanatory.

USING DEC SOFTWARE TO FORMAT

Warning: These instructions are for people with 11/780 CPU's. The steps needed for 11/750 or 11/730 cpu's are similar, but not covered in detail here.

The formatting procedures are different for each type of disk. Listed here are the formatting procedures for RK07's, RP0X, and RM0X disks.

You should shut down UNIX and halt the machine to do any disk formatting. Make certain you put in the pack you want formatted. It is also a good idea to spin down or write protect the disks you don't want to format, just in case.

Formatting an RK07. Load the console floppy labeled, "RX11 VAX DSK LD DEV #1" in the console disk drive, and type the following commands:

```
>>>BOOT
DIAGNOSTIC SUPERVISOR.  ZZ-ESSAA-X5.0-119  23-JAN-1980 12:44:40.03
DS>ATTACH DW780 SBI DW0 3 5
DS>ATTACH RK611 DMA
DS>ATTACH RK07 DW0 DMA0
DS>SELECT DMA0
DS>LOAD EVRAC
DS>START/SEC:PACKINIT
```

Formatting an RP0X. Follow the above procedures except that the ATTACH and SELECT lines should read:

```
DS>ATTACH RH780 SBI RH0 8 5
DS>ATTACH RP0X RH0 DBA0(RP0X is, e.g. RP06)
DS>SELECT DBA0
```

This is for drive 0 on mba0; use 9 instead of 8 for mba1, etc.

Formatting an RM0X. Follow the above procedures except that the ATTACH and SELECT lines should read:

```
DS>ATTACH RH780 SBI RH0 8 5
DS>ATTACH RM0X RH0 DRA0
DS>SELECT DRA0
```

Don't forget to put your UNIX console floppy back in the floppy disk drive.

SEE ALSO

bad144(8), badsect(8), newfs(8)

BUGS

An equivalent facility should be available which operates under a running UNIX system.

It should be possible to define more precisely what a "hard ECC" error is; e.g. the maximum unacceptable ECC width.

NAME

fsck − file system consistency check and interactive repair

SYNOPSIS

/etc/fsck −p [filesystem ...]
/etc/fsck [**−b** block#] [**−y**] [**−n**] [filesystem] ...

DESCRIPTION

The first form of *fsck* preens a standard set of filesystems or the specified file systems. It is normally used in the script **/etc/rc** during automatic reboot. In this case *fsck* reads the table **/etc/fstab** to determine which file systems to check. It uses the information there to inspect groups of disks in parallel taking maximum advantage of i/o overlap to check the file systems as quickly as possible. Normally, the root file system will be checked on pass 1, other "root" ("a" partition) file systems on pass 2, other small file systems on separate passes (e.g. the "d" file systems on pass 3 and the "e" file systems on pass 4), and finally the large user file systems on the last pass, e.g. pass 5. A pass number of 0 in fstab causes a disk to not be checked; similarly partitions which are not shown as to be mounted "rw" or "ro" are not checked.

The system takes care that only a restricted class of innocuous inconsistencies can happen unless hardware or software failures intervene. These are limited to the following:

Unreferenced inodes

Link counts in inodes too large

Missing blocks in the free list

Blocks in the free list also in files

Counts in the super-block wrong

These are the only inconsistencies which *fsck* with the **−p** option will correct; if it encounters other inconsistencies, it exits with an abnormal return status and an automatic reboot will then fail. For each corrected inconsistency one or more lines will be printed identifying the file system on which the correction will take place, and the nature of the correction. After successfully correcting a file system, *fsck* will print the number of files on that file system and the number of used and free blocks.

Without the **−p** option, *fsck* audits and interactively repairs inconsistent conditions for file systems. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that a number of the corrective actions which are not fixable under the **−p** option will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *fsck* will default to a **−n** action.

Fsck has more consistency checks than its predecessors *check, dcheck, fcheck,* and *icheck* combined.

The following flags are interpreted by *fsck.*

−b Use the block specified immediately after the flag as the super block for the file system. Block 32 is always an alternate super block.

−y Assume a yes response to all questions asked by *fsck;* this should be used with great caution as this is a free license to continue after essentially unlimited trouble has been encountered.

−n Assume a no response to all questions asked by *fsck;* do not open the file system for writing.

If no filesystems are given to *fsck* then a default list of file systems is read from the file **/etc/fstab**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Directory size not of proper format.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated inode.
   Inode number out of range.
8. Super Block checks:

   More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. The only restriction is that the directory **lost+found** must preexist in the root of the filesystem being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster.

FILES

/etc/fstab	contains default list of file systems to check.

DIAGNOSTICS

The diagnostics produced by *fsck* are intended to be self-explanatory.

SEE ALSO

fstab(5), fs(5), newfs(8), mkfs(8), crash(8V), reboot(8)

BUGS

Inode numbers for **.** and **..** in each directory should be checked for validity.

There should be some way to start a **fsck −p** at pass *n*.

NAME

fsckblks − print alternate super block numbers for fsck -b

SYNOPSIS

/etc/fsckblks [−v] [mkfs-options] special disk-type

DESCRIPTION

Fsckblks is used to print out the alternate super block numbers for a disk, for use with the −b option of *fsck*(8). This is useful in case your primary super block is trashed and you forgot to write down (or lost) the numbers that *mkfs*(8) printed three years ago when you constructed the file system. You really don't want to have to run *mkfs* or *newfs* because that would erase the file system you're trying to repair.

Fsckblks looks up the type of disk the file system is on in the disk description file */etc/disktab*, calculates the parameters that would be used in calling *mkfs*, then prints the configuration information that *mkfs* would print. One way to view *fsckblks* is as "*newfs* without side-effects".

If the −v option is supplied, *fsckblks* will print out its actions, including the parameters which would be passed to *mkfs*.

Options which may be used to override default parameters which would passed to *mkfs* are:

−s size The size of the file system in sectors.

−b block-size
The block size of the file system in bytes.

−f frag-size
The fragment size of the file system in bytes.

−t #tracks/cylinder

−c #cylinders/group
The number of cylinders per cylinder group in a file system. The default value used is 16.

−m free space %
The percentage of space reserved from normal users; the minimum free space threshhold. The default value used is 10%.

−r revolutions/minute
The speed of the disk in revolutions per minute (normally 3600).

−S sector-size
The size of a sector in bytes (almost never anything but 512).

−i number of bytes per inode
This specifies the density of inodes in the file system. The default is to create an inode for each 2048 bytes of data space. If fewer inodes are desired, a larger number should be used; to create more inodes a smaller number should be given.

FILES

/etc/disktab	for disk geometry and file system partition information

SEE ALSO

disktab(5), fs(5), diskpart(8), fsck(8), format(8), mkfs(8), newfs(8), tunefs(8)

McKusick, Joy, Leffler; "A Fast File System for Unix", Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720; TR #7, September 1982.

BUGS

Should figure out the type of the disk without the user's help.

Might not print out all the diagnostic warnings that *mkfs* would; however, *fsckblks* is useful only with disk parameters that have already been used to construct the broken filesystem.

Might not print the right answers if someone changes *mkfs* or *newfs* and doesn't change *fsckblks.*

Not much use if the root partition is broken; however, you can run it on any system with an identical /etc/disktab, even if the specified special device doesn't exist, since it's never actually used by *fsckblks* anyway.

NAME

fstat − filter filenames according to commands in a status file

SYNOPSIS

find / -print | fstat specfile

DESCRIPTION

No documentation yet. Here are some examples I culled from Hartwell's usage of the program.

```
# get rid of junk files which haven't been accessed in 3 days
        atime           >= 1w
        type            f
        anyof {
           basename     ".BAK$"
           basename     ".CKP$"
           basename     ".otl$"
           basename     "↑a.out$"
           basename     "↑core$"
           basename     "↑.emacs_[0-9][0-9]*$"
           basename     "↑#"
        }
        exec            "rm - '%N'"
# get rid of ,* files 1 day old
        atime           >= 1d
        type            f
        basename        "↑,"
        exec            "rm - '%N'"
# match /tmp/Emacs-tty* files (so next pattern is not done)
        name            "↑/tmp/Emacs-tty[↑/][↑/]*$"
# clean out regular files in /tmp and /usr/tmp (not subdirectories)
        atime           >= 1h
        type            f
        anyof {
           name         "↑/tmp/[↑/][↑/]*$"
           name         "↑/usr/tmp/[↑/][↑/]*$"
        }
        exec            "rm - '%N'"
---------------------------------------
# This specification file weeds out files which should not be backed up.
        anyof {
           size  = 0b
           size  > 500kb
           type d
           type c
           type b
           type l
           basename     "↑core$"
           basename     "↑rogue.save$"
           basename     "↑mbox$"
           basename     "↑.emacs_[0-9][0-9]*$"
           basename     ".BAK$"
           basename     ".CKP$"
           basename     ".o$"
           basename     ".b$"
           basename     ".a$"
```

```
            basename    ".press$"
            basename    ".imp$"
            basename    ".stip[ABC]$"
            basename    "↑,"
            magic       "0407, 0410, 0413, 0404, 0411, 0700200000"
        }
        succeed
# Anything that gets thru is a file that ought to be backed up.
        echo    "%N0
```

AUTHOR

Steve Hartwell. Manual page by Brian Reid.

NAME

ftpd − DARPA Internet File Transfer Protocol server

SYNOPSIS

/etc/ftpd [−d] [−l] [−ttimeout **]**

DESCRIPTION

Ftpd is the DARPA Internet File Transfer Protocol server process. The server uses the TCP protocol and listens at the port specified in the "ftp" service specification; see *services*(5).

If the **−d** option is specified, each socket created will have debugging turned on (SO.DEBUG). With debugging enabled, the system will trace all TCP packets sent and received on a socket. The program *trpt*(8C) may then be used to interpret the packet traces.

If the **−l** option is specified, each ftp session is logged on the standard output. This allows a line of the form '/etc/ftpd -l > /tmp/ftplog" to be used to conveniently maintain a log of ftp sessions.

The ftp server will timeout an inactive session after 60 seconds. If the **−t** option is specified, the inactivity timeout period will be set to *timeout.*

The ftp server currently supports the following ftp requests; case is not distinguished.

Request	**Description**
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory ("ls -lg")
MODE	specify data transfer *mode*
NLST	give name list of files in directory ("ls")
NOOP	do nothing
PASS	specify password
PORT	specify data connection port
QUIT	terminate session
RETR	retrieve a file
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
STRU	specify data transfer *structure*
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

The remaining ftp requests specified in Internet RFC 765 are recognized, but not implemented.

Ftpd interprets file names according to the "globbing" conventions used by *csh*(1). This allows users to utilize the metacharacters "*?[]{}~".

Ftpd authenticates users according to three rules.

1) The user name must be in the password data base, */etc/passwd*, and not have a null password. In this case a password must be provided by the client before any file

operations may be performed.

2) The user name must not appear in the file */etc/ftpusers*.

3) If the user name is "anonymous" or "ftp", an anonymous ftp account must be present in the password file (user "ftp"). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, *ftpd* takes special measures to restrict the client's access privileges. The server performs a *chroot*(2) command to the home directory of the "ftp" user. In order that system security is not breached, it is recommended that the "ftp" subtree be constructed with care; the following rules are recommended.

~ftp) Make the home directory owned by "ftp" and unwritable by anyone.

~ftp/bin)
Make this directory owned by the super-user and unwritable by anyone. The program *ls*(1) must be present to support the list commands. This program should have mode 111.

~ftp/etc)
Make this directory owned by the super-user and unwritable by anyone. The files *passwd*(5) and *group*(5) must be present for the *ls* command to work properly. These files should be mode 444.

~ftp/pub)
Make this directory mode 777 and owned by "ftp". Users should then place files which are to be accessible via the anonymous account in this directory.

SEE ALSO
ftp(1C),

BUGS
There is no support for aborting commands.

The anonymous account is inherently dangerous and should avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user id of the logged in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

NAME

ftpser – PUP File Transer Protocol Service

SYNOPSIS

/etc/pup/ftpser [arg1] [arg2]

DESCRIPTION

Ftpser provides the PUP File Transfer Protocol service for a Unix time-sharing system. You must have a valid user name and password to access any files. The server normally runs as root, and then does a setuid to users for each connection.

If one command line argument is given, then helpful debugging information will be written in the argv area to be seen with the *ps* command. With two arguents, much more debugging information will be printed.

SEE ALSO

pupftp(1), puptelnet(1), netalias(1), remote(1)

AUTHOR

Bill Nowicki

BUGS

There should be a way of providing or prohibiting annonymous logins, like the 4.2 IP ftp server.

NAME

gatewayinfo – Pup GatewayInfo routing table server

SYNOPSIS

/etc/pup/gatewayinfo [–b] [–d] [–e] [–i] [–r] [–s] [–t]

DESCRIPTION

Gatewayinfo is a server that keeps track of Pup routing information. Other programs can ask *gatewayinfo* for this information using IPC requests. For the format of these requests, see the header file <*pup/puprouter.h*>. If the Pup gateway program is running, *gatewayinfo* also tells other hosts and gateways how the network topology looks from our point of view.

Gatewayinfo should be run out of */etc/pup/rc* before any other programs; it should be run without an '&', so that it has time to get ready before other programs start making requests. When it is ready, it will fork and the parent process will exit. It must run as root.

/etc/pup/pupnettab must be properly configured before *gatewayinfo* is run.

If the gateway program is running, it is supposed to tell *gatewayinfo* about this every 30 seconds; this is so *gatewayinfo* will not advertise us as a route to other nets if the gateway program dies.

OPTIONS

–b	Broadcast our routing table every 30 seconds, and when it changes, even if no gateway process is running.
–d	Debug; prints information about error conditions on stderr.
–e	Send empty tables on the network in broadcasts or in response to request packets. This option should only be used in conjunction with –b or –r when the gateway process is known not to be running. Its purpose is to avoid advertising ourself as a route to any other network.
–i	IPC debug; prints debugging information about IPC requests.
–r	Respond to request packets from the network even if the gateway process is not running.
–s	Slow-gateway mode; routing tables send out on the network show route-lengths one hop longer than they actually are; this is to encourage other hosts not to use us as a gateway if there are other, more willing, gateways.
–t	Trace most operations.

The –d and –t options are useful only to wizards. The –b, –r, and –e options are probably useful only on isolated networks with no gateways.

FILES

/etc/pup/pupnettab list of network interfaces

SEE ALSO

pupgateway(8), pupnettab(9)

AUTHOR

Jeffrey Mogul

NAME

gettable — get NIC format host tables from a host

SYNOPSIS

/etc/gettable *host*

DESCRIPTION

Gettable is a simple program used to obtain the NIC standard host tables from a "nicname" server. The indicated *host* is queried for the tables. The tables, if retrieved, are placed in the file *hosts.txt.*

Gettable operates by opening a TCP connection to the port indicated in the service specification for "nicname". A request is then made for "ALL" names and the resultant information is placed in the output file.

Gettable is best used in conjunction with the *htable*(8) program which converts the NIC standard file format to that used by the network library lookup routines.

SEE ALSO

intro(3N), htable(8)

BUGS

Should allow requests for only part of the database.

NAME

getty − set terminal mode

SYNOPSIS

/etc/getty [type]

DESCRIPTION

Getty is invoked by *init*(8) immediately after a terminal is opened, following the making of a connection. While reading the name *getty* attempts to adapt the system to the speed and type of terminal being used.

Init calls *getty* with an argument specified by the *ttys* file entry for the terminal line. The argument can be used to make *getty* treat the line specially. This argument is used as an index into the *gettytab*(5) database, to determine the characteristics of the line. If there is no argument, or there is no such table, the **default** table is used. If there is no **/etc/gettytab** a set of system defaults is used. If indicated by the table located, *getty* will clear the terminal screen, print a banner heading, and prompt for a login name. Usually either the banner of the login prompt will include the system hostname. Then the user's name is read, a character at a time. If a null character is received, it is assumed to be the result of the user pushing the 'break' ('interrupt') key. The speed is usually then changed and the 'login:' is typed again; a second 'break' changes the speed again and the 'login:' is typed once more. Successive 'break' characters cycle through the some standard set of speeds.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *tty*(4)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is nonempty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, login is called with the user's name as argument.

Most of the default actions of *getty* can be circumvented, or modified, by a suitable *gettytab* table.

Getty can be set to timeout after some interval, which will cause dial up lines to hang up if the login name is not entered reasonably quickly.

FILES

/etc/gettytab

SEE ALSO

gettytab(5), init(8), login(1), ioctl(2), tty(4), ttys(5).

BUGS

Currently, the format of **/etc/ttys** limits the permitted table names to a single character, this should be expanded.

/etc/ttys should be replaced completely.

NAME

gsa − group system accounting

SYNOPSIS

gsa [−cprsunx] [−g group **] [−f** filename **] [−w** filename **]**

DESCRIPTION

Gsa reads and formats system accounting information by group. This is meant to help managers of project-oriented systems to keep track of usage on a higher level than by user.

Gsa lists each group and the primary users of that group, "primary" meaning that the users default to that group on login. For each user, selected fields are printed; these default to the user's *real name,* his total *cpu time* in minutes, and his total *connect time* in hours. With each numeric field the percentage of system grand total is printed.

After the users of each group are listed, system usage is summarized by group.

Gsa requires that *sa(8)* be run in advance to create a per-user summary file; */usr/adm/usracct* is the output file of *sa,* and it is the default input file for *gsa.* Many systems run *sa* as part of a daily routine, making it unnecessary to run it manually before running *gsa.* This is the case on Diablo.

