999-801-312IS

AT&T UNIX™ PC Model 7300

UNIX System V Administrator's Manual

NOTICE

The information in this document is subject to change without notice. AT&T assumes no responsibility for any errors that may appear in this document.

TABLE OF CONTENTS

1. System Maintenance Commands

intro . . . . introduction to maintenance and application programs
accept . . . . . allow/prevent LP requests
bcopy . . . . . interactive block copy
chroot . . . . . change root directory for a command
clri . . . . . clear i-node
cron . . . . . clock daemon
devnm . . . . . device name
df . . . . . report number of free disk blocks
dismount . . . . . remove floppy or cartridge disk
fsck . . . . . file system consistency check and interactive repair
fsdb . . . . . file system debugger
fuser . . . . . identify processes using a file or file structure
getty . . . . . set terminal type, modes, speed, and line discipline
init . . . . . process control initialization
iv . . . . . initialize and maintain volume
killall . . . . . kill all active processes
login . . . . . sign on
lpadmin . . . . . configure the LP spooling system
lpsched . . . start/stop the LP request scheduler and move requests
mkfs . . . . . construct a file system
mknod . . . . . build special file
mount . . . . . mount and dismount file system
ncheck . . . . . generate names from i-numbers
rc . . . . . rc - system initialization shell script
reboot . . . . . reboot the system
setmnt . . . . . setmnt - establish mount table
shutdown . . . . . terminate all processing
uuclean . . . . . uucp spool directory clean-up
uusub . . . . . monitor uucp network
volcopy . . . . . copy file systems with label checking
wall . . . . . write to all users
whodo . . . . . who is doing what

7. Special Files

intro . . . . . introduction to special files
err . . . . . error-logging interface
escape . . . . . escape - output escape codes for bitmap windows
gd . . . . . general driver for moving-head disks
kbd . . . . . keyboard codes
lp . . . . . line printer
mem . . . . . core memory
null . . . . . the null file
termio . . . . . general terminal interface
tty . . . . . controlling terminal interface
window . . . . . window - bitmap windows

PERMUTED INDEX

LP requests. accept, reject: allow/prevent . . . . accept(1M)
killall: kill all active processes. . . . . killall(1M)
accept, reject: allow/prevent LP requests. . . . . accept(1M)
maintenance commands and application programs. /system . . . intro(1M)
bcopy: interactive block copy. . . . . bcopy(1M)
- output escape codes for bitmap windows. escape . . . . . escape(7)
window - bitmap windows. . . . . window(7)
bcopy: interactive block copy. . . . . bcopy(1M)
df: report number of free disk blocks. . . . . df(1M)
mknod: build special file. . . . . mknod(1M)
dismount: remove floppy or cartridge disk. . . . . dismount(1M)
fsck: file system consistency check and interactive repair. . . . fsck(1M)
copy file systems with label checking. volcopy, labelit: . . . . volcopy(1M)
for a command. chroot: change root directory . . . . chroot(1M)
uuclean: uucp spool directory clean-up. . . . . uuclean(1M)
clri: clear i-node. . . . . clri(1M)
cron: clock daemon. . . . . cron(1M)
clri: clear i-node. . . . . clri(1M)
escape - output escape codes for bitmap windows. . . . . escape(7)
kbd: keyboard codes. . . . . kbd(7)
change root directory for a command. chroot: . . . . . chroot(1M)
/to system maintenance commands and application/ . . . . intro(1M)
system. lpadmin: configure the LP spooling . . . . . lpadmin(1M)
interactive/ fsck: file system consistency check and . . . . . fsck(1M)
mkfs: construct a file system. . . . . mkfs(1M)
init, telinit: process control initialization. . . . . init(1M)
interface. tty: controlling terminal . . . . . tty(7)
bcopy: interactive block copy. . . . . bcopy(1M)
checking. volcopy, labelit: copy file systems with label . . . . volcopy(1M)
mem, kmem: core memory. . . . . mem(7)
cron: clock daemon. . . . . cron(1M)
cron: clock daemon. . . . . cron(1M)
fsdb: file system debugger. . . . . fsdb(1M)
devnm: device name. . . . . devnm(1M)
devnm: device name. . . . . devnm(1M)
blocks. df: report number of free disk . . . . df(1M)
uuclean: uucp spool directory clean-up. . . . . uuclean(1M)
chroot: change root directory for a command. . . . . chroot(1M)
type, modes, speed, and line discipline. /set terminal . . . . . getty(1M)
df: report number of free disk blocks. . . . . df(1M)
remove floppy or cartridge disk. dismount: . . . . . dismount(1M)
general driver for moving-head disks. gd: . . . . . gd(7)
mount, umount: mount and dismount file system. . . . . mount(1M)
cartridge disk. dismount: remove floppy or . . . . . dismount(1M)
whodo: who is doing what. . . . . whodo(1M)
gd: general driver for moving-head disks. . . . gd(7)
err: error-logging interface. . . . . err(7)
err: error-logging interface. . . . . err(7)
for bitmap windows. escape - output escape codes . . . . escape(7)
windows. escape - output escape codes for bitmap . . . . . escape(7)
setmnt - establish mount table. . . . . setmnt(1M)
mknod: build special file. . . . . mknod(1M)
null: the null file. . . . . null(7)
/identify processes using a file or file structure. . . . . fuser(1M)
processes using a file or file structure. /identify . . . . . fuser(1M)
and interactive repair. fsck: file system consistency check . . . . fsck(1M)

fsdb: file system debugger. . . . . fsdb(1M)
mkfs: construct a file system. . . . . mkfs(1M)
umount: mount and dismount file system. mount, ... mount(1M)
volcopy, labelit: copy file systems with label/ . . . . . volcopy(1M)
intro: introduction to special files. . . . . intro(7)
dismount: remove floppy or cartridge disk. . . . . dismount(1M)
df: report number of free disk blocks. . . . . df(1M)
ncheck: generate names from i-numbers. . . . . ncheck(1M)
check and interactive repair. fsck: file system consistency . . . fsck(1M)
fsdb: file system debugger. . . . . fsdb(1M)
using a file or file/ fuser: identify processes . . . . . fuser(1M)
moving-head disks. gd: general driver for . . . . . gd(7)
ncheck: generate names from i-numbers. . . ncheck(1M)
modes, speed, and line/ getty: set terminal type, ... getty(1M)
file or file/ fuser: identify processes using a . . . . . fuser(1M)
initialization. init, telinit: process control . . . . . init(1M)
init, telinit: process control initialization. . . . . init(1M)
rc - system initialization shell script. . . . . rc(1M)
volume. iv: initialize and maintain . . . . . iv(1M)
clri: clear i-node. . . . . clri(1M)
bcopy: interactive block copy. . . . . bcopy(1M)
system consistency check and interactive repair. /file . . . . . fsck(1M)
err: error-logging interface. . . . . err(7)
termio: general terminal interface. . . . . termio(7)
tty: controlling terminal interface. . . . . tty(7)
files. intro: introduction to special . . . . intro(7)
maintenance commands and/ intro: introduction to system . . . . intro(1M)
intro: introduction to special files. . . . . intro(7)
maintenance commands/ intro: introduction to system . . . . . intro(1M)
ncheck: generate names from i-numbers. . . . . ncheck(1M)
volume. iv: initialize and maintain . . . . . iv(1M)
kbd: keyboard codes. . . . . kbd(7)
kbd: keyboard codes. . . . . kbd(7)
killall: kill all active processes. . . . . killall(1M)
processes. killall: kill all active . . . . . killall(1M)
mem, kmem: core memory. . . . . mem(7)
copy file systems with label checking. /labelit: . . . . . volcopy(1M)
with label checking. volcopy, labelit: copy file systems . . . . . volcopy(1M)
type, modes, speed, and line discipline. /set terminal . . . . getty(1M)
lp: line printer. . . . . lp(7)
login: sign on. . . . . login(1M)
lp: line printer. . . . . lp(7)
/lpshut, lpmove: start/stop the LP request scheduler and move/ . . lpsched(1M)
accept, reject: allow/prevent LP requests. . . . . accept(1M)
lpadmin: configure the LP spooling system. . . . . lpadmin(1M)
spooling system. lpadmin: configure the LP . . . . lpadmin(1M)
request/ lpsched, lpshut, lpmove: start/stop the LP . . . . lpsched(1M)
start/stop the LP request/ lpsched, lpshut, lpmove: . . . . . lpsched(1M)
LP request scheduler/ lpsched, lpshut, lpmove: start/stop the . . . lpsched(1M)
iv: initialize and maintain volume. . . . . iv(1M)
intro: introduction to system maintenance commands and/ . . . . intro(1M)
mem, kmem: core memory. . . . . mem(7)
mem, kmem: core memory. . . . . mem(7)
mkfs: construct a file system. . . . mkfs(1M)
mknod: build special file. . . . . mknod(1M)
getty: set terminal type, modes, speed, and line/ . . . . . getty(1M)
uusub: monitor uucp network. . . . . uusub(1M)
system. mount, umount: mount and dismount file . . . . . mount(1M)
setmnt - establish mount table. . . . . setmnt(1M)