The default file for connect time information is */usr/adm/wtmp.*

Options to *gsa* are:

c	Don't include connect time information. This saves a great deal of time on execution.
f	Read *filename* instead of */usr/adm/usracct* for per-user cpu summary.
g	Print only information about *group;* percentages become those of that group's totals rather than the system totals. The system-wide group summary is omitted, unless forced with the 's' option. Useful for group managers.
p	Print stats about number of processes.
r	Don't print users' real names. Useful if you need more horizontal space on the screen.
s	Print only the system-wide group summary, omit stats on individual users.
u	Don't print cpu time statistics.
w	Read *filename* instead of */usr/adm/wtmp* for connect-time information.
n	Give names of groups which have no primary (i.e., login) users, as well as groups which have no users at all, at the end of the summary.
x	Exclude (ignore) wtmp entries made after the given *usracct* file was written. Useful for synchronizing *usracct* and *wtmp* files last updated at grossly different times. If the first entry in the *wtmp* file was made after the *usracct* file was written, the entire *wtmp* file will be ignored. If the two files were initialized at different times, this option will not help that.

AUTHOR

Bill Burgess

FILES

/etc/group	group names and numbers
/etc/passwd	user names and default groups
/usr/adm/usracct	default per-user summary
/usr/adm/wtmp	default connect-time log

SEE ALSO

group(5), passwd(5), wtmp(5), sa(8)

BUGS

Gsa can't know when the *usracct* file was first written. The only way to know the time when the cpu statistics began to be collected is to know when *sa* first CREATED (NOT modified) *usracct.* If *sa* is run as part of a daily and/or monthly routine, this should not be too difficult. Look for files like /usr/adm/daily.ctl or /usr/adm/monthly.ctl, or in /usr/lib/crontab. On Diablo both cpu and connect accounting files are initialized on the first day of each month.

NAME

halt − stop the processor

SYNOPSIS

/etc/halt [−n] [−q] [−y]

DESCRIPTION

Halt writes out sandbagged information to the disks and then stops the processor. The machine does not reboot, even if the auto-reboot switch is set on the console.

The **−n** option prevents the sync before stopping. The **−q** option causes a quick halt, no graceful shutdown is attempted. The **−y** option is needed if you are trying to halt the system from a dialup.

SEE ALSO

reboot(8), shutdown(8)

BUGS

It is very difficult to halt a VAX, as the machine wants to then reboot itself. A rather tight loop suffices.

NAME

htable − convert NIC standard format host tables

SYNOPSIS

/etc/htable [−c *connected-nets*] [−l *local-nets*] *file*

DESCRIPTION

Htable is used to convert host files in the format specified in Internet RFC 810 to the format used by the network library routines. Three files are created as a result of running *htable*: *hosts*, *networks*, and *gateways*. The *hosts* file is used by the *gethostent*(3N) routines in mapping host names to addresses. The *networks* file is used by the *getnetent*(3N) routines in mapping network names to numbers. The *gateways* file is used by the routing daemon in identifying "passive" Internet gateways; see *routed*(8C) for an explanation.

If any of the files *localhosts*, *localnetworks*, or *localgateways* are present in the current directory, the file's contents is prepended to the output file. Of these, only the gateways file is interpreted. This allows sites to maintain local aliases and entries which are not normally present in the master database. Only one gateway to each network will be placed in the gateways file; a gateway listed in the localgateways file will override any in the input file.

A list of networks to which the host is directly connected is specified with the −c flag. The networks, separated by commas, may be given by name or in internet-standard dot notation, e.g. −c arpanet,128.32,local-ether-net. *Htable* only includes gateways which are directly connected to one of the networks specified, or which can be reached from another gateway on a connected net.

If the −l option is given with a list of networks (in the same format as for −c), these networks will be treated as "local," and information about hosts on local networks is taken only from the localhosts file. Entries for local hosts from the main database will be omitted. This allows the localhosts file to completely override any entries in the input file.

Htable is best used in conjunction with the *gettable*(8C) program which retrieves the NIC database from a host.

SEE ALSO

intro(3N), gettable(8C)

NAME

icheck — file system storage consistency check

SYNOPSIS

/etc/icheck [−s] [−b numbers **]** [filesystem **]**

DESCRIPTION

N.B.: *Icheck* is obsoleted for normal consistency checking by *fsck*(8).

Icheck examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. If the file system is not specified, a set of default file systems is checked. The normal output of *icheck* includes a report of

> The total number of files and the numbers of regular, directory, block special and character special files.
>
> The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.
>
> The number of free blocks.
>
> The number of blocks missing; i.e. not in any file nor in the free list.

The **−s** option causes *icheck* to ignore the actual free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the super-block will not continue to be used. Notice also that the words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The **−s** option causes the normal output reports to be suppressed.

Following the **−b** option is a list of block numbers; whenever any of the named blocks turns up in a file, a diagnostic is produced.

Icheck is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), dcheck(8), ncheck(8), fs(5), clri(8)

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) *icheck* announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and *icheck* considers it to contain 0. 'Bad freeblock' means that a block number outside the available space was encountered in the free list. '*n* dups in free' means that *n* blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

BUGS

Since *icheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

The system should be fixed so that the reboot after fixing the root file system is not necessary.

NAME

ifconfig – configure network interface parameters

SYOPNSIS

/etc/ifconfig interface [*address*] [*parameters*]

DESCRIPTION

Ifconfig is used to assign an address to a network interface and/or configure network interface parameters. *Ifconfig* must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address. The *interface* parameter is a string of the form "name unit", e.g. "en0", while the address is either a host name present in the host name data base, *hosts*(5), or a DARPA Internet address expressed in the Internet standard "dot notation".

The following parameters may be set with *ifconfig*:

up	Mark an interface "up".
down	Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface.
trailers	Enable the use of a "trailer" link level encapsulation when sending (default). If a network interface supports *trailers*, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver.
–trailers	Disable the use of a "trailer" link level encapsulation.
arp	Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addreses and 10Mb/s Ethernet addresses.
–arp	Disable the use of the Address Resolution Protocol.
ipsubwidth *nnn*	Sets the "Internet Protocol Subnet width" for the interface to *nnn*. This should be used *before* setting the interface address; for example, `/etc/ifconfig en0 ipsubwidth 8` `/etc/ifconfig en0 36.40.0.102 -trailers`

Ifconfig displays the current configuration for a network interface when no optional parameters are supplied.

Only the super-user may modify the configuration of a network interface.

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

rc(8), intro(4N), netstat(1)

NAME

implog − IMP log interpreter

SYNOPSIS

/etc/implog [−D] [−f] [−c] [−l [*link* **]] [−h** *host#* **] [−i** *imp#* **] [−t** *message-type* **]**

DESCRIPTION

Implog is program which interprets the message log produced by *implogd*(8C).

If no arguments are specified, *implog* interprets and prints every message present in the message file. Options may be specified to force printing only a subset of the logged messages.

−D Do not show data messages.

−f Follow the logging process in action. This flags causes *implog* to print the current contents of the log file, then check for new logged messages every 5 seconds.

−c In addition to printing any data messages logged, show the contents of the data in hexadecimal bytes.

−l [*link#*]
Show only those messages received on the specified "link". If no value is given for the link, the link number of the IP protocol is assumed.

−h *host#*
Show only those messages received from the specified host. (Usually specified in conjunction with an imp.)

−i *imp#*
Show only those messages received from the specified imp.

−t *message-type*
Show only those messages received of the specified message type.

SEE ALSO

imp(4P), implogd(8C)

BUGS

Can not specify multiple hosts, imps, etc. Can not follow reception of messages without looking at those currently in the file.

NAME

implogd — IMP logger process

SYNOPSIS

/etc/implogd [−d]

DESCRIPTION

Implogd is program which logs messages from the IMP, placing them in the file */usr/adm/implog*.

Entries in the file are variable length. Each log entry has a fixed length header of the form:

```
struct sockstamp {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        time_t  sin_time;
        int     sin_len;
};
```

followed, possibly, by the message received from the IMP. Each time the logging process is started up it places a time stamp entry in the file (a header with *sin_len* field set to 0).

The logging process will catch only those message from the IMP which are not processed by a protocol module, e.g. IP. This implies the log should contain only status information such as "IMP going down" messages and, perhaps, stray NCP messages.

SEE ALSO

imp(4P), implog(8C)

BUGS

The messages should probably be sent to the system error logging process instead of maintaining yet another log file.

NAME

inetd − DARPA little protocol server

SYNOPSIS

/etc/inetd [−d] [−f program] [−o options] [−q program] protocol/service ...

DESCRIPTION

The *inetd* server implements a number of the so−called "little" protocols in the IP/TCP protocol suite. In particular, the following protocols are implemented at this time:

Service	RFC	Description
echo	862	sends back whatever you send it
sink	863	throws away whatever you send it
daytime	867	provides the day and time in ASCII
time	868	provides the number of seconds from a reference time
users	866	lists the currently active users
chargen	864	sends you ASCII data
qotd	865	sends you a short ASCII message
finger	742	provides information on the activity of a user

See *services* (5) for the list of ports that *inetd* will operate at. The *inetd* server supports these services using both TCP and UDP (see *protocols* (5)).

If the '−d' option is specified, each socket created by *inetd* will have debugging enabled (see SO_DEBUG in *socket* (2)). If the '−f' option is given, the following argument is taken to be the pathname of the program to run when servicing *finger* requests. Similarly, if the '−o' option is given, the following argument is take to be the options that should be given to this program. Finally, if the '−q' option is given, the following argument is taken to be the pathname of the program to run when servicing *qotd* requests.

SEE ALSO

rfinger(1C)

NAME
init − process control initialization

SYNOPSIS
/etc/init

DESCRIPTION
Init is invoked inside UNIX as the last step in the boot procedure. It normally then runs the automatic reboot sequence as described in *reboot*(8), and if this succeeds, begins multi-user operation. If the reboot fails, it commences single user operation by giving the super-user a shell on the console. It is possible to pass parameters from the boot program to *init* so that single user operation is commenced immediately. When such single user operation is terminated by killing the single-user shell (i.e. by hitting ^D), *init* runs */etc/rc* without the reboot parameter. This command file performs housekeeping operations such as removing temporary files, mounting file systems, and starting daemons.

In multi-user operation, *init's* role is to create a process for each terminal port on which a user may log in. To begin such operations, it reads the file */etc/ttys* and forks several times to create a process for each terminal specified in the file. Each of these processes opens the appropriate terminal for reading and writing. These channels thus receive file descriptors 0, 1 and 2, the standard input and output and the diagnostic output. Opening the terminal will usually involve a delay, since the *open* is not completed until someone is dialed up and carrier established on the channel. If a terminal exists but an error occurs when trying to open the terminal *init* complains by writing a message to the system console; the message is repeated every 10 minutes for each such terminal until the terminal is shut off in /etc/ttys and init notified (by a hangup, as described below), or the terminal becomes accessible (init checks again every minute). After an open succeeds, */etc/getty* is called with argument as specified by the second character of the *ttys* file line. *Getty* reads the user's name and invokes *login* to log in the user and execute the Shell.

Ultimately the Shell will terminate because of an end-of-file either typed explicitly or generated as a result of hanging up. The main path of *init*, which has been waiting for such an event, wakes up and removes the appropriate entry from the file *utmp*, which records current users, and makes an entry in */usr/adm/wtmp*, which maintains a history of logins and logouts. The *wtmp* entry is made only if a user logged in successfully on the line. Then the appropriate terminal is reopened and *getty* is reinvoked.

Init catches the *hangup* signal (signal SIGHUP) and interprets it to mean that the file */etc/ttys* should be read again. The Shell process on each line which used to be active in *ttys* but is no longer there is terminated; a new process is created for each added line; lines unchanged in the file are undisturbed. Thus it is possible to drop or add phone lines without rebooting the system by changing the *ttys* file and sending a *hangup* signal to the *init* process: use 'kill −HUP 1.'

Init will terminate multi-user operations and resume single-user mode if sent a terminate (TERM) signal, i.e. "kill −TERM 1". If there are processes outstanding which are deadlocked (due to hardware or software failure), *init* will not wait for them all to die (which might take forever), but will time out after 30 seconds and print a warning message.

Init will cease creating new *getty*'s and allow the system to slowly die away, if it is sent a terminal stop (TSTP) signal, i.e. "kill −TSTP 1". A later hangup will resume full multi-user operations, or a terminate will initiate a single user shell. This hook is used by *reboot*(8) and *halt*(8).

Init's role is so critical that if it dies, the system will reboot itself automatically. If, at bootstrap time, the *init* process cannot be located, the system will loop in user mode at location 0x13.

DIAGNOSTICS
init: *tty*: **cannot open.** A terminal which is turned on in the *rc* file cannot be opened, likely because the requisite lines are either not configured into the system or the associated device

was not attached during boot-time system configuration.

WARNING: Something is hung (wont die); ps axl advised. A process is hung and could not be killed when the system was shutting down. This is usually caused by a process which is stuck in a device driver due to a persistent device error condition.

FILES

/dev/console, /dev/tty*, /etc/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc

SEE ALSO

login(1), kill(1), sh(1), ttys(5), crash(8V), getty(8), rc(8), reboot(8), halt(8), shutdown(8)

NAME

insecure − user security monitor

SYNOPSIS

/etc/insecure [−s] [username ...]

DESCRIPTION

Insecure is useful in spotting potential security holes due to easily-guessed passwords. It will try to guess the passwords of the specified users, (or of all users, if none are specified.) If an account has an easily-guessed password, the account name and shell will be on the standard output.

The −s flag suppresses guessing for users whose login shell is not one of those allowed by *chsh*(1). This avoids false positives for pseudo-users which are meant to be open accounts; e.g., a user named "finger" with no password with a login shell of /usr/ucb/finger.

DIAGNOSTICS

The exit status is the number of passwords guessed.

SEE ALSO

chsh(1), passwd(1)

BUGS

It takes about 7 cpu seconds on a Vax-11/750 to check each user.

The very existence of this program might be a security hole. Install it so as to be unreadable/unexecutable by random users.

The −s mechanism leaves something to be desired, especially if additional interactive login shells are allowed by *chsh*(1).

NAME
kgmon − generate a dump of the operating system's profile buffers

SYNOPSIS
/etc/kgmon [**−b**] [**−h**] [**−r**] [**−p**] [system] [memory]

DESCRIPTION
Kgmon is a tool used when profiling the operating system. When no arguments are supplied, *kgmon* indicates the state of operating system profiling as running, off, or not configured. (see *config*(8)) If the **−p** flag is specified, *kgmon* extracts profile data from the operating system and produces a *gmon.out* file suitable for later analysis by *gprof*(1).

The following options may be specified:

−b	Resume the collection of profile data.
−h	Stop the collection of profile data.
−p	Dump the contents of the profile buffers into a *gmon.out* file.
−r	Reset all the profile buffers. If the **−p** flag is also specified, the *gmon.out* file is generated before the buffers are reset.

If neither **−b** nor **−h** is specified, the state of profiling collection remains unchanged. For example, if the **−p** flag is specified and profile data is being collected, profiling will be momentarily suspended, the operating system profile buffers will be dumped, and profiling will be immediately resumed.

FILES
/vmunix − the default system
/dev/kmem − the default memory

SEE ALSO
gprof(1), config(8)

DIAGNOSTICS
Users with only read permission on /dev/kmem cannot change the state of profiling collection. They can get a *gmon.out* file with the warning that the data may be inconsistent if profiling is in progress.

NAME
leaf – PUP Leaf Remote File Access Protocol Server

SYNOPSIS
/etc/pup/leaf **[-i] [-d] [-42] [-p] [-t] [-l]**

DESCRIPTION
The *Leaf* server provides the Leaf remote file access protocol service for a Unix time-sharing system. You must have a valid user name and password to access any files. The server normally runs as root, and then does a setuid to users for each connection.

-i	This option is used to enable version number simulation as used in Xerox InterLisp on the 1100 series workstations.
-d	This option turns on much debugging information.
-42	This option indicates that socket 42 (octal) is to be used instead of the default socket of 043.
-p	This option dumps every packet, resulting in a huge amount of debugging information.
-t	This option sends the debug output to the terminal (standard output) instead of the default of /usr/adm/leaf.log.
-l	This option logs all leaf-level operations like opening and closing files.

SEE ALSO
ftpser(8)

Leaf and Seqiun Protocols The protocol specification by Jeff Mogul, 1982

Unix Leaf Server Documentation by Craig, Mogul, and Nowicki, updated 1984.

AUTHORS
Originally written by Doug Hartman and John Craig. Bill Nowicki of Stanford University has been maintaining it for the last few years. Jeff Mogul of Stanford, Jim Koda of ISI, and Craig Milo Rogers of ISI have also made contributions.

BUGS
There should be a way of providing or prohibiting annonymous logins, like the 4.2 IP ftp server.

NAME

lpc — line printer control program

SYNOPSIS

/etc/lpc [command [argument ...]]

DESCRIPTION

Lpc is used by the system administrator to control the operation of the line printer system. For each line printer configured in /etc/printcap, *lpc* may be used to:

- disable or enable a printer,
- disable or enable a printer's spooling queue,
- rearrange the order of jobs in a spooling queue,
- find the status of printers, and their associated spooling queues and printer dameons.

Without any arguments, *lpc* will prompt for commands from the standard input. If arguments are supplied, *lpc* interprets the first argument as a command and the remaining arguments as parameters to the command. The standard input may be redirected causing *lpc* to read commands from file. Commands may be abreviated; the following is the list of recognized commands.

? [command ...]

help [command ...]
: Print a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

abort { all | printer ... }
: Terminate an active spooling daemon on the local host immediately and then disable printing (preventing new daemons from being started by *lpr*) for the specified printers.

clean { all | printer ... }
: Remove all files beginning with "cf", "tf", or "df" from the specified printer queue(s) on the local machine.

enable { all | printer ... }
: Enable spooling on the local queue for the listed printers. This will allow *lpr* to put new jobs in the spool queue.

exit

quit
: Exit from lpc.

disable { all | printer ... }
: Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by *lpr*.

restart { all | printer ... }
: Attempt to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. *Lpq* will report that there is no daemon present when this condition occurs.

start { all | printer ... }
: Enable printing and start a spooling daemon for the listed printers.

status [all] [printer ...]
: Display the status of daemons and queues on the local machine.

stop { all | printer ... }
: Stop a spooling daemon after the current job completes and disable printing.

topq printer [jobnum ...] [user ...]
Place the jobs in the order listed at the top of the printer queue.

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/usr/spool/*/lock	lock file for queue control

SEE ALSO

lpd(8), lpr(1), lpq(1), lprm(1), printcap(5)

DIAGNOSTICS

?Ambiguous command	abreviation matches more than one command
?Invalid command	no match was found
?Privileged command	command can be executed by root only

NAME

lpd — line printer daemon

SYNOPSIS

/usr/lib/lpd [-l] [-L logfile] [port #]

DESCRIPTION

Lpd is the line printer daemon (spool area handler) and is normally invoked at boot time from the *rc*(8) file. It makes a single pass through the *printcap*(5) file to find out about the existing printers and prints any files left after a crash. It then uses the system calls *listen*(2) and *accept*(2) to receive requests to print files in the queue, transfer files to the spooling area, display the queue, or remove jobs from the queue. In each case, it forks a child to handle the request so the parent can continue to listen for more requests. The Internet port number used to rendezvous with other processes is normally obtained with *getservbyname*(3) but can be changed with the *port#* argument. The **−L** option changes the file used for writing error conditions from the system console to *logfile.* The **−l** flag causes *lpd* to log valid requests received from the network. This can be useful for debugging purposes.

Access control is provided by two means. First, All requests must come from one of the machines listed in the file */etc/hosts.equiv*. Second, if the "rs" capability is specified in the *printcap* entry for the printer being accessed, *lpr* requests will only be honored for those users with accounts on the machine with the printer.

The file *lock* in each spool directory is used to prevent multiple daemons from becoming active simultaneously, and to store information about the daemon process for *lpr*(1), *lpq*(1), and *lprm*(1). After the daemon has successfully set the lock, it scans the directory for files beginning with *cf.* Lines in each *cf* file specify files to be printed or non-printing actions to be performed. Each such line begins with a key character to specify what to do with the remainder of the line.

J	Job Name. String to be used for the job name on the burst page.
C	Classification. String to be used for the classification line on the burst page.
L	Literal. The line contains identification info from the password file and causes the banner page to be printed.
T	Title. String to be used as the title for *pr*(1).
H	Host Name. Name of the machine where *lpr* was invoked.
P	Person. Login name of the person who invoked *lpr*. This is used to verify ownership by *lprm*.
M	Send mail to the specified user when the current print job completes.
f	Formatted File. Name of a file to print which is already formatted.
l	Like "f" but passes control characters and does not make page breaks.
p	Name of a file to print using *pr*(1) as a filter.
t	Troff File. The file contains *troff*(1) output (cat phototypesetter commands).
d	DVI File. The file contains *Tex*(l) output (DVI format from Standford).
g	Graph File. The file contains data produced by *plot*(3X).
c	Cifplot File. The file contains data produced by *cifplot*.
v	The file contains a raster image.
r	The file contains text data with FORTRAN carriage control characters.
1	Troff Font R. Name of the font file to use instead of the default.

2	Troff Font I. Name of the font file to use instead of the default.
3	Troff Font B. Name of the font file to use instead of the default.
4	Troff Font S. Name of the font file to use instead of the default.
W	Width. Changes the page width (in characters) used by *pr*(1) and the text filters.
I	Indent. The number of characters to indent the output by (in ascii).
U	Unlink. Name of file to remove upon completion of printing.
N	File name. The name of the file which is being printed, or a blank for the standard input (when *lpr* is invoked in a pipeline).

If a file can not be opened, a message will be placed in the log file (normally the console). *Lpd* will try up to 20 times to reopen a file it expects to be there, after which it will skip the file to be printed.

Lpd uses *flock*(2) to provide exclusive access to the lock file and to prevent multiple deamons from becoming active simultaneously. If the daemon should be killed or die unexpectedly, the lock file need not be removed. The lock file is kept in a readable ASCII form and contains two lines. The first is the process id of the daemon and the second is the control file name of the current job being printed. The second line is updated to reflect the current status of *lpd* for the programs *lpq*(1) and *lprm*(1).

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/dev/lp*	line printer devices
/dev/printer	socket for local requests
/etc/hosts.equiv	lists machine names allowed printer access

SEE ALSO

lpc(8), pac(1), lpr(1), lpq(1), lprm(1), printcap(5)
4.2BSD Line Printer Spooler Manual

NAME

mailer — Mailing list, forwarding, and alias manager

SYNOPSIS

mail mailer

DESCRIPTION

Mailer is a pseudo-user to which you can mail commands to change aliases, mail forwarding, and mailing lists. The commands can be given as the first line of the body of the message, or the "Subject:" line if the body is empty. Most of the arguments are optional; omitting the *user* name assumes the operation is to be done on the user sending the message. A response is mailed back with the results of the command. For users who insist on being terse, you can even say "add list" instead of "add me to list," or "alias smith" instead of "alias smith for me."

COMMANDS

add *name* to *list*	Adds a name to a mailing list which already exists. Valid only if the list is public, or if it is sent by a mail-wizard or a mailing list maintainer.
alias *name* for *user*	Set up an alias for the current user. The alias must not conflict with any existing aliases, user names, or mailing lists.
create *list* with *name,...*	Creates a mailing list with the indicated users as the members and the maintainers. If the word "public" is included, then anybody will be allowed to add themselves to the list. The rest of the message body is used as a description of the purpose of the mailing list.
delete *name* from *list*	The indicated name is deleted from the mailing list. Anybody can delete themselves, or be deleted by a mail wizard or a maintainer of the mailing list.
forward *user* to *name*	The user's mail will be forwarded to the indicated name (which may be at another host). For example, "forward smith to ds@sail."
help	Mails you back a message containing a list of the commands.
keep *user* at *host*	Disables any forwarding for the indicated user. This may be abbreviated to one word, "keep".
list *name,...*	Mails you a message describing the indicated names. They may be aliases, users, or mailing lists.
pluslots *list*	Adds lots of users to the list. The user addresses are provided in the remainder of the body of the message, separated by commas.
remove *name* for *user*	Removes the alias for the indicated user. If someone has an alias that you want, mail a message to mail-wizards.

FILES

/usr/lib/aliases /usr/lib/mailinglists /usr/stanford/lib/vinegar

SEE ALSO

mail(1), aliases(5), newaliases(1), delivermail(8)

AUTHOR

Bill Nowicki, Stanford University

DIAGNOSTICS

Mailed back in a message.

BUGS

Once a mailing list is created, there is no way to modify its parameters (other than its membership), or destroy it, except by editing the text file. Simultaneous update is not checked.

NAME

makedev – make system special files

SYNOPSIS

/dev/MAKEDEV *device...*

DESCRIPTION

MAKEDEV is a shell script normally used to install special files. It resides in the */dev* directory, as this is the normal location of special files. Arguments to *MAKEDEV* are usually of the form *device-name*? where *device-name* is one of the supported devices listed in section 4 of the manual and "?" is a logical unit number (0-9). A few special arguments create assorted collections of devices and are listed below.

std Create the *standard* devices for the system; e.g. /dev/console, /dev/tty. The VAX-11/780 console floppy device, /dev/floppy, and VAX-11/750 and VAX-11/730 console cassette device(s), /dev/tu?, are also created with this entry.

local Create those devices specific to the local site. This request causes the shell file */dev/MAKEDEV.local* to be executed. Site specific commands, such as those used to setup dialup lines as "ttyd?" should be included in this file.

Since all devices are created using *mknod*(8), this shell script is useful only to the super-user.

DIAGNOSTICS

Either self-explanatory, or generated by one of the programs called from the script. Use "sh -x MAKEDEV" in case of trouble.

SEE ALSO

intro(4), config(8), mknod(8)

BUGS

When more than one piece of hardware of the same "kind" is present on a machine (for instance, a dh and a dmf), naming conflicts arise.

NAME

makekey − generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (that is, to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, upper- and lower-case letters, and '.' and '/'. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key.*

The transformation performed is essentially the following: the salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

Makekey is intended for programs that perform encryption (for instance, *ed* and *crypt*(1)). Usually makekey's input and output will be pipes.

SEE ALSO

crypt(1), ed(1)

NAME

miscserver – MiscServices server for Pup

SYNOPSIS

/etc/miscserver [aAdDILmMnNsStT]

DESCRIPTION

Miscserver is a server program which listens for MiscServices requests directed to the Unix system from the Pup Internet. It implements a subset of the MiscServices requests defined in Edition 3 of the Xerox Parc memorandum on Miscellaneous Services. It also implements local extensions to these services.

The program is structured as a loop which listens for packets directed to the MISCSERVICES Pup socket (it accepts both packets specifically destined for the local host, and broadcast packets, but some services reject broadcast packets). The PupType field is extracted from a received packet, and a switch statement is used to dispatch the received packet to one of several action routines. The program can only handle one request at a time, so most of the action routines are entirely included in the server program; this eliminates the time wasted in reading programs off of the disk. However, to achieve some concurrency (and to protect the server against "lost resources"), requests are handled by fork processes.

The services currently implemented are:

MailCheck – The server responds equivalently to both Msg-style and Laurel-style mailcheck requests, and returns the appropriate response. If new mail exists, the PupData portion of the reply include the message " since <time mail was last written>", which can be used by the inquiring program, such as *mailcheck(1).*

WhereIsUser – The server indicates whether the user is logged in or not; it returns an error if the user is not known. (This service has been disabled on Stanford systems, since it confuses the Alto Chat and Telnet programs and causes them to display passwords at odd times.)

SendUserMessage – This is a locally defined protocol which is explained in detail in the manual entry for *msendumsg*(9). This service accepts broadcast packets.

AltoTimeCheck (New standard) – This service returns the time in seconds since midnight, January 1, 1901, which is the Alto internal time format. It also returns timezone and Daylight Savings Time information, in Xerox format.

KissOfDeath – This is a non-standard use of the KissOfDeath protocol; the server process terminates when it receives a KissOfDeath from a process that is newer than itself. (The sending process puts its process creation time into PupID.) This should only be sent by another miscserver process on the same host; the intent is that this is used to prevent more than one miscserver to be running at a time.

Authenticate – This service takes a username and password, and indicates whether the host system considers them to be valid. If a username contains a period followed by a registry name, the username is extracted.

Name lookup – Translates a host name to a Pup internet address. This service conforms fairly well with the Xerox definition of a Pup name.

Address lookup – Translates a Pup internet address to the preferred name..

Sun Boot Load – Download a program file to a Sun workstation. See

/usr/sun/doc/sunboot/SunBoot.press for details.

Sun Boot Directory – Return a directory of standard Sun bootfiles. Like the Alto Boot Directory Protocol, but with different Pup types.

"New" Sun Boot Directory – Improved version of Sun Boot Directory protocol. Directory is returned with an EFTP to the requesting Port.

Alto Boot Load – Download a program file to an Alto. Files are read from /usr/altoboots.

Alto Boot Directory – Return a directory of Alto bootfiles.

Net Directory Attributes – This is a locally-defined protocol that allows access to the "attributes" field of Network Directory entries. The request format is a packet with a Pup address (Port) as data; the reply is a string. Normally, the attributes of an entry include its location.

Load Dolphin Microcode – Download a microcode file to a Dolphin. Files are read from /usr/dolphin_ucode. This function is not well-tested.

The program is meant to be run (with an ampersand) from /etc/pup/rc; only one copy of it should be running, although there is no obvious harm caused by running a second copy, since the protocols are all connectionless. A running copy can be killed without any ill effects (actually, there is a tiny chance that this might corrupt the local copy of the binary-format network directory; as long as there is another copy somewhere on the net, this is not a problem.) However, in general the servers agree among themselves which one is newer, and the older one goes away; thus, there is not much reason to actually kill one.

Options:

If an argument is given, it is taken as a key; each letter of the key may be used to control an option. For each option, a lower case occurance of the key letter turns it "on", while an upper case key letter turns it "off". The possible options are:

a	If on, server responds to Alto boot request (and Alto boot directory requests.) Default: on.
d	If on, server prints debugging information on stderr. Default: off.
l	If on, server prints a log of all requests on stdout. Default: off.
m	If on, server responds to Dolphin microcode loader requests. Default: on.
n	If on, server responds to broadcast name/address requests. If off, server responds to name/address requests only if directed to local host. Default: on.
s	If on, server responds to Sun boot request (and Sun boot directory requests.) Default: on.
t	If on, server responds to broadcast time requests. If off, server responds to time requests only if directed to local host. Default: on.

FILES

/bin/nwrite	used for SendUserMessage service
/etc/pup/Pup-Network.txt	Human-readable Name services database
/etc/pup/Pup-Network.Dir	Binary-format NameServer database
/usr/sun/bootfile	default directory for Sun bootfiles
/usr/altoboots	directory for Alto bootfiles
/usr/dolphin_ucode	directory for Dolphin microcode files

In order to make this program more portable, the name and location of each of these files is defined in a header file, "miscserver.h" in the directory containing the miscserver sources. The system manager may wish to edit this file before compiling the server.

Another compile-time option is whether the Nameserver uses a binary or human-readable database; the former is prefered. If the binary database exists, it will be automatically updated; otherwise, it will *not* be created. New versions may be created with the *buildnetdir*(8) program.

AUTHOR

Jeffrey Mogul

SEE ALSO

msendumsg(9), mmailcheck(9), buildnetdir(8)

DIAGNOSTICS

This is a server program, so no diagnostics should appear. It may cause a core-dump if badly-written action routines are included, of course.

NAME

mkfs − construct a file system

SYNOPSIS

/etc/mkfs special size [nsect] [ntrack] [blksize] [fragsize] [ncpg] [minfree] [rps]

DESCRIPTION

N.B.: file system are normally created with the *newfs*(8) command.

Mkfs constructs a file system by writing on the special file *special.* The numeric size specifies the number of sectors in the file system. *Mkfs* builds a file system with a root directory and a *lost+found* directory. (see *fsck*(8)) The number of i-nodes is calculated as a function of the file system size. No boot program is initialized by *mkfs* (see *newfs*(8).)

The optional arguments allow fine tune control over the parameters of the file system. **Nsect** specify the number of sectors per track on the disk. **Ntrack** specify the number of tracks per cylinder on the disk. **Blksize** gives the primary block size for files on the file system. It must be a power of two, currently selected from 4096 or 8192. **Fragsize** gives the fragment size for files on the file system. The **fragsize** represents the smallest amount of disk space that will be allocated to a file. It must be a power of two currently selected from the range 512 to 8192. **Ncpg** specifies the number of disk cylinders per cylinder group. This number must be in the range 1 to 32. **Minfree** specifies the minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the super-user is allowed to allocate disk blocks. The default value is 10%. If a disk does not revolve at 60 revolutions per second, the **rps** parameter may be specified. Users with special demands for their file systems are referred to the paper cited below for a discussion of the tradeoffs in using different configurations.

SEE ALSO

fs(5), dir(5), fsck(8), newfs(8), tunefs(8)

McKusick, Joy, Leffler; "A Fast File System for Unix", Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720; TR #7, September 1982.

BUGS

There should be some way to specify bad blocks.

NAME

mklost+found — make a lost+found directory for fsck

SYNOPSIS

/etc/mklost+found

DESCRIPTION

A directory *lost+found* is created in the current directory and a number of empty files are created therein and then removed so that there will be empty slots for *fsck*(8). This command should not normally be needed since *mkfs*(8) automatically creates the *lost+found* directory when a new file system is created.

SEE ALSO

fsck(8), mkfs(8)

NAME

mknod — build special file

SYNOPSIS

/etc/mknod name [**c**] [**b**] major minor

DESCRIPTION

Mknod makes a special file. The first argument is the *name* of the entry. The second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g. unit, drive, or line number).

The assignment of major device numbers is specific to each system. They have to be dug out of the system source file *conf.c.*

SEE ALSO

mknod(2)

NAME

mkproto — construct a prototype file system

SYNOPSIS

/etc/mkproto special proto

DESCRIPTION

Mkproto is used to bootstrap a new file system. First a new file system is created using *newfs*(8). *Mkproto* is then used to copy files from the old file system into the new file system according to the directions found in the prototype file *proto.* The prototype file contains tokens separated by spaces or new lines. The first tokens comprise the specification for the root directory. File specifications consist of tokens giving the mode, the user-id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **−bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or **−** to specify set-user-id mode or not. The third is **g** or **−** for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions, see *chmod*(1).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, *mkproto* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

SEE ALSO

fs(5), dir(5), fsck(8), newfs(8)

BUGS

There should be some way to specify links.

There should be some way to specify bad blocks.