dismount file system. mount, umount: mount and . . . . . mount(1M)
the LP request scheduler and move requests. /start/stop . . . . . lpsched(1M)
gd: general driver for moving-head disks. . . . . gd(7)
i-numbers. ncheck: generate names from . . . . ncheck(1M)
uusub: monitor uucp network. . . . . uusub(1M)
null: the null file. . . . . null(7)
null: the null file. . . . . null(7)
windows. escape - output escape codes for bitmap . . . escape(7)
lp: line printer. . . . . lp(7)
init, telinit: process control/ . . . . . init(1M)
killall: kill all active processes. . . . . killall(1M)
structure. fuser: identify processes using a file or file . . . . fuser(1M)
shutdown: terminate all processing. . . . . shutdown(1M)
shell script. rc - system initialization . . . . . rc(1M)
reboot: reboot the system. . . . . reboot(1M)
reboot: reboot the system. . . . . reboot(1M)
requests. accept, reject: allow/prevent LP . . . . . accept(1M)
disk. dismount: remove floppy or cartridge . . . . . dismount(1M)
check and interactive repair. /system consistency . . . . . fsck(1M)
blocks. df: report number of free disk . . . . . df(1M)
/lpmove: start/stop the LP request scheduler and move/ . . . . lpsched(1M)
reject: allow/prevent LP requests. accept, ... accept(1M)
LP request scheduler and move requests. /start/stop the . . . . . lpsched(1M)
chroot: change root directory for a command. . . . chroot(1M)
/start/stop the LP request scheduler and move requests. . . . . lpsched(1M)
- system initialization shell script. rc . . . . . rc(1M)
table. setmnt - establish mount . . . . . setmnt(1M)
rc - system initialization shell script. . . . . rc(1M)
processing. shutdown: terminate all . . . . . shutdown(1M)
login: sign on. . . . . login(1M)
/set terminal type, modes, speed, and line discipline. . . . . getty(1M)
uuclean: uucp spool directory clean-up. . . . . uuclean(1M)
lpadmin: configure the LP spooling system. . . . . lpadmin(1M)
lpsched, lpshut, lpmove: start/stop the LP request/ . . . . . lpsched(1M)
processes using a file or file structure. fuser: identify . . . . . fuser(1M)
setmnt - establish mount table. . . . . setmnt(1M)
initialization. init, telinit: process control . . . . . init(1M)
termio: general terminal interface. . . . . termio(7)
tty: controlling terminal interface. . . . . tty(7)
and line/ getty: set terminal type, modes, speed, ... getty(1M)
shutdown: terminate all processing. . . . . shutdown(1M)
interface. termio: general terminal . . . . . termio(7)
interface. tty: controlling terminal . . . . . tty(7)
getty: set terminal type, modes, speed, and line/ . . . . getty(1M)
file system. mount, umount: mount and dismount . . . . mount(1M)
wall: write to all users. . . . . wall(1M)
fuser: identify processes using a file or file/ . . . . . fuser(1M)
clean-up. uuclean: uucp spool directory . . . . uuclean(1M)
uusub: monitor uucp network. . . . . uusub(1M)
uuclean: uucp spool directory clean-up. . . . uuclean(1M)
uusub: monitor uucp network. . . . uusub(1M)
systems with label checking. volcopy, labelit: copy file . . . . . volcopy(1M)
iv: initialize and maintain volume. . . . . iv(1M)
wall: write to all users. . . . . wall(1M)
whodo: who is doing what. . . . . whodo(1M)
whodo: who is doing what. . . . . whodo(1M)
window - bitmap windows. . . . . window(7)
output escape codes for bitmap windows. escape - . . . . . escape(7)
window - bitmap windows. . . . . window(7)

wall: write to all users. . . . . wall(1M)