Mkproto can only be run on virgin file systems. It should be possible to copy files into existent file systems.

NAME

mount, umount — mount and dismount file system

SYNOPSIS

/etc/mount [special name [**−r**]]

/etc/mount −a

/etc/umount special

/etc/umount −a

DESCRIPTION

Mount announces to the system that a removable file system is present on the device *special.* The file *name* must exist already; it must be a directory (unless the root of the mounted file system is not a directory). It becomes the name of the newly mounted root. The optional argument **−r** indicates that the file system is to be mounted read-only.

Umount announces to the system that the removable file system previously mounted on device *special* is to be removed.

If the **−a** option is present for either *mount* or *umount,* all of the file systems described in */etc/fstab* are attempted to be mounted or unmounted. In this case, *special* and *name* are taken from */etc/fstab.* The *special* file name from */etc/fstab* is the block special name.

These commands maintain a table of mounted devices in */etc/mtab.* If invoked without an argument, *mount* prints the table.

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.

FILES

/etc/mtab	mount table
/etc/fstab	file system table

SEE ALSO

mount(2), mtab(5), fstab(5)

BUGS

Mounting file systems full of garbage will crash the system.
Mounting a root directory on a non-directory makes some apparently good pathnames invalid.

NAME

ncheck − generate names from i-numbers

SYNOPSIS

/etc/ncheck [**−i** numbers] [**−a**] [**−s**] [filesystem]

DESCRIPTION

N.B.: For most normal file system maintenance, the function of *ncheck* is subsumed by *fsck*(8).

Ncheck with no argument generates a pathname vs. i-number list of all files on a set of default file systems. Names of directory files are followed by '/.'. The **−i** option reduces the report to only those files whose i-numbers follow. The **−a** option allows printing of the names '.' and '..', which are ordinarily suppressed. The **−s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

A file system may be specified.

The report is in no useful order, and probably should be sorted.

SEE ALSO

sort(1), dcheck(8), fsck(8), icheck(8)

DIAGNOSTICS

When the filesystem structure is improper, '??' denotes the 'parent' of a parentless file and a pathname beginning with '...' denotes a loop.

NAME

netdirprint – print text version of Pup Network Directory

SYNOPSIS

netdirprint

netdirprint [–ip] [–nic] [dirfile] |grep -v

DESCRIPTION

Netdirprint reads the binary-format Pup Network directory file and reproduces, more or less, a properly formatted text version of this file. (Necessarily, the format is not as useful as that of the original text network directory.)

If the **-ip** flag is given, the output is instead in the format required as input to the BBN host table "compiler". The output must be filtered to remove lines marked "BAD" before further use. ("BAD" lines are those refering to hosts not on the local IP-subnet, or to addresses that do not simple specify a host.)

If the **-nic** flag is given, the output is instead in the NIC's host table format . This output also must be filtered to remove lines marked "BAD" before further use.

Normally, *netdirprint* reads from a default file (the same as is used by *miscserver*(8) and *buildnetdir*(8)). Specifying the *dirfile* argument causes that file to be used, instead.

FILES

/etc/pup/Pup-Network.Dir	Default Pup-Network binary name table.

SEE ALSO

miscserver(8), buildnetdir(8)

AUTHOR

Jeffrey Mogul

BUGS

The program has a built-in idea of the local IP subnet number (Stanford's is 36), and assumes that it is a "class-A" subnet. Probably a bunch of other assumptions are made, as well.

NAME

newfs — construct a new file system

SYNOPSIS

/etc/newfs [−v] [−n] [mkfs-options] special disk-type

DESCRIPTION

Newfs is a "friendly" front-end to the *mkfs*(8) program. *Newfs* will look up the type of disk a file system is being created on in the disk description file */etc/disktab*, calculate the appropriate parameters to use in calling *mkfs*, then build the file system by forking *mkfs* and, if the file system is a root partition, install the necessary bootstrap programs in the initial 8 sectors of the device. The **−n** option prevents the bootstrap programs from being installed.

If the **−v** option is supplied, *newfs* will print out its actions, including the parameters passed to *mkfs*.

Options which may be used to override default parameters passed to *mkfs* are:

−s size The size of the file system in sectors.

−b block-size
The block size of the file system in bytes.

−f frag-size
The fragment size of the file system in bytes.

−t #tracks/cylinder

−c #cylinders/group
The number of cylinders per cylinder group in a file system. The default value used is 16.

−m free space %
The percentage of space reserved from normal users; the minimum free space threshhold. The default value used is 10%.

−r revolutions/minute
The speed of the disk in revolutions per minute (normally 3600).

−S sector-size
The size of a sector in bytes (almost never anything but 512).

−i number of bytes per inode
This specifies the density of inodes in the file system. The default is to create an inode for each 2048 bytes of data space. If fewer inodes are desired, a larger number should be used; to create more inodes a smaller number should be given.

FILES

/etc/disktab	for disk geometry and file system partition information
/etc/mkfs	to actually build the file system
/usr/mdec	for boot strapping programs

SEE ALSO

disktab(5), fs(5), diskpart(8), fsck(8), format(8), mkfs(8), tunefs(8)

McKusick, Joy, Leffler; "A Fast File System for Unix", Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720; TR #7, September 1982.

BUGS

Should figure out the type of the disk without the user's help.

NAME

nu – manage user login accounts (create, modify, destroy Unix accounts)

SYNOPSIS

/etc/nu -a
/etc/nu -m
/etc/nu -d
/etc/nu -k user1 user2 ...

DESCRIPTION

Nu is a program to help a Unix system manager create, modify, delete, and flush accounts on that machine. While everything accomplished by *nu* can be done manually by editing files and issuing shell commands, *nu* will steer you through getting all the details right, worrying about file locking, checking for typos, etc.

When *nu* is run with the "a" option, it adds new accounts. The program prompts you for the login id, password, name, and other information about each new user, and then goes off and creates the account, creates its directories, initializes their contents, and makes an entry in a log file.

When *nu* is run with the "m" option, it modifies existing accounts. It repeatedly asks for account names and instructions for the changes that you want to make to those accounts, until you tell it that you are done making changes. At that time it sorts the updated account records and merges them all at once into /etc/passwd.

When *nu* is run with the "d" option, it allows you to interactively delete accounts. For each account that you specify, *nu* deletes the login directory and all of its contents, and deletes the mailbox. It does *not* delete the entry from /etc/passwd, but it changes the password field so that the user cannot log in. It is a good idea to leave the /etc/passwd entry for a while after an account is deleted, so that accounting information and "lost" files can be related to a user's name.

When *nu* is run with the "k" option and a list of login id's, it deletes from the system almost all information pertaining to those login id's. Specifically, it removes the entry from /etc/passwd, deletes the login directory and all of its contents, and deletes the mailbox. It does not currently remove that user from any mailing lists in /usr/lib/aliases. The "k" option is not interactive: the complete list of accounts to be deleted is provided on the command line after the "–k".

CONFIGURATION

When *nu* is started up, it reads configuration commands from the file /etc/nu.cf. This file specifies the details of how new accounts are to be created on your machine. Typically you will need to change only the GroupHome declarations in that file, which declare the file systems that hold the login directories for members of different groups. However, you can change anything that you find there if your system management policies require it.

When *nu* wants to create a new directory, it runs a shell script named in /etc/nu.cf. Similarly, when it wants to initialize the files in a newly-created directory, it runs another shell script whose name it determines from /etc/nu.cf. By way of configuration and customization, you can edit those shell scripts to conform to local practices. When you do that editing, please remember that *nu* runs as root and that the shell scripts contain statements like "rm –rf *"; it goes without saying that you must be quite cautious. There is a debug mode available, in which *nu* will try not to hurt anything, but whenever you are running as root you should be unusually careful. *Nu* can be run by non-root users if its debug mode is enabled by a "Debug=1" statement in /etc/nu.cf.

CONFIGURATION FILE FORMAT

The configuration file /etc/nu.cf is a text file containing a series of statements, one statement per line. A semicolon that is not inside a quoted string causes the rest of that line to be treated as a comment. Each line in the file that is nonblank after stripping comments is treated as an assignment statement. Each statement assigns a value to one variable. With the exception of the variable "GroupHome", which is special, all of the variables act like ordinary shell variables, which is to say that they can take either integer values or string values. All integers are decimal; all strings must be delimited with double-quotes ("). There is no quoting or doubling convention for putting a double-quote character inside a string.

Here are the configuration variables and what they mean. Case is significant.

Backupfile
: This string variable gives the pathname that *nu* will use to make a backup copy of /etc/passwd, to protect itself from disaster in case something happens while it is writing /etc/passwd. Typical value of Backupfile is "/usr/adm/nu.passwd".

CreateDir
: This string variable identifies the shell script that is run whenever *nu* needs to create a new directory. That shell script must be executable. It is called with 6 arguments: 1, the integer uid; 2, the integer groupid; 3, the name of the user's actual home directory; 4, the name of a symbolic link that should be set up to point to that home directory; 5, an integer that is nonzero iff it is ok to clobber an existing directory of the same name as argument 4; and 6, an integer that is nonzero iff nu is running in debug mode. The standard value for CreateDir is "/etc/nulib/nu1.sh".

CreateFiles
: This string variable identifies the shell script that is run whenever *nu* needs to initialize a directory (newly-created or otherwise) with some standard files. For example, /usr/skel/.[a-z]* are often copied into a new login directory. This shell script must be executable. It is called with 5 arguments: 1, the name of the login directory to be initialized; 2, the integer uid of the user; 3, the integer groupid of the user; 4, an integer that is nonzero iff an MH-format mail-box is to be set up with some initial contents; and 5, an integer that is nonzero iff *nu* is running in debug mode. The standard value for CreateFiles is "/etc/nulib/nu2.sh".

Debug
: This integer variable is set nonzero to cause *nu* to run in debug mode. Debug mode is intended to help you get the bugs out of your shell scripts before you go foolishly running them as root. If Debug is nonzero, then you do not need to be logged on as root to run *nu*. The standard value for Debug is 0.

DefaultGroup
: This integer variable is set to the group number of the default user group. The default is used if the person running *nu* types a carriage return in response to the question asking for a group id for the new user. *nu* requires that a valid GroupHome assignment exist for the default group number. The standard value for DefaultGroup is any group number from /etc/group.

DefaultHome
: This string variable is set to the file system or top-level directory that will be used to hold the login directory for accounts in groups not explicitly set up to have their login directories somewhere else. When you are creating a new account, *nu* asks you what group number you would like the account in. If that group number is mentioned in a GroupHome declaration (see below), then the home directory for the group is the one named in that GroupHome declaration. If the group number is not mentioned in a GroupHome declaration, then login accounts created in that group will have their login directories put into DefaultHome. The standard value for DefaultGroup is "/mnt".

DefaultShell
: This string variable is set to the name of the shell file to use by default. The standard value for DefaultShell is "/bin/csh".

DestroyAccts

This string variable identifies the shell script that is run whenever *nu* needs to destroy a user's account that was created in some earlier session with *nu*. Destroying accounts involves removing the user from the password file, deleting all of his files and directories, and deleting his mailbox. This shell script must be executable. It is called with 5 arguments: 1, the login id of the account to be deleted; 2, the login directory for that account; 3, the name given in /etc/passwd for the login directory (which might possibly be a symbolic link to item 2, above, and therefore needs to be named separately); 4, the name of the log file in which account changes are being logged, and 5, an integer that is nonzero iff *nu* is running in debug mode. The standard value for DestroyAccts is "/etc/nulib/nu3.sh".

DeleteAccts

This string variable identifies the shell script that is run whenever *nu* needs to delete a user's account that was created in some earlier session with *nu*. Deleting accounts involves removing all the user's files and directories, and deleting his mailbox. It should *not* touch /etc/passwd. This shell script must be executable. It is called with 5 arguments: 1, the login id of the account to be deleted; 2, the login directory for that account; 3, the name given in /etc/passwd for the login directory (which might possibly be a symbolic link to item 2, above, and therefore needs to be named separately); 4, the name of the log file in which account changes are being logged, and 5, an integer that is nonzero iff *nu* is running in debug mode. The standard value for DeleteAccts is "/etc/nulib/nu4.sh".

Dummyfile

This string variable holds the name of the hard link that is created as part of the locking process on /etc/passwd; see *vipw(8)*. The correct value for Dummyfile is "/etc/vipw.lock". The only reason that it is specified in the configuration file and not hardwired into the code of *nu* is that in debugging you do not want to muck with the real lock (and might in fact not even have permissions to lock it).

GroupHome

This pseudo-variable is the only name defined in the configuration file that has any trickery attached to it. GroupHome is not really a variable; rather, it is a name by which the configuration code can load entries into a directory location table. In particular, if you provide two GroupHome declarations, they are both processed, while if you provide two of any other declaration, only the latest one has any effect. A typical set of GroupHome declarations might look something like this:

```
GroupHome= 10 "/usr"
GroupHome= 20 "/mnt"
GroupHome= 25 "/usr/cis"
GroupHome= 31 "/usr/guest"
```

The GroupHome declarations serve as default login directory location information for new accounts. You can put any account anywhere you want; the GroupHome information is used to make the defaults come out in the right places, so that the process of creating a new account consists mostly of hitting the return key to accept the defaults. The sample declarations above cause group 10 to default to /usr, i.e. /usr/smith or /usr/jones, and group 31 to default to /usr/guest, i.e. /usr/guest/smith or /usr/guest/jones. If the login group is not mentioned in a GroupHome declaration, then the DefaultHome variable is used. A GroupHome declaration is required for the default group (see variable DefaultGroup); all others are optional.

Linkfile

See also "Dummyfile". This string variable gives the name of the file to which links are made for the purpose of locking the password file. Any value besides "/etc/ptmp" is suspect.

Logfile
: This string variable names the file in which all *nu* transactions are logged. The standard value of Logfile is "/usr/adm/nu.log".

MaxNameLength
: This integer variable gives the maximum number of characters permitted in a login name. For unmodified 4BSD systems it should be set to 8.

PasswdFile
: This string variable gives the name of the file into which *nu* will write its new account entries. Unless you are debugging, its value should be "/etc/passwd".

SymbolicLinkDir
: This string variable gives the name of a directory that can be filled with symbolic links to real login directories. The value of SymbolicLinkDir is ignored unless the variable WantSymbolicLinks is nonzero. See its description, below, for more information. Standard values for SymbolicLinkDir are "/user" or "/udir".

Tempfile
: This string variable names the file that *nu* will use for building a scratch copy of /etc/passwd during the account modification process. The value doesn't really matter much; it is created at the beginning of an *nu* execution and destroyed before exit. A typical value for Tempfile is "/usr/adm/nu.temp".

WantMHsetup
: This integer variable should be set to 1 if you would like *nu* to take care of initializing mailbox contents. Initializing an MH mailbox turns out to be a pleasant way to provide new users with information about the system, and to give them a tutorial on the use of MH. *Nu* just passes the value of WantMHsetup through to the shell script named in CreateFiles, which is responsible for doing the actual initialization. Standard value is 1.

WantSymbolicLinks
: This integer variable controls whether login directory names or symbolic links to them are put in the actual /etc/passwd file. If WantSymbolicLinks is nonzero, then all created accounts are given uniform login directory names in some directory that exists only for the purpose of holding symbolic links, e.g. /user/smith and /user/jones; the file /user/smith or /user/jones is then made to be a symbolic link to the real login directory. This is preferable to the ~smith or ~jones scheme for finding login directories because the ~ notation is not handled by the kernel, but must be handled individually by all programs that open files. If the variable WantSymbolicLinks is 0, then accounts will be created such that the true directory name is stored in /etc/passwd.

SYSTEM ISSUES

Nu obeys the standard locking protocol for /etc/passwd; see *vipw(8)*. It traps INTR characters (e.g. ↑C) and refuses to die if you try to stop it in the middle of a critical section. Critical sections are primarily the updates of /etc/passwd. A list of all changes is recorded in a log file, usually /usr/adm/nu.log.

FILES

/etc/passwd	system password file
/etc/group	system group file
/etc/ptmp	lock file
/etc/vipw.lock	dummy file linked to by /etc/ptmp
/etc/nu.cf	Configuration file

/etc/nulib/*.sh	Shell scripts to perform the work
others	nu.cf and nulib/*.sh reference other files.

SEE ALSO

adduser(8), getgrent(3), getpwent(3), group(5), passwd(5), vipw(8)

AUTHOR

Brian Reid, Erik Hedberg, Fred Yankowski

BUGS

The extensive use of shell scripts for doing sensitive things like purging accounts means that somebody can make *nu* fail in horrible ways without having access to the source code. With increased flexibility comes increased responsibility.

Delete mode takes a long time per account, thus making deletion of many users at one go somewhat painful. It could be arranged to do this much faster, but at the risk of leaving an inconsistency between the /etc/passwd file and the rest of the system.

Since delete mode does not actually remove the user from /etc/passwd, incoming mail for that user will still be accepted. Perhaps *nu* should arrange for such mail to be bounced.

NAME

pac — printer/ploter accounting information

SYNOPSIS

/etc/pac [−Pprinter] [−pprice] [−s] [−r] [−c] [name ...]

DESCRIPTION

Pac reads the printer/plotter accounting files, accumulating the number of pages (the usual case) or feet (for raster devices) of paper consumed by each user, and printing out how much each user consumed in pages or feet and dollars. If any *names* are specified, then statistics are only printed for those users; usually, statistics are printed for every user who has used any paper.

The **−P** flag causes accounting to be done for the named printer. Normally, accounting is done for the default printer (site dependent) or the value of the environment variable **PRINTER** is used.

The **−p** flag causes the value *price* to be used for the cost in dollars instead of the default value of 0.02.

The **−c** flag causes the output to be sorted by cost; usually the output is sorted alphabetically by name.

The **−r** flag reverses the sorting order.

The **−s** flag causes the accounting information to be summarized on the summary accounting file; this summarization is necessary since on a busy system, the accounting file can grow by several lines per day.

FILES

/usr/adm/?acct	raw accounting files
/usr/adm/?_sum	summary accounting files

BUGS

The relationship between the computed price and reality is as yet unknown.

NAME

patchroute – kludge to support Stanford Pup-based subnet routing

SYNOPSIS

patchroute [–v] [–d] [–s sleep-time]

DESCRIPTION

Patchroute gets a copy of the Pup routing table from *gatewayinfo*(8), converts it into an IP subnet routing table, and stuffs it into the kernel.

Except in debug mode, must be ruin by the super-user.

OPTIONS

–d	Debug mode - doesn't change the kernel table, just prints some information.
–v	Verbose mode - tells you what it's doing.
–s time	Sleep mode - instead of exiting, tells *patchroute* to sleep for the specified number of seconds, then run again. *Patchroute* is smart enough not to do too much extra work if the tables don't change. If *time* is not specified, it defaults to 300 seconds (5 minutes).

BUGS

This is a kludge.

Verbose mode is really slow.

Stanford's Class A IP net number is compiled in; this is perhaps a safe assumption, perhaps not.

NAME

pstat – print system facts

SYNOPSIS

/etc/pstat –aixptufkT [suboptions] [system] [corefile]

DESCRIPTION

Pstat interprets the contents of certain system tables. If *corefile* is given, the tables are sought there, otherwise in */dev/kmem. (If corefile* is a core dump, then the **–k** option must be given.) The required namelist is taken from */vmunix* unless *system* is specified. Options are

–a Under **–p**, describe all process slots rather than just active ones.

–i Print the inode table with the these headings:

LOC	The core location of this table entry.
FLAGS	Miscellaneous state variables encoded thus:
	L locked
	U update time (*fs*(5)) must be corrected
	A access time must be corrected
	M file system is mounted here
	W wanted by another process (L flag is on)
	T contains a text file
	C changed time must be corrected
	S shared lock applied
	E exclusive lock applied
	Z someone waiting for an exclusive lock
CNT	Number of open file table entries for this inode.
DEV	Major and minor device number of file system in which this inode resides.
RDC	Reference count of shared locks on the inode.
WRC	Reference count of exclusive locks on the inode (this may be > 1 if, for example, a file descriptor is inherited across a fork).
INO	I-number within the device.
MODE	Mode bits, see *chmod*(2).
NLK	Number of links to this inode.
UID	User ID of owner.
SIZ/DEV	Number of bytes in an ordinary file, or major and minor device of special file.

–x Print the text table with these headings:

LOC	The core location of this table entry.
FLAGS	Miscellaneous state variables encoded thus:
	T *ptrace*(2) in effect
	W text not yet written on swap device
	L loading in progress
	K locked
	w wanted (L flag is on)
	P resulted from demand-page-from-inode exec format (see *execve*(2))
DADDR	Disk address in swap, measured in multiples of 512 bytes.
CADDR	Head of a linked list of loaded processes using this text segment.
SIZE	Size of text segment, measured in multiples of 512 bytes.
IPTR	Core location of corresponding inode.
CNT	Number of processes using this text segment.
CCNT	Number of processes in core using this text segment.

−p	Print process table for active processes with these headings:
LOC	The core location of this table entry.
S	Run state encoded thus:

0	no process
1	waiting for some event
3	runnable
4	being created
5	being terminated
6	stopped under trace

F	Miscellaneous state variables, or-ed together (hexadecimal):

000001	loaded
000002	the scheduler process
000004	locked for swap out
000008	swapped out
000010	traced
000020	used in tracing
000080	in page-wait
000100	prevented from swapping during *fork*(2)
000200	gathering pages for raw i/o
000400	exiting
001000	process resulted from a *vfork*(2) which is not yet complete
002000	another flag for *vfork*(2)
004000	process has no virtual memory, as it is a parent in the context of *vfork*(2)
008000	process is demand paging data pages from its text inode.
010000	process has advised of anomalous behavior with *vadvise*(2).
020000	process has advised of sequential behavior with *vadvise*(2).
040000	process is in a sleep which will timeout.
080000	a parent of this process has exited and this process is now considered detached.
100000	process used 4.1BSD compatibility mode signal primitives, no system calls will restart.
200000	process is owed a profiling tick.

POIP	number of pages currently being pushed out from this process.
PRI	Scheduling priority, see *setpriority*(2).
SIGNAL	Signals received (signals 1-32 coded in bits 0-31),
UID	Real user ID.
SLP	Amount of time process has been blocked.
TIM	Time resident in seconds; times over 127 coded as 127.
CPU	Weighted integral of CPU time, for scheduler.
NI	Nice level, see *setpriority*(2).
PGRP	Process number of root of process group (the opener of the controlling terminal).
PID	The process ID number.
PPID	The process ID of parent process.
ADDR	If in core, the page frame number of the first page of the 'u-area' of the process. If swapped out, the position in the swap area measured in multiples of 512 bytes.
RSS	Resident set size − the number of physical page frames allocated to this process.
SRSS	RSS at last swap (0 if never swapped).
SIZE	Virtual size of process image (data+stack) in multiples of 512 bytes.
WCHAN	Wait channel number of a waiting process.
LINK	Link pointer in list of runnable processes.
TEXTP	If text is pure, pointer to location of text table entry.
CLKT	Countdown for real interval timer, *setitimer*(2) measured in clock ticks (10 milliseconds).
−t	Print table for terminals with these headings:

RAW	Number of characters in raw input queue.
CAN	Number of characters in canonicalized input queue.
OUT	Number of characters in putput queue.
MODE	See *tty*(4).
ADDR	Physical device address.
DEL	Number of delimiters (newlines) in canonicalized input queue.
COL	Calculated column position of terminal.
STATE	Miscellaneous state variables encoded thus:

W	waiting for open to complete
O	open
S	has special (output) start routine
C	carrier is on
B	busy doing output
A	process is awaiting output
X	open for exclusive use
H	hangup on close
S	output stopped by ctrl/s
Q	tandem queue blocked

PGRP	Process group for which this is controlling terminal.
DISC	Line discipline; blank is old tty OTTYDISC or "new tty" for NTTYDISC or "net" for NETLDISC (see *bk*(4)).

−u print information about a user process; the next argument is its address as given by *ps*(1). The process must be in main memory, or the file used can be a core image and the address 0.

−f Print the open file table with these headings:

LOC	The core location of this table entry.
TYPE	The type of object the file table entry points to.
FLG	Miscellaneous state variables encoded thus:

R	open for reading
W	open for writing
A	open for appending

CNT	Number of processes that know this open file.
INO	The location of the inode table entry for this file.
OFFS/SOCK	The file offset (see *lseek*(2)), or the core address of the associated socket structure.

−s print information about swap space usage: the number of (1k byte) pages used and free is given as well as the number of used pages which belong to text images.

−T prints the number of used and free slots in the several system tables and is useful for checking to see how full system tables have become if the system is under heavy load.

FILES

/vmunix	namelist
/dev/kmem	default source of tables

SEE ALSO

ps(1), stat(2), fs(5)
K. Thompson, *UNIX Implementation*

BUGS

It would be very useful if the system recorded "maximum occupancy" on the tables reported by −T; even more useful if these tables were dynamically allocated.

NAME

pup-mailer – deliver mail over the PUP network

SYNOPSIS

/usr/local/lib/pup-mailer *from-address to-host to-user*
/usr/local/lib/pupdaemon *[file [debug]]*

DESCRIPTION

Pup-mailer queues the letter found on its standard input for delivery to the host and user specified. The actual delivery will be performed by the PUP mailer daemon.

If the letter does not appear to have a full ARPANET style header, *pup-mailer* will insert "Date:" and "From:" fields in the proper format. The "From:" person is determined by the from-address argument, with "at <hostname>" appended where the hostname is obtained from <whereami.h>. The from-address argument is also used by the *pupdaemon* to return the mail to you if there is a problem at the receiving host.

Pupdaemon is invoked by the *pup-mailer* with the name of the file containing the message to be sent. It will attempt to make a connection to the given PUP host and send the mail. If the host does not respond or returns a transient error, the message is left in the queue. If the host returns a "no such user" response, that status is returned.

When *Pupdaemon* is invoked without any arguments it attempts to send all the files in the queue. This should be done periodically by an entry in /etc/crontab. Mail that is not sent in two days is returned to the sender.

AUTHOR

Bill Nowicki, based on the Arpanet mailer.

FILES

/usr/spool/pupmail/*

SEE ALSO

delivermail(8) cron(8)

NAME

pup10arpser – Pup GatewayInfo routing table server

SYNOPSIS

/etc/pup/pup10arpser

DESCRIPTION

Pup10arpser is a server for the 10mb Pup Address Resolution Protocol (ARP). Other programs can ask *pup10arpser* to resolve addresses, using IPC requests. For the format of these requests, see the header file <*pup/puparpser.h*>.

Pup10arpser should be run out of */etc/pup/rc* before any other programs.

/etc/pup/pupnettab must be properly configured before *pup10arpser* is run.

FILES

/etc/pup/pupnettab	list of network interfaces

SEE ALSO

pupnettab(9)

AUTHOR

Jeffrey Mogul

NAME

gateway – a Pup gateway program

SYNOPSIS

/etc/pup/gateway [−d] [−t] [−1]

DESCRIPTION

Gateway is a program that turns the host into a Pup gateway between two Ethernets. It must be run out of /etc/pup/rc after *gatewayinfo(8)* (because gatewayinfo is effectively part of the gateway function) and before much of anything else (because it needs to acquire a so-called "high-priority" ethernet minor device on each network; see *enet*(4).)

Gateway must run with the same UID as gatewayinfo; i.e., it must be run as root. This is because it uses Unix signals to inform gatewayinfo that it is up and healthy, and a process can only send signals to another if their UIDs match.

It is alright to run *gateway* out of /etc/pup/rc even if the host may only have one interface; it will exit if there is nothing for it to do.

OPTIONS

−d	Debug; print information about error conditions on stderr.
−t	Trace; print lots of information on stdout.
−1	"One-interface" mode: the gateway will function as a forwarder even if only one interface is present. This is not an "incorrect" thing to do, but it's inefficient and/or unnecessary in almost any conceivable situation.

SEE ALSO

gatewayinfo(8)

AUTHOR

Jeffrey Mogul

BUGS

It doesn't gather any statistics about its operations.

It should be one process with gatewayinfo; splitting this service into two processes is an inefficient use of ethernet minor devices, CPU time and memory, and complicates things. It also makes use of the Xerox "gatecontrol" protocol pretty near impossible. On the other hand, this way was easier for historical reasons.

It's slow to use a Vax like this. (Maybe gatewayinfo should increase hop counts slightly when it send out routing tables, to encourage the use of other gateways.)

NAME

quot − summarize file system ownership

SYNOPSIS

/etc/quot [option] ... [filesystem]

DESCRIPTION

Quot prints the number of blocks in the named *filesystem* currently owned by each user. If no *filesystem* is named, a default name is assumed. The following options are available:

−n Cause the pipeline **ncheck filesystem | sort +0n | quot −n filesystem** to produce a list of all files and their owners.

−c Print three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file.

−f Print count of number of files as well as space owned by each user.

FILES

Default file system varies with system.
/etc/passwd to get user names

SEE ALSO

ls(1), du(1)

NAME

quotacheck − file system quota consistency checker

SYNOPSIS

/etc/quotacheck [**−v**] filesystem...
/etc/quotacheck [**−v**] **−a**

DESCRIPTION

Quotacheck examines each file system, builds a table of current disc usage, and compares this table against that stored in the disc quota file for the file system. If any inconsistencies are detected, both the quota file and the current system copy of the incorrect quotas are updated (the latter only occurs if an active file system is checked).

If the **−a** flag is supplied in place of any file system names, *quotacheck* will check all the file systems indicated in */etc/fstab* to be read-write with disc quotas.

Normally *quotacheck* reports only those quotas modified. If the **−v** option is supplied, *quotacheck* will indicate the calculated disc quotas for each user on a particular file system.

Quotacheck expects each file system to be checked to have a quota file named *quotas* in the root directory. If none is present, *quotacheck* will ignore the file system.

Quotacheck is normally run at boot time from the */etc/rc.local* file, see *rc*(8), before enabling disc quotas with *quotaon*(8).

Quotacheck accesses the raw device in calculating the actual disc usage for each user. Thus, the file systems checked should be quiescent while *quotacheck* is running.

FILES

/etc/fstab	default file systems

SEE ALSO

quota(2), setquota(2), quotaon(8)

NAME

quotaon, quotaoff — turn file system quotas on and off

SYNOPSIS

/etc/quotaon [**−v**] *filsys...*

/etc/quotaon [**−v**] **−a**

/etc/quotaoff [**−v**] *filsys...*

/etc/quotaoff [**−v**] **−a**

DESCRIPTION

Quotaon announces to the system that disc quotas should be enabled on one or more file systems. The file systems specified must have entries in /etc/fstab and be mounted at the time. The file system quota files must be present in the root directory of the specified file system and be named *quotas*. The optional argument **−v** causes *quotaon* to print a message for each file system where quotas are turned on. If, instead of a list of file systems, a **−a** argument is give to *quotaon*, all file systems in /etc/fstab marked read-write with quotas will have their quotas turned on. This is normally used at boot time to enable quotas.

Quotaoff announces to the system that file systems specified should have any disc quotas turned off. As above, the **−v** forces a verbose message for each file system affected; and the **−a** option forces all file systems in /etc/fstab to have their quotas disabled.

These commands update the status field of devices located in */etc/mtab* to indicate when quotas are on or off for each file system.

FILES

/etc/mtab	mount table
/etc/fstab	file system table

SEE ALSO

setquota(2), mtab(5), fstab(5)

NAME

rc — command script for auto-reboot and daemons

SYNOPSIS

/etc/rc
/etc/rc.local

DESCRIPTION

Rc is the command script which controls the automatic reboot and *rc.local* is the script holding commands which are pertinent only to a specific site.

When an automatic reboot is in progress, *rc* is invoked with the argument *autoboot* and runs a *fsck* with option **−p** to "preen" all the disks of minor inconsistencies resulting from the last system shutdown and to check for serious inconsistencies caused by hardware or software failure. If this auto-check and repair succeeds, then the second part of *rc* is run.

The second part of *rc*, which is run after a auto-reboot succeeds and also if *rc* is invoked when a single user shell terminates (see *init*(8)), starts all the daemons on the system, preserves editor files and clears the scratch directory **/tmp.** *Rc.local* is executed immediately before any other commands after a successful *fsck*. Normally, the first commands placed in the *rc.local* file define the machine's name, using *hostname*(1), and save any possible core image that might have been generated as a result of a system crash, *savecore*(8). The latter command is included in the *rc.local* file because the directory in which core dumps are saved is usually site specific.

SEE ALSO

init(8), reboot(8), savecore(8)

BUGS

NAME

rdump − file system dump across the network

SYNOPSIS

/etc/rdump [key [*argument* ...] filesystem]

DESCRIPTION

Rdump copies to magnetic tape all files changed after a certain date in the *filesystem*. The command is identical in operation to *dump*(8) except the *f* key should be specified and the file supplied should be of the form *machine:device*.

Rdump creates a remote server, */etc/rmt*, on the client machine to access the tape device.

SEE ALSO

dump(8), rmt(8C)

DIAGNOSTICS

Same as *dump*(8) with a few extra related to the network.

NAME

reboot — UNIX bootstrapping procedures

SYNOPSIS

/etc/reboot [−n] [−q]

DESCRIPTION

UNIX is started by placing it in memory at location zero and transferring to zero. Since the system is not reenterable, it is necessary to read it in from disk or tape each time it is to be bootstrapped.

Rebooting a running system. When a UNIX is running and a reboot is desired, *shutdown*(8) is normally used. If there are no users then **/etc/reboot** can be used. Reboot causes the disks to be synced, and then a multi-user reboot (as described below) is initiated. This causes a system to be booted and an automatic disk check to be performed. If all this succeeds without incident, the system is then brought up for many users.

Options to reboot are:

−n option avoids the sync. It can be used if a disk or the processor is on fire.

−q reboots quickly and ungracefully, without shutting down running processes first.

Power fail and crash recovery. Normally, the system will reboot itself at power-up or after crashes. Provided the auto-restart is enabled on the machine front panel, an automatic consistency check of the file systems will be performed then and unless this fails the system will resume multi-user operations.

Cold starts. These are processor type dependent. On an 11/780, there are two floppy files for each disk controller, both of which cause boots from unit 0 of the root file system of a controller located on mba0 or uba0. One gives a single user shell, while the other invokes the multi-user automatic reboot. Thus these files are HPS and HPM for the single and multi-user boot from MASSBUS RP06/RM03/RM05 disks, UPS and UPM for UNIBUS storage module controller and disks such as the EMULEX SC-21 and AMPEX 9300 pair, or HKS and HKM for RK07 disks.

Giving the command

```
>>>BOOT HPM
```

Would boot the system from (e.g.) an RP06 and run the automatic consistency check as described in *fsck*(8). (Note that it may be necessary to type control-P to gain the attention of the LSI-11 before getting the >>> prompt.) The command

```
>>>BOOT ANY
```

invokes a version of the boot program in a way which allows you to specify any system as the system to be booted. It reads from the console a device specification (see below) followed immediately by a pathname.