NAME

intro - introduction to system maintenance commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes. The commands in this section should be used along with those listed in Section 1 of the *UNIX System User's Manual*. References to other manual entries not of the form *name*(1M), *name*(7) or *name*(8) refer to entries of that manual.

COMMAND SYNTAX

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option*(*s*)] [*cmdarg*(*s*)]
where:

name	The name of an executable file.
option	– *noargletter*(*s*) or, – *argletter*<>*optarg* where <> is optional white space.
noargletter	A single letter representing an option without an argument.
argletter	A single letter representing an option requiring an argument.
optarg	Argument (character string) satisfying preceding *argletter*.
cmdarg	Path name (or other command argument) *not* beginning with – or, – by itself indicating the standard input.

SEE ALSO

getopt(1), getopt(3C).
UNIX System User's Manual.
UNIX System Administrator's Guide.

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (see *wait*(2) and *exit*(2)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

BUGS

Regretfully, many commands do not adhere to the aforementioned syntax.

NAME

accept, reject - allow/prevent LP requests

SYNOPSIS

/usr/lib/accept destinations
/usr/lib/reject [**−r**[reason]] destinations

DESCRIPTION

Accept allows *lp*(1) to accept requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*.

Reject prevents *lp*(1) from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*. The following option is useful with *reject*.

−r[*reason*] Associates a *reason* with preventing *lp* from accepting requests. This *reason* applies to all printers mentioned up to the next **−r** option. *Reason* is reported by *lp* when users direct requests to the named *destinations* and by *lpstat*(1). If the **−r** option is not present or the **−r** option is given without a *reason*, then a default *reason* will be used.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1).

NAME

bcopy - interactive block copy

SYNOPSIS

/etc/bcopy

DESCRIPTION

Bcopy dates from a time when neither the UNIX file system nor the DEC disk drives were as reliable as they are now. *Bcopy* copies from and to files starting at arbitrary block (512-byte) boundaries.

The following questions are asked:

to: (you name the file or device to be copied to).
offset: (you provide the starting "to" block number).
from: (you name the file or device to be copied from).
offset: (you provide the starting "from" block number).
count: (you reply with the number of blocks to be copied).

After **count** is exhausted, the **from** question is repeated (giving you a chance to concatenate blocks at the **to+offset+count** location). If you answer **from** with a carriage return, everything starts over.

Two consecutive carriage returns terminate *bcopy*.

SEE ALSO

cpio(1), dd(1).

NAME

chroot - change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

The given command is executed *relative to the new root*. The meaning of any initial slashes (/) in path names is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Notice that:

```
chroot newroot command >x
```

will create the file **x** relative to the original root, not the new one.

This command is restricted to the super-user.

The new root path name is always relative to the current root: even if a *chroot* is currently in effect, the *newroot* argument is relative to the current root of the running process.

SEE ALSO

chdir(2).

BUGS

One should exercise extreme caution when referencing special files in the new root file system.

NAME

clri - clear i-node

SYNOPSIS

/etc/clri file-system i-number ...

DESCRIPTION

Clri writes zeros on the 64 bytes occupied by the i-node numbered *i-number*. *File-system* must be a special file name referring to a device containing a file system. After *clri* is executed, any blocks in the affected file will show up as "missing" in an *fsck*(1M) of the *file-system*. This command should only be used in emergencies and extreme care should be exercised.

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

fsck(1M), fsdb(1M), ncheck(1M), fs(4).

BUGS

If the file is open, *clri* is likely to be ineffective.

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**. Because *cron* never exits, it should be executed only once. This is best done by running *cron* from the initialization process through the file **/etc/rc** (see *init*(1M)).

The file **crontab** consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify in order:

minute (0-59),
hour (0-23),
day of the month (1-31),
month of the year (1-12),
and day of the week (0-6, with 0=Sunday).