On an 11/750, the reset button will boot from the device selected by the front panel boot device switch. In systems with RK07's, position B normally selects the RK07 for boot. This will boot multi-user. To boot from RK07 with boot flags you may specify

>>>B/*n* DMA0

where, giving a *n* of 1 causes the boot program to ask for the name of the system to be bootstrapped, giving a *n* of 2 causes the boot program to come up single user, and a *n* of 3 causes both of these actions to occur.

The 11/750 boot procedure uses the boot roms to load block 0 off of the specified device. The /usr/mdec directory contains a number of bootstrap programs for the various disks which should be placed in a new pack automatically by *newfs*(8) when the "a" partition file system on the pack is created.

On both processors, the *boot* program finds the corresponding file on the given device, loads that file into memory location zero, and starts the program at the entry address specified in the program header (after clearing off the high bit of the specified entry address.) Normal line editing characters can be used in specifying the pathname.

If you have a MASSBUS disk and wish to boot off of a file system which starts at cylinder 0 of unit 0, you can type "hp(0,0)vmunix" to the boot prompt; "up(0,0)vmunix" would specify a UNIBUS drive, "hk(0,0)vmunix" would specify an RK07 disk drive, "ra(0,0)vmunix" would specify a UDA50 disk drive, and "rb(0,0)vmunix" would specify a disk on a 730 IDC.

A device specification has the following form:

device(unit, minor)

where *device* is the type of the device to be searched, *unit* is 8* the mba or uba number plus the unit number of the device, and *minor* is the minor device index. The following list of supported devices may vary from installation to installation:

hp	MASSBUS disk drive
up	UNIBUS storage module drive
ht	TE16,TU45,TU77 on MASSBUS
mt	TU78 on MASSBUS
hk	RK07 on UNIBUS
ra	storage module on a UDA50
rb	storage module on a 730 IDC
rl	RL02 on UNIBUS
tm	TM11 emulation tape drives on UNIBUS
ts	TS11 on UNIBUS
ut	UNIBUS TU45 emulator

For tapes, the minor device number gives a file offset.

In an emergency, the bootstrap methods described in the paper "Installing and Operating 4.2bsd" can be used to boot from a distribution tape.

FILES

/vmunix	system code
/boot	system bootstrap

SEE ALSO

crash(8V), fsck(8), init(8), rc(8), shutdown(8), halt(8), newfs(8)

NAME

recnews – receive unprocessed articles via mail

SYNOPSIS

/usr/lib/news/recnews [*newsgroup* [*sender*]]

DESCRIPTION

Recnews reads a letter from the standard input; determines the article title, sender, and newsgroup; and gives the body to inews with the right arguments for insertion.

If *newsgroup* is omitted, the to line of the letter will be used. If *sender* is omitted, the sender will be determined from the from line of the letter. The title is determined from the subject line.

SEE ALSO

inews(1), uurec(8), sendnews(8), readnews(1), checknews(1)

NAME

renice − alter priority of running processes

SYNOPSIS

/etc/renice priority [[**−p**] pid ...] [[**−g**] pgrp ...] [[**−u**] user ...]

DESCRIPTION

Renice alters the scheduling priority of one or more running processes. The *who* parameters are interpreted as process ID's, process group ID's, or user names. *Renice*'ing a process group causes all processes in the process group to have their scheduling priority altered. *Renice*'ing a user causes all processes owned by the user to have their scheduling priority altered. By default, the processes to be affected are specified by their process ID's. To force *who* parameters to be interpreted as process group ID's, a **−g** may be specified. To force the *who* parameters to be interpreted as user names, a **−u** may be given. Supplying **−p** will reset *who* interpretation to be (the default) process ID's. For example,

```
/etc/renice +1 987 -u daemon root -p 32
```

would change the priority of process ID's 987 and 32, and all processes owned by users daemon and root.

Users other than the super-user may only alter the priority of processes they own, and can only monotonically increase their "nice value" within the range 0 to PRIO_MIN (20). (This prevents overriding administrative fiats.) The super-user may alter the priority of any process and set the priority to any value in the range PRIO_MAX (−20) to PRIO_MIN. Useful priorities are: 19 (the affected processes will run only when nothing else in the system wants to), 0 (the "base" scheduling priority), anything negative (to make things go very fast).

FILES

/etc/passwd	to map user names to user ID's

SEE ALSO

getpriority(2), setpriority(2)

BUGS

If you make the priority very negative, then the process cannot be interrupted. To regain control you make the priority greater than zero. Non super-users can not increase scheduling priorities of their own processes, even if they were the ones that decreased the priorities in the first place.

NAME

repquota − summarize quotas for a file system

SYNOPSIS

repquota *filesys*...

DESCRIPTION

Repquota prints a summary of the disc usage and quotas for the specified file systems. For each user the current number files and amount of space (in kilobytes) is printed, along with any quotas created with *edquota*(8).

Only the super-user may view quotas which are not their own.

FILES

quotas	at the root of each file system with quotas
/etc/fstab	for file system names and locations

SEE ALSO

quota(1), quota(2), quotacheck(8), quotaon(8), edquota(8)

DIAGNOSTICS

Various messages about inaccessible files; self-explanatory.

NAME

restore — incremental file system restore

SYNOPSIS

/etc/restore key [name ...]

DESCRIPTION

Restore reads tapes dumped with the *dump*(8) command. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying the files that are to be restored. Unless the **h** key is specified (see below), the appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The tape is read and loaded into the current directory. This should not be done lightly; the **r** key should only be used to restore a complete dump tape onto a clear file system or to restore an incremental dump tape after a full level zero restore. Thus

```
/etc/newfs /dev/rrp0g eagle
/etc/mount /dev/rp0g /mnt
cd /mnt
restore r
```

is a typical sequence to restore a complete dump. Another *restore* can be done to get an incremental dump in on top of this.
A *dump*(8) followed by a *newfs*(8) and a *restore* is used to change the size of a file system.

R *Restore* requests a particular tape of a multi volume set on which to restart a full restore (see the **r** key above). This allows *restore* to be interrupted and then restarted.

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, and the **h** key is not specified, the directory is recursively extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, then the root directory is extracted, which results in the entire content of the tape being extracted, unless the **h** key has been specified.

t The names of the specified files are listed if they occur on the tape. If no file argument is given, then the root directory is listed, which results in the entire content of the tape being listed, unless the **h** key has been specified. Note that the **t** key replaces the function of the old *dumpdir* program.

i This mode allows interactive restoration of files from a dump tape. After reading in the directory information from the tape, *restore* provides a shell like interface that allows the user to move around the directory tree selecting files to be extracted. The available commands are given below; for those commands that require an argument, the default is the current directory.

ls [arg] — List the current or specified directory. Entries that are directories are appended with a "/". Entries that have been marked for extraction are prepended with a "*". If the verbose key is set the inode number of each entry is also listed.

cd arg — Change the current working directory to the specified argument.

pwd — Print the full pathname of the current working directory.

add [arg] — The current directory or specified argument is added to the list of files to be extracted. If a directory is specified, then it and all its descendents are added to the extraction list (unless the **h** key is specified on the command line). Files that are on the extraction list are prepended with a "*" when they are listed by **ls**.

delete [arg] — The current directory or specified argument is deleted from the list of files to be extracted. If a directory is specified, then it and all its descendents are deleted from the extraction list (unless the **h** key is specified on the command line). The most expedient way to extract most of the files from a directory is to add the directory to the extraction list and then delete those files that are not needed.

extract — All the files that are on the extraction list are extracted from the dump tape. *Restore* will ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

verbose — The sense of the **v** key is toggled. When set, the verbose key causes the **ls** command to list the inode numbers of all entries. It also causes *restore* to print out information about each file as it is extracted.

help — List a summary of the available commands.

quit — Restore immediately exits, even if the extraction list is not empty.

The following characters may be used in addition to the letter that selects the function desired.

v Normally *restore* does its work silently. The **v** (verbose) key causes it to type the name of each file it treats preceded by its file type.

f The next argument to *restore* is used as the name of the archive instead of /dev/rmt?. If the name of the file is "—", *restore* reads from standard input. Thus, *dump*(8) and *restore* can be used in a pipeline to dump and restore a file system with the command

```
dump 0f - /usr | (cd /mnt; restore xf -)
```

y *Restore* will not ask whether it should abort the restore if gets a tape error. It will always try to skip over the bad tape block(s) and continue as best it can.

m *Restore* will extract by inode numbers rather than by file name. This is useful if only a few files are being extracted, and one wants to avoid regenerating the complete pathname to the file.

h *Restore* extracts the actual directory, rather than the files that it references. This prevents hierarchical restoration of complete subtrees from the tape.

DIAGNOSTICS

Complaints about bad key characters.

Complaints if it gets a read error. If **y** has been specified, or the user responds "y", *restore* will attempt to continue the restore.

If the dump extends over more than one tape, *restore* will ask the user to change tapes. If the **x** or **i** key has been specified, *restore* will also ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

There are numerous consistency checks that can be listed by *restore*. Most checks are self-explanatory or can "never happen". Common errors are given below.

Converting to new file system format.
A dump tape created from the old file system has been loaded. It is automatically

converted to the new file system format.

<filename>: not found on tape
: The specified file name was listed in the tape directory, but was not found on the tape. This is caused by tape read errors while looking for the file, and from using a dump tape created on an active file system.

expected next file <inumber>, got <inumber>
: A file that was not listed in the directory showed up. This can occur when using a dump tape created on an active file system.

Incremental tape too low
: When doing incremental restore, a tape that was written before the previous incremental tape, or that has too low an incremental level has been loaded.

Incremental tape too high
: When doing incremental restore, a tape that does not begin its coverage where the previous incremental tape left off, or that has too high an incremental level has been loaded.

Tape read error while restoring <filename>
Tape read error while skipping over inode <inumber>
Tape read error while trying to resynchronize
: A tape read error has occurred. If a file name is specified, then its contents are probably partially wrong. If an inode is being skipped or the tape is trying to resynchronize, then no extracted files have been corrupted, though files may not be found on the tape.

resync restore, skipped <num> blocks
: After a tape read error, *restore* may have to resynchronize itself. This message lists the number of blocks that were skipped over.

FILES

/dev/rmt?	the default tape drive
/tmp/rstdir*	file containing directories on the tape.
/tmp/rstmode*	owner, mode, and time stamps for directories.
./restoresymtab	information passed between incremental restores.

SEE ALSO

rrestore(8C) dump(8), newfs(8), mount(8), mkfs(8)

BUGS

Restore can get confused when doing incremental restores from dump tapes that were made on active file systems.

A level zero dump must be done after a full restore. Because restore runs in user code, it has no control over inode allocation; thus a full restore must be done to get a new set of directories reflecting the new inode numbering, even though the contents of the files is unchanged.

NAME

rexecd − remote execution server

SYNOPSIS

/etc/rexecd

DESCRIPTION

Rexecd is the server for the *rexec*(3X) routine. The server provides remote execution facilities with authentication based on user names and encrypted passwords.

Rexecd listens for service requests at the port indicated in the "exec" service specification; see *services*(5). When a service request is received the following protocol is initiated:

1) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine.

3) A null terminated user name of at most 16 characters is retrieved on the initial socket.

4) A null terminated, encrypted, password of at most 16 characters is retrieved on the initial socket.

5) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

6) *Rexecd* then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.

7) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rexecd.*

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

"username too long"
The name is longer than 16 characters.

"password too long"
The password is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"Password incorrect."
The wrong was password supplied.

"No remote directory."
The *chdir* command to the home directory failed.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

BUGS

Indicating "Login incorrect" as opposed to "Password incorrect" is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rlogind − remote login server

SYNOPSIS

/etc/rlogind [−d]

DESCRIPTION

Rlogind is the server for the *rlogin*(1C) program. The server provides a remote login facility with authentication based on privileged port numbers.

Rlogind listens for service requests at the port indicated in the "login" service specification; see *services*(5). When a service request is received the following protocol is initiated:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection.

2) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see *hosts*(5)), the server aborts the connection.

Once the source port and address have been checked, *rlogind* allocates a pseudo terminal (see *pty*(4)), and manipulates file descriptors so that the slave half of the pseudo terminal becomes the **stdin** , **stdout** , and **stderr** for a login process. The login process is an instance of the *login*(1) program, invoked with the **−r** option. The login process then proceeds with the authentication process as described in *rshd*(8C), but if automatic authentication fails, it reprompts the user to login as one finds on a standard terminal line.

The parent of the login process manipulates the master side of the pseduo terminal, operating as an intermediary between the login process and the client instance of the *rlogin* program. In normal operation, the packet protocol described in *pty*(4) is invoked to provide ^S/^Q type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, "TERM"; see *environ*(7).

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1.

"Hostname for your address unknown."
No entry in the host name database existed for the client's machine.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rmt − remote magtape protocol module

SYNOPSIS

/etc/rmt

DESCRIPTION

Rmt is a program used by the remote dump and restore programs in manipulating a magnetic tape drive through an interprocess communication connection. *Rmt* is normally started up with an *rexec*(3X) or *rcmd*(3X) call.

The *rmt* program accepts requests specific to the manipulation of magnetic tapes, performs the commands, then responds with a status indication. All responses are in ASCII and in one of two forms. Successful commands have responses of

A*number*\n

where *number* is an ASCII representation of a decimal number. Unsuccessful commands are responded to with

E*error-number*\n*error-message*\n,

where *error-number* is one of the possible error numbers described in *intro*(2) and *error-message* is the corresponding error string as printed from a call to *perror*(3). The protocol is comprised of the following commands (a space is present between each token).

O device mode Open the specified *device* using the indicated *mode*. *Device* is a full pathname and *mode* is an ASCII representation of a decimal number suitable for passing to *open*(2). If a device had already been opened, it is closed before a new open is performed.

C device Close the currently open device. The *device* specified is ignored.

L whence offset Perform an *lseek*(2) operation using the specified parameters. The response value is that returned from the *lseek* call.

W count Write data onto the open device. *Rmt* reads *count* bytes from the connection, aborting if a premature end-of-file is encountered. The response value is that returned from the *write*(2) call.

R count Read *count* bytes of data from the open device. If *count* exceeds the size of the data buffer (10 kilobytes), it is truncated to the data buffer size. *Rmt* then performs the requested *read*(2) and responds with A*count-read*\n if the read was successful; otherwise an error in the standard format is returned. If the read was successful, the data read is then sent.

I operation count
Perform a MTIOCOP *ioctl*(2) command using the specified parameters. The parameters are interpreted as the ASCII representations of the decimal values to place in the *mt_op* and *mt_count* fields of the structure used in the *ioctl* call. The return value is the *count* parameter when the operation is successful.

S Return the status of the open device, as obtained with a MTIOCGET *ioctl* call. If the operation was successful, an "ack" is sent with the size of the status buffer, then the status buffer is sent (in binary).

Any other command causes *rmt* to exit.

DIAGNOSTICS

All responses are of the form described above.

SEE ALSO

rcmd(3X), rexec(3X), mtio(4), rdump(8C), rrestore(8C)

BUGS

People tempted to use this for a remote file access protocol are discouraged.

NAME

route − manually manipulate the routing tables

SYNOPSIS

/etc/route [−n] [−f] [*command args* **]**

DESCRIPTION

Route is a program used to manually manipulate the network routing tables. It normally is not needed, as the system routing table management daemon, *routed*(8C), should tend to this task.

Route accepts three commands: *add*, to add a route; *delete*, to delete a route; and *change*, to modify an existing route.

All commands have the following syntax:

/etc/route *command* **destination gateway [metric]**

where *destination* is a host or network for which the route is "to", *gateway* is the gateway to which packets should be addressed, and *metric* is an optional count indicating the number of hops to the *destination*. If no metric is specified, *route* assumes a value of 0. Routes to a particular host are distinguished from those to a network by a metric of "host"; e.g.,
/etc/route add sri-nic cmu-gateway host
If the route is to a destination connected via a gateway, the *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first in the host name database, *hosts*(5). If this lookup fails, the name is then looked for in the network name database, *networks*(5).

Route uses a raw socket and the SIOCADDRT and SIOCDELRT *ioctl*'s to do its work. As such, only the super-user may modify the routing tables.

If the **−f** option is specified, *route* will "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

If the **−n** option is specified, *route* will not print the names of hosts, networks, or gateways in its diagnostic messages. This makes it run much faster.

DIAGNOSTICS

"add %s: gateway %s flags %x"
The specified route is being added to the tables. The values printed are from the routing table entry supplied in the *ioctl* call.

"delete %s: gateway %s flags %x"
As above, but when deleting an entry.

"%s %s done"
When the **−f** flag is specified, each routing table entry deleted is indicated with a message of this form.

"not in table"
A delete operation was attempted for an entry which wasn't present in the tables.

"routing table overflow"
An add operation was attempted, but the system was low on resources and was unable to allocate memory to create the new entry.

SEE ALSO

intro(4N), routed(8C)

BUGS

The change operation is not implemented, one should add the new route, then delete the old one.

NAME

sendnews − send news articles via mail

SYNOPSIS

sendnews [**−o**] [**−a**] [**−b**] [**−n** newsgroups] destination

DESCRIPTION

sendnews reads an article from it's standard input, performs a set of changes to it, and gives it to the mail program to mail it to *destination.*

An 'N' is prepended to each line for decoding by *uurec(1).*

The **−o** flag handles old format articles.

The **−a** flag is used for sending articles via the ARPANET. It maps the article's path from *uucphost!xxx* to *xxx@arpahost.*

The **−b** flag is used for sending articles via the **Berknet.** It maps the article's path from *uucphost!xxx* to *berkhost:xxx.*

The **−n** flag changes the article's newsgroup to the specified *newsgroup.*

SEE ALSO

inews(1), uurec(8), recnews(8), readnews(1), checknews(1)

NAME

routed — network routing daemon

SYNOPSIS

/etc/routed [−s] [−q] [−t] [*logfile*]

DESCRIPTION

Routed is invoked at boot time to manage the network routing tables. The routing daemon uses a variant of the Xerox NS Routing Information Protocol in maintaining up to date kernel routing table entries.

In normal operation *routed* listens on *udp*(4P) socket 520 (decimal) for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When *routed* is started, it uses the SIOCGIFCONF *ioctl* to find those directly connected interfaces configured into the system and marked "up" (the software loopback interface is ignored). If multiple interfaces are present, it is assumed the host will forward packets between networks. *Routed* then transmits a *request* packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, *routed* formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a "hop count" metric (a count of 16, or greater, is considered "infinite"). The metric associated with each route returned provides a metric *relative to the sender*.

Response packets received by *routed* are used to update the routing tables if one of the following conditions is satisfied:

(1) No routing table entry exists for the destination network or host, and the metric indicates the destination is "reachable" (i.e. the hop count is not infinite).

(2) The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.

(3) The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.

(4) The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, *routed* records the change in its internal tables and generates a *response* packet to all directly connected hosts and networks. *Routed* waits a short period of time (no more than 30 seconds) before modifying the kernel's routing tables to allow possible unstable situations to settle.

In addition to processing incoming packets, *routed* also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to insure the invalidation is propagated throughout the internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks.

Supplying the **−s** option forces *routed* to supply routing information whether it is acting as an internetwork router or not. The **−q** option is the opposite of the **−s** option. If the **−t** option is specified, all packets sent or received are printed on the standard output. In addition, *routed* will not divorce itself from the controlling terminal so that interrupts from the keyboard will kill the process. Any other argument supplied is interpreted as the name of file in which *routed*'s actions should be logged. This log contains information about any changes to the routing tables and a history of recent messages sent and received which are related to the changed route.

In addition to the facilities described above, *routed* supports the notion of "distant" *passive* and *active* gateways. When *routed* is started up, it reads the file */etc/gateways* to find gateways which may not be identified using the SIOGIFCONF *ioctl*. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (i.e. they should have a *routed* process running on the machine). Passive gateways are maintained in the routing tables forever and information regarding their existence is included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of the time, the associated route is deleted.

The */etc/gateways* is comprised of a series of lines, each in the following format:

< net | host > *name1* **gateway** *name2* **metric** *value* **< passive | active >**

The **net** or **host** keyword indicates if the route is to a network or specific host.

Name1 is the name of the destination network or host. This may be a symbolic name located in */etc/networks* or */etc/hosts*, or an Internet address specified in "dot" notation; see *inet*(3N).

Name2 is the name or address of the gateway to which messages should be forwarded.

Value is a metric indicating the hop count to the destination host or network.

The keyword **passive** or **active** indicates if the gateway should be treated as *passive* or *active* (as described above).

FILES

/etc/gateways for distant gateways

SEE ALSO

"Internet Transport Protocols", XSIS 028112, Xerox System Integration Standard.
udp(4P)

BUGS

The kernel's routing tables may not correspond to those of *routed* for short periods of time while processes utilizing existing routes exit; the only remedy for this is to place the routing process in the kernel.

Routed should listen to intelligent interfaces, such as an IMP, and to error protocols, such as ICMP, to gather more information.

NAME
rrestore − restore a file system dump across the network

SYNOPSIS
/etc/rrestore [key [name ...]

DESCRIPTION
Rrestore obtains from magnetic tape files saved by a previous *dump*(8). The command is identical in operation to *restore*(8) except the *f* key should be specified and the file supplied should be of the form *machine:device.*

Rrestore creates a remote server, */etc/rmt*, on the client machine to access the tape device.

SEE ALSO
restore(8), rmt(8C)

DIAGNOSTICS
Same as *restore*(8) with a few extra related to the network.

BUGS

NAME

rshd – remote shell server

SYNOPSIS

/etc/rshd

DESCRIPTION

Rshd is the server for the *rcmd*(3X) routine and, consequently, for the *rsh*(1C) program. The server provides remote execution facilities with authentication based on privileged port numbers.

Rshd listens for service requests at the port indicated in the "cmd" service specification; see *services*(5). When a service request is received the following protocol is initiated:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection.

2) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.

3) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 0-1023.

4) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see *hosts*(5)), the server aborts the connection.

5) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the **server**'s machine.

6) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the **client**'s machine.

7) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

8) *Rshd* then validates the user according to the following steps. The remote user name is looked up in the password file and a *chdir* is performed to the user's home directory. If either the lookup or *chdir* fail, the connection is terminated. If the user is not the super-user, (user id 0), the file */etc/hosts.equiv* is consulted for a list of hosts considered "equivalent". If the client's host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the super-user, then the file *.rhosts* in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. If this lookup fails, the connection is terminated.

9) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rshd.*

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the command execution).

"locuser too long"
The name of the user on the client's machine is longer than 16 characters.

"remuser too long"
The name of the user on the remote machine is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Hostname for your address unknown."
No entry in the host name database existed for the client's machine.

"Login incorrect."
No password file entry for the user name existed.

"No remote directory."
The *chdir* command to the home directory failed.

"Permission denied."
The authentication procedure described above failed.

"Can't make pipe."
The pipe needed for the **stderr**, wasn't created.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

SEE ALSO

rsh(1C), rcmd(3X)

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rwhod — system status server

SYNOPSIS

/etc/rwhod

DESCRIPTION

Rwhod is the server which maintains the database used by the *rwho*(1C) and *ruptime*(1C) programs. Its operation is predicated on the ability to *broadcast* messages on a network.

Rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other *rwhod* servers' status messages, validating them, then recording them in a collection of files located in the directory */usr/spool/rwho*.

The *rwho* server transmits and receives messages at the port indicated in the "rwho" service specification, see *services*(5). The messages sent and received, are of the form:

```
struct  outmp {
        char    out_line[8];/* tty name */
        char    out_name[8];/* user id */
        long    out_time;/* time on */
};

struct  whod {
        char    wd_vers;
        char    wd_type;
        char    wd_fill[2];
        int     wd_sendtime;
        int     wd_recvtime;
        char    wd_hostname[32];
        int     wd_loadav[3];
        int     wd_boottime;
        struct  whoent {
                struct  outmp we_utmp;
                int     we_idle;
        } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the *w*(1) program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission. The host name included is that returned by the *gethostname*(2) system call. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the *utmp*(5) entry for each non-idle terminal line and a value indicating the time since a character was last received on the terminal line.

Messages received by the *rwho* server are discarded unless they originated at a *rwho* server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by *rwhod* are placed in files named *whod.hostname* in the directory */usr/spool/rwho*. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 60 seconds. *Rwhod* performs an *nlist*(3) on /vmunix every 10 minutes to guard against the possibility that this file is not the system image currently operating.

SEE ALSO

rwho(1C), ruptime(1C)

BUGS

Should relay status information between networks. People often interpret the server dieing as a machine going down.

NAME

rxformat - format floppy disks

SYNOPSIS

/etc/rxformat [−d] special

DESCRIPTION

The *rxformat* program formats a diskette in the specified drive associated with the special device *special.* (*Special* is normally /dev/rrx0, for drive 0, or /dev/rrx1, for drive 1.) By default, the diskette is formatted single density; a **−d** flag may be supplied to force double density formatting. Single density is compatible with the IBM 3740 standard (128 bytes/sector). In double density, each sector contains 256 bytes of data.

Before formatting a diskette *rxformat* prompts for verification (this allows a user to cleanly abort the operation; note that formatting a diskette will destroy any existing data). Formatting is done by the hardware. All sectors are zero-filled.

DIAGNOSTICS

'No such device' means that the drive is not ready, usually because no disk is in the drive or the drive door is open. Other error messages are selfexplanatory.

FILES

/dev/rx?

SEE ALSO

rx(4V)

BUGS

A floppy may not be formatted if the header info on sector 1, track 0 has been damaged. Hence, it is not possible to format a completely degaussed disk. (This is actually a problem in the hardware.)

NAME

sa, accton − system accounting

SYNOPSIS

/etc/sa [−abcdDfijkKlnrstuv] [file]

/etc/accton [file]

DESCRIPTION

With an argument naming an existing *file, accton* causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

Sa reports on, cleans up, and generally maintains accounting files.

Sa is able to condense the information in */usr/adm/acct* into a summary file */usr/adm/savacct* which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system */usr/adm/acct* can grow by 100 blocks per day. The summary file is normally read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; */usr/adm/acct* is the default.

Output fields are labeled: "cpu" for the sum of user+system time (in minutes), "re" for real time (also in minutes), "k" for cpu-time averaged core usage (in 1k units), "avio" for average number of i/o operations per execution. With options fields labeled "tio" for total i/o operations, "k*sec" for cpu storage integral (kilo-core seconds), "u" and "s" for user and system cpu time alone (both in minutes) will sometimes appear.

There are near a googol of options:

a	Place all command names containing unprintable characters and those used only once under the name '***other.'
b	Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.
c	Besides total user, system, and real time for each command print percentage of total time over all commands.
d	Sort by average number of disk i/o operations.
D	Print and sort by total number of disk i/o operations.
f	Force no interactive threshold compression with −v flag.
i	Don't read in summary file.
j	Instead of total minutes time for each category, give seconds per call.
k	Sort by cpu-time average memory usage.
K	Print and sort by cpu-storage integral.
l	Separate system and user time; normally they are combined.
m	Print number of processes and number of CPU minutes for each user.
n	Sort by number of calls.
r	Reverse order of sort.
s	Merge accounting file into summary file */usr/adm/savacct* when done.
t	For each command report ratio of real time to the sum of user and system times.
u	Superseding all other flags, print for each command in the accounting file the user ID

and command name.

v	Followed by a number *n*, types the name of each command used *n* times or fewer. Await a reply from the terminal; if it begins with 'y', add the command to the category '**junk**.' This is used to strip out garbage.

FILES

/usr/adm/acct	raw accounting
/usr/adm/savacct	summary
/usr/adm/usracct	per-user summary

SEE ALSO

ac(8), acct(2)

BUGS

The number of options to this program is absurd.

NAME

savecore — save a core dump of the operating system

SYNOPSIS

/etc/savecore *dirname* [*system*]

DESCRIPTION

Savecore is meant to be called near the end of the /etc/rc file. Its function is to save the core dump of the system (assuming one was made) and to write a reboot message in the shutdown log.

Savecore checks the core dump to be certain it corresponds with the current running unix. If it does it saves the core image in the file *dirname*/vmcore.n and it's brother, the namelist, *dirname*/vmunix.n The trailing ".n" in the pathnames is replaced by a number which grows every time *savecore* is run in that directory.

Before savecore writes out a core image, it reads a number from the file *dirname*/minfree. If there are fewer free blocks on the filesystem which contains *dirname* than the number obtained from the minfree file, the core dump is not done. If the minfree file does not exist, savecore always writes out the core file (assuming that a core dump was taken).

Savecore also writes a reboot message in the shut down log. If the system crashed as a result of a panic, *savecore* records the panic string in the shut down log too.

If the core dump was from a system other than /vmunix, the name of that system must be supplied as *sysname*.

FILES

/usr/adm/shutdownlog	shut down log
/vmunix	current UNIX

BUGS

Can be fooled into thinking a core dump is the wrong size.

NAME

sendmail − send mail over the internet

SYNOPSIS

/usr/lib/sendmail [flags] [address ...]

newaliases

mailq

DESCRIPTION

Sendmail sends a message to one or more people, routing the message over whatever networks are necessary. *Sendmail* does internetwork forwarding as necessary to deliver the message to the correct place.

Sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; *sendmail* is used only to deliver pre-formatted messages.

With no flags, *sendmail* reads its standard input up to a control-D or a line with a single dot and sends a copy of the letter found there to all of the addresses listed. It determines the network to use based on the syntax and contents of the addresses.

Local addresses are looked up in a file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in any alias expansions, e.g., if 'john' sends to 'group', and 'group' includes 'john' in the expansion, then the letter will not be delivered to 'john'.

Flags are:

−ba	Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the "From:" and "Sender:" fields are examined for the name of the sender.
−bd	Run as a daemon. This requires Berkeley IPC.
−bi	Initialize the alias database.
−bm	Deliver mail in the usual way (default).
−bp	Print a listing of the queue.
−bs	Use the SMTP protocol as described in RFC821. This flag implies all the operations of the **−ba** flag that are compatible with SMTP.
−bt	Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.
−bv	Verify names only − do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.
−bz	Create the configuration freeze file.
−C*file*	Use alternate configuration file.
−d*X*	Set debugging value to *X*.
−F*fullname*	Set the full name of the sender.
−f*name*	Sets the name of the "from" person (i.e., the sender of the mail). **−f** can only be used by the special users *root, daemon,* and *network,* or if the person you are trying to become is the same as the person you are.
−h*N*	Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop.
−n	Don't do aliasing.

−o*x value*	Set option *x* to the specified *value.* Options are described below.
−q[*time*]	Processed saved messages in the queue at given intervals. If is omitted, process the queue once. is given as a tagged number, with 's' being seconds, 'm' being minutes, 'h' being hours, 'd' being days, and 'w' being weeks. For example, "−q1h30m" or "−q90m" would both set the timeout to one hour thirty minutes.
−r*name*	An alternate and obsolete form of the **−f** flag.
−t	Read message for recipients. To:, Cc:, and Bcc: lines will be scanned for people to send to. The Bcc: line will be deleted before transmission. Any addresses in the argument list will be suppressed.
−v	Go into verbose mode. Alias expansions will be announced, etc.

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **−o** flag or in the configuration file. These are described in detail in the *Installation and Operation Guide.* The options are:

A*file*	Use alternate alias file.
c	On mailers that are considered "expensive" to connect to, don't initiate immediate connection. This requires queueing.
d*x*	Set the delivery mode to *x.* Delivery modes are 'i' for interactive (synchronous) delivery, 'b' for background (asynchronous) delivery, and 'q' for queue only − i.e., actual delivery is done the next time the queue is run.
D	Try to automatically rebuild the alias database if necessary.
e*x*	Set error processing to mode *x.* Valid modes are 'm' to mail back the error message, 'w' to "write" back the error message (or mail it back if the sender is not logged in), 'p' to print the errors on the terminal (default), 'q' to throw away error messages (only exit status is returned), and 'e' to do special processing for the BerkNet. If the text of the message is not mailed back by modes 'm' or 'w' and if the sender is local to this machine, a copy of the message is appended to the file "dead.letter" in the sender's home directory.
F*mode*	The mode to use when creating temporary files.
f	Save UNIX-style From lines at the front of messages.
g*N*	The default group id to use when calling mailers.
H*file*	The SMTP help file.
i	Do not take dots on a line by themselves as a message terminator.
L*n*	The log level.
m	Send to "me" (the sender) also if I am in an alias expansion.
o	If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (i.e., commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.
Q*queuedir*	Select the directory in which to queue messages.
r*timeout*	The timeout on reads; if none is set, *sendmail* will wait forever for a mailer.
S*file*	Save statistics in the named file.

s	Always instantiate the queue file, even under circumstances where it is not strictly necessary.
T*time*	Set the timeout on messages in the queue to the specified time. After sitting in the queue for this amount of time, they will be returned to the sender. The default is three days.
t*stz,dtz*	Set the name of the time zone.
u*N*	Set the default user id for mailers.

If the first character of the user name is a vertical bar, the rest of the user name is used as the name of a program to pipe the mail to. It may be necessary to quote the name of the user to keep *sendmail* from suppressing the blanks from between arguments.

Sendmail returns an exit status describing what it did. The codes are defined in <*sysexits.h*>

EX_OK	Successful completion on all addresses.
EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as "cannot fork".
EX_NOHOST	Host name not recognized.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

If invoked as *newaliases, sendmail* will rebuild the alias database. If invoked as *mailq, sendmail* will print the contents of the mail queue.

FILES

Except for /usr/lib/sendmail.cf, these pathnames are all specified in /usr/lib/sendmail.cf. Thus, these values are only approximations.

/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.pag	
/usr/lib/aliases.dir	data base of alias names
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	help file
/usr/lib/sendmail.st	collected statistics
/usr/bin/uux	to deliver uucp mail
/usr/net/bin/v6mail	to deliver local mail
/usr/net/bin/sendberkmail	to deliver Berknet mail
/usr/lib/mailers/arpa	to deliver ARPANET mail
/usr/spool/mqueue/*	temp files

SEE ALSO

biff(1), binmail(1), mail(1), aliases(5), sendmail.cf(5), rmail(1), mailaddr(7);
DARPA Internet Request For Comments RFC819, RFC821, RFC822;
Sendmail — An Internetwork Mail Router;
Sendmail Installation and Operation Guide.

BUGS

Sendmail converts blanks in addresses to dots. This is incorrect according to the old ARPANET mail protocol RFC733 (NIC 41952), but is consistent with the new protocols (RFC822).

NAME

sendnews – send news articles via mail

SYNOPSIS

sendnews [**–o**] [**–a**] [**–b**] [**–n** newsgroups] destination

DESCRIPTION

sendnews reads an article from it's standard input, performs a set of changes to it, and gives it to the mail program to mail it to *destination.*

An 'N' is prepended to each line for decoding by *uurec(1).*

The **–o** flag handles old format articles.

The **–a** flag is used for sending articles via the **ARPANET**. It maps the article's path from *uucphost!xxx* to *xxx@arpahost.*

The **–b** flag is used for sending articles via the **Berknet.** It maps the article's path from *uucphost!xxx* to *berkhost:xxx.*

The **–n** flag changes the article's newsgroup to the specified *newsgroup.*

SEE ALSO

inews(1), uurec(8), recnews(8), readnews(1), checknews(1)

NAME

shutdown − close down the system at a given time

SYNOPSIS

/etc/shutdown [**−k**] [**−r**] [**−h**] time [warning-message ...]

DESCRIPTION

Shutdown provides an automated shutdown procedure which a super-user can use to notify users nicely when the system is shutting down, saving them from system administrators, hackers, and gurus, who would otherwise not bother with niceties.

Time is the time at which *shutdown* will bring the system down and may be the word **now** (indicating an immediate shutdown) or specify a future time in one of two formats: +number and hour:min. The first form brings the system down in *number* minutes and the second brings the system down at the time of day indicated (as a 24−hour clock).

At intervals which get closer together as apocalypse approaches, warning messages are displayed at the terminals of all users on the system. Five minutes before shutdown, or immediately if shutdown is in less than 5 minutes, logins are disabled by creating /etc/nologin and writing a message there. If this file exists when a user attempts to log in, *login*(1) prints its contents and exits. The file is removed just before *shutdown* exits.

At shutdown time a message is written in the file /usr/adm/shutdownlog, containing the time of shutdown, who ran shutdown and the reason. Then a terminate signal is sent at *init* to bring the system down to single-user state. Alternatively, if **−r**, **−h**, or **−k** was used, then *shutdown* will exec *reboot*(8), *halt*(8), or avoid shutting the system down (respectively). (If it isn't obvious, **−k** is to make people *think* the system is going down!)

The time of the shutdown and the warning message are placed in /etc/nologin and should be used to inform the users about when the system will be back up and why it is going down (or anything else).

FILES

/etc/nologin	tells login not to let anyone log in
/usr/adm/shutdownlog	log file for successful shutdowns.

SEE ALSO

login(1), reboot(8)

BUGS

Only allows you to kill the system between now and 23:59 if you use the absolute time for shutdown.

NAME

sticky − executable files with persistent text

DESCRIPTION

While the 'sticky bit', mode 01000 (see *chmod*(2)), is set on a sharable executable file, the text of that file will not be removed from the system swap area. Thus the file does not have to be fetched from the file system upon each execution. As long as a copy remains in the swap area, the original text cannot be overwritten in the file system, nor can the file be deleted. (Directory entries can be removed so long as one link remains.)

Sharable files are made by the **−n** and **−z** options of *ld*(1).

To replace a sticky file that has been used do: (1) Clear the sticky bit with *chmod*(1). (2) Execute the old program to flush the swapped copy. This can be done safely even if others are using it. (3) Overwrite the sticky file. If the file is being executed by any process, writing will be prevented; it suffices to simply remove the file and then rewrite it, being careful to reset the owner and mode with *chmod* and *chown*(2). (4) Set the sticky bit again.

Only the super-user can set the sticky bit.

BUGS

Are self-evident.

Is largely unnecessary on the VAX; matters only for large programs that will page heavily to start, since text pages are normally cached incore as long as possible after all instances of a text image exit.

NAME

swapon – specify additional device for paging and swapping

SYNOPSIS

/etc/swapon −a
/etc/swapon name ...

DESCRIPTION

Swapon is used to specify additional devices on which paging and swapping are to take place. The system begins by swapping and paging on only a single device so that only one disk is required at bootstrap time. Calls to *swapon* normally occur in the system multi-user initialization file */etc/rc* making all swap devices available, so that the paging and swapping activity is interleaved across several devices.

Normally, the **−a** argument is given, causing all devices marked as "sw" swap devices in **/etc/fstab** to be made available.

The second form gives individual block devices as given in the system swap configuration table. The call makes only this space available to the system for swap allocation.

SEE ALSO

swapon(2), init(8)

FILES

/dev/[ru][pk]?b	normal paging devices

BUGS

There is no way to stop paging and swapping on a device. It is therefore not possible to make use of devices which may be dismounted during system operation.

NAME

sync − update the super block

SYNOPSIS

/etc/sync

DESCRIPTION

Sync executes the *sync* system primitive. *Sync* can be called to insure all disk writes have been completed before the processor is halted in a way not suitably done by *reboot*(8) or *halt*(8).

See *sync*(2) for details on the system primitive.

SEE ALSO

sync(2), fsync(2), halt(8), reboot(8), update(8)

NAME

syslog − log systems messages

SYNOPSIS

/etc/syslog [**−m***N*] [**−f***name*] [**−d**]

DESCRIPTION

Syslog reads a datagram socket and logs each line it reads into a set of files described by the configuration file /etc/syslog.conf. *Syslog* configures when it starts up and whenever it receives a hangup signal.

Each message is one line. A message can contain a priority code, marked by a digit in angle braces at the beginning of the line. Priorities are defined in <*syslog.h*>, as follows:

LOG_ALERT	this priority should essentially never be used. It applies only to messages that are so important that every user should be aware of them, e.g., a serious hardware failure.
LOG_SALERT	messages of this priority should be issued only when immediate attention is needed by a qualified system person, e.g., when some valuable system resource dissappears. They get sent to a list of system people.
LOG_EMERG	Emergency messages are not sent to users, but represent major conditions. An example might be hard disk failures. These could be logged in a separate file so that critical conditions could be easily scanned.
LOG_ERR	these represent error conditions, such as soft disk failures, etc.
LOG_CRIT	such messages contain critical information, but which can not be classed as errors, for example, 'su' attempts. Messages of this priority and higher are typically logged on the system console.
LOG_WARNING	issued when an abnormal condition has been detected, but recovery can take place.
LOG_NOTICE	something that falls in the class of "important information"; this class is informational but important enough that you don't want to throw it away casually. Messages without any priority assigned to them are typically mapped into this priority.
LOG_INFO	information level messages. These messages could be thrown away without problems, but should be included if you want to keep a close watch on your system.
LOG_DEBUG	it may be useful to log certain debugging information. Normally this will be thrown away.

It is expected that the kernel will not log anything below LOG_ERR priority.

The configuration file is in two sections separated by a blank line. The first section defines files that *syslog* will log into. Each line contains a single digit which defines the lowest priority (highest numbered priority) that this file will receive, an optional asterisk which guarantees that something gets output at least every 20 minutes, and a pathname. The second part of the file contains a list of users that will be informed on SALERT level messages. For example, the configuration file:

```
5*/dev/console
8/usr/spool/adm/syslog
3/usr/adm/critical

eric
```

```
kridle
kalash
```

logs all messages of priority 5 or higher onto the system console, including timing marks every 20 minutes; all messages of priority 8 or higher into the file /usr/spool/adm/syslog; and all messages of priority 3 or higher into /usr/adm/critical. The users "eric", "kridle", and "kalash" will be informed on any subalert messages.

The flags are:

−m Set the mark interval to *N* (default 20 minutes).

−f Specify an alternate configuration file.

−d Turn on debugging (if compiled in).

To bring *syslog* down, it should be sent a terminate signal. It logs that it is going down and then waits approximately 30 seconds for any additional messages to come in.

There are some special messages that cause control functions. "<*>N" sets the default message priority to *N*. "<$>" causes *syslog* to reconfigure (equivalent to a hangup signal). This can be used in a shell file run automatically early in the morning to truncate the log.

Syslog creates the file /etc/syslog.pid if possible containing a single line with its process id. This can be used to kill or reconfigure *syslog*.

FILES

/etc/syslog.conf — the configuration file
/etc/syslog.pid — the process id

BUGS

LOG_ALERT and LOG_SUBALERT messages should only be allowed to privileged programs.

Actually, *syslog* is not clever enough to deal with kernel error messages in the current implementation.

SEE ALSO

syslog(3)

NAME

telnetd — DARPA TELNET protocol server

SYNOPSIS

/etc/telnetd [**−d**] [*port*]

DESCRIPTION

Telnetd is a server which supports the DARPA standard TELNET virtual terminal protocol. The TELNET server operates at the port indicated in the "telnet" service description; see *services*(5). This port number may be overridden (for debugging purposes) by specifying a port number on the command line. If the **−d** option is specified, each socket created by *telnetd* will have debugging enabled (see SO_DEBUG in *socket*(2)).

Telnetd operates by allocating a pseudo-terminal device (see *pty*(4)) for a client, then creating a login process which has the slave side of the pseudo-terminal as **stdin, stdout**, and **stderr**. *Telnetd* manipulates the master side of the pseudo terminal, implementing the TELNET protocol and passing characters between the client and login process.

When a TELNET session is started up, *telnetd* sends a TELNET option to the client side indicating a willingness to do "remote echo" of characters. The pseudo terminal allocated to the client is configured to operate in "cooked" mode, and with XTABS and CRMOD enabled (see *tty*(4)). Aside from this initial setup, the only mode changes *telnetd* will carry out are those required for echoing characters at the client side of the connection.

Telnetd supports binary mode, and most of the common TELNET options, but does not, for instance, support timing marks. Consult the source code for an exact list of which options are not implemented.

SEE ALSO

telnet(1C)

BUGS

A complete list of the options supported should be given here.

NAME

telser – PUP Telnet Protocol Service

SYNOPSIS

/etc/pup/telser **[arg1] [Debug]**

DESCRIPTION

Telser provides the PUP Telnet service for a Unix time-sharing system. This allows users from other systems (including EtherTips) running PUP Telnet to log into the system. This command is normally run only by the super-user when the system is brought into multi-user mode.

If one command line argument is given, then helpful debugging information will be written in the argv area to be seen with the *ps* command. With two arguents, more debugging information will be printed, and if the second argument starts with the letter D then BSP debug information will be printed.

SEE ALSO

puptelnet(1), ftpser(8)

AUTHOR

Bill Nowicki

BUGS

There are some suspected bugs in the pty device drivers. If you figure them out, please let us know.

NAME

tftpd − DARPA Trivial File Transfer Protocol server

SYNOPSIS

/etc/tftpd [**−d**] [*port*]

DESCRIPTION

Tftpd is a server which supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the "tftp" service description; see *services*(5). This port number may be overridden (for debugging purposes) by specifying a port number on the command line. If the **−d** option is specified, each socket created by *tftpd* will have debugging enabled (see SO_DEBUG in *socket*(2)).

The use of *tftp* does not require an account or password on the remote system. Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling tftp service.

SEE ALSO

tftp(1C)

BUGS

This server is known only to be self consistent (i.e. it operates with the user TFTP program, *tftp*(1C)). Due to the unreliability of the transport protocol (UDP) and the scarcity of TFTP implementations, it is uncertain whether it really works.

The search permissions of the directories leading to the files accessed are not checked.

NAME

timeck − poll the localnet for the current time

SYNOPSIS

/etc/timeck [−s]

DESCRIPTION

The *timeck* program sends a broadcast message to the localnet (as defined in */etc/networks*, see *networks* (5)), requesting the current time (as defined in */etc/services*, see *services* (5)).

If the '−s' option is specified, then *timeck* will set the current host's time to the time reported by the first host responding.

SEE ALSO

inetd(8C)

BUGS

If the local network doesn't support broadcasting, *timeck* should consult */etc/hosts* (see *hosts* (5)) and send a message to each host it finds there that resides on the local network.

It is a feature, not a bug, that the name and address of the local net are determined by looking at */etc/networks* instead of */dev/kmem*.

NAME

trpt − transliterate protocol trace

SYNOPSIS

trpt [**−a**] [**−s**] [**−t**] [**−j**] [**−p** hex-address] [system [core]]

DESCRIPTION

Trpt interrogates the buffer of TCP trace records created when a socket is marked for "debugging" (see *setsockopt*(2)), and prints a readable description of these records. When no options are supplied, *trpt* prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB). The following options may be used to alter this behavior.

−s in addition to the normal output, print a detailed description of the packet sequencing information,

−t in addition to the normal output, print the values for all timers at each point in the trace,

−j just give a list of the protocol control block addresses for which there are trace records,

−p show only trace records associated with the protocol control block who's address follows,

−a in addition to the normal output, print the values of the source and destination addresses for each packet recorded.

The recommended use of *trpt* is as follows. Isolate the problem and enable debugging on the socket(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the **−A** option to *netstat*(1). Then run *trpt* with the **−p** option, supplying the associated protocol control block addresses. If there are many sockets using the debugging option, the **−j** option may be useful in checking to see if any trace records are present for the socket in question.

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

FILES

/vmunix
/dev/kmem

SEE ALSO

setsockopt(2), netstat(1)

DIAGNOSTICS

"no namelist" when the system image doesn't contain the proper symbols to find the trace buffer; others which should be self explanatory.

BUGS

Should also print the data for each input or output, but this is not saved in the race record.

The output format is inscrutable and should be described here.

NAME

tunefs − tune up an existing file system

SYNOPSIS

/etc/tunefs *tuneup-options special|filesys*

DESCRIPTION

Tunefs is designed to change the dynamic parameters of a file system which affect the layout policies. The parameters which are to be changed are indicated by the flags given below:

−a maxcontig

This specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay (see −d below). The default value is one, since most device drivers require an interrupt per disk transfer. Device drivers that can chain several buffers together in a single transfer should set this to the maximum chain length.

−d rotdelay

This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

−e maxbpg

This indicates the maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. Typically this value is set to about one quarter of the total blocks in a cylinder group. The intent is to prevent any single file from using up all the blocks in a single cylinder group, thus degrading access times for all files subsequently allocated in that cylinder group. The effect of this limit is to cause big files to do long seeks more frequently than if they were allowed to allocate all the blocks in a cylinder group before seeking elsewhere. For file systems with exclusively large files, this parameter should be set higher.

−m minfree

This value specifies the percentage of space held back from normal users; the minimum free space threshold. The default value used is 10%. This value can be set to zero, however up to a factor of three in throughput will be lost over the performance obtained at a 10% threshold. Note that if the value is raised above the current usage level, users will be unable to allocate files until enough files have been deleted to get under the higher threshold.

SEE ALSO

fs(5), newfs(8), mkfs(8)

McKusick, Joy, Leffler; "A Fast File System for Unix", Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720; TR #7, September 1982.

BUGS

This program should work on mounted and active file systems. Because the super-block is not kept in the buffer cache, the program will only take effect if it is run on dismounted file systems. (if run on the root file system, the system must be rebooted)

You can tune a file system, but you can't tune a fish.

NAME

update — periodically update the super block

SYNOPSIS

/etc/update

DESCRIPTION

Update is a program that executes the *sync*(2) primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file.

SEE ALSO

sync(2), sync(8), init(8), rc(8)

BUGS

With *update* running, if the CPU is halted just as the *sync* is executed, a file system can be damaged. This is partially due to DEC hardware that writes zeros when NPR requests fail. A fix would be to have *sync*(8) temporarily increment the system time by at least 30 seconds to trigger the execution of *update*. This would give 30 seconds grace to halt the CPU.

NAME

utime – adjust the access or modification time of a file

SYNOPSIS

utime [-a | -m] file [+-]nnU ...

DESCRIPTION

Utime adjusts the access (with -a option) or modification (-m option) time of a Unix file. The time is adjusted to be later (+) or earlier (-) than its current value by <nn> units U of time. <nn> is an integer; U is a single character standing for the units: s for seconds, m for minutes, h for hours, d for days, w for weeks.

This program is just a simple interface to utime(3). The default time option is -m (modification time). There is no default unit.

AUTHOR

Steve Hartwell. Manual page by Brian Reid.

BUGS

No way to set the time to an absolute value, although see filetime(1) for a means by which it can be hacked.

NAME

uurec – receive processed news articles via mail

SYNOPSIS

uurec

DESCRIPTION

uurec reads news articles on the standard input sent by *sendnews(8)*, decodes them, and gives them to *inews(1)* for insertion.

SEE ALSO

inews(1), readnews(1), recnews(8), sendnews(8), newscheck(1)

NAME

uusnap − show snapshot of the UUCP system

SYNOPSIS

uusnap

DESCRIPTION

Uusnap displays in tabular format a synopsis of the current UUCP situation. The format of each line is as follows:

```
site   N Cmds   N Data   N Xqts   Message
```

Where "site" is the name of the site with work, "N" is a count of each of the three possible types of work (command, data, or remote execute), and "Message" is the current status message for that site as found in the STST file.

Included in "Message" may be the time left before UUCP can re-try the call, and the count of the number of times that UUCP has tried to reach the site.

SEE ALSO

uucp(1C), UUCP Implementation Guide

NAME

vipw − edit the password file

SYNOPSIS

vipw

DESCRIPTION

Vipw edits the password file while setting the appropriate locks, and does any necessary processing after the password file is unlocked. If the password file is already being edited, then you will be told to try again later. The *vi* editor will be used unless the environment variable EDITOR indicates an alternate editor. *Vipw* performs a number of consistency checks on the password entry for *root*, and will not allow a password file with a "mangled" root entry to be installed.

SEE ALSO

chfn(1), chsh(1), passwd(1), passwd(5), adduser(8)

FILES

/etc/ptmp