Each of these patterns may contain:

a number in the (respective) range indicated above;
two numbers separated by a minus (indicating an inclusive range);
a list of numbers separated by commas (meaning all of these numbers); or
an asterisk (meaning all legal values).

The sixth field is a string that is executed by the shell at the specified time(s). A % in this field is translated into a new-line character. Only the first line (up to a % or the end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

Cron examines **crontab** once a minute to see if it has changed; if it has, *cron* reads it. Thus it takes only a minute for entries to become effective.

FILES

/usr/lib/crontab
/usr/adm/cronlog

SEE ALSO

init(1M), sh(1).

DIAGNOSTICS

A history of all actions by *cron* are recorded in **/usr/adm/cronlog**.

BUGS

Cron reads **crontab** only when it has changed, but it reads the in-core version of that table once a minute. A more efficient algorithm could be used. The overhead in running *cron* is about one percent of the CPU, exclusive of any commands executed by *cron*.

NAME

devnm - device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

Devnm identifies the special file associated with the mounted file system where the argument *name* resides (as a special case, both the block device name and the swap device name is printed for the argument name / if swapping is done on the same disk section as the **root** file system). Argument names must be full path names.

This command is most commonly used by **/etc/rc** (see *rc*(1M)) to construct a mount table entry for the **root** device.

EXAMPLE

The command:

```
/etc/devnm /
```

produces

```
fp002 /
```

if /dev/fp002 is mounted on /.

Or the command:

```
/etc/devnm /u
```

produces

```
fp003 /u
```

if /dev/fp003 is mounted on /u.

FILES

/etc/mnttab

SEE ALSO

rc(1M), setmnt(1M).

NAME

df - report number of free disk blocks

SYNOPSIS

df [**−t**] [**−f**] [file-systems]

DESCRIPTION

Df prints out the number of free blocks and free i-nodes available for on-line file systems by examining the counts kept in the super-blocks; *file-systems* may be specified either by device name (e.g., **/dev/fp002**) or by mounted directory name (e.g., **/usr**). If the *file-systems* argument is unspecified, the free space on all of the mounted file systems is printed.

The **−t** flag causes the total allocated block figures to be reported as well.

If the **−f** flag is given, only an actual count of the blocks in the free list is made (free i-nodes are not reported). With this option, *df* will report on raw devices.

FILES

/dev/fp*
/etc/mnttab

SEE ALSO

fs(4), mnttab(4).

NAME

dismount - remove floppy or cartridge disk

SYNOPSIS

dismount [**-f**]

DESCRIPTION

DISMOUNT prevents damage to a floppy or cartridge disk caused by sudden removal of the disk from its drive. The program waits for pending input/output on the disk to complete, forbids further input/output to the disk, unmounts the disk's file systems, and clears the pulled flag in the disk's volume home block. When *dismount* finishes without error, it is safe to take the disk out of the drive.

-f is historical.

A disk that was removed without a prior dismount is noticeable because its pulled flag is still set. Inserting such a disk in the drive causes UNIX to print a warning on the system console. If you receive such a warning, check the consistency of file systems and databases on the disk.

FILES

/etc/mtab - mounted file system list

SEE ALSO

fsck(1M), update(1), fp(7).

NAME

fsck - file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [**−y**] [**−n**] [**−sX**] [**−SX**] [**−t** file] [**−q**] [**−D**] [**−f**] [file-systems]

DESCRIPTION

Fsck

Fsck audits and interactively repairs inconsistent conditions for UNIX file systems. If the file system is consistent then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions 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 options are interpreted by *fsck*.

−y Assume a yes response to all questions asked by *fsck*.

−n Assume a no response to all questions asked by *fsck*; do not open the file system for writing.

−s*X* Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The **−s***X* option allows for creating an optimal free-list organization. The following forms of *X* are supported for the following devices:

-s3 (RP03)
-s4 (RP04, RP05, RP06)
-sBlocks-per-cylinder:Blocks-to-skip (for anything else)

If *X* is not given, the values used when the file system was created are used. If these values were not specified, then the value *400*:*7* is used.

−S*X* Conditionally reconstruct the free list. This option is like **−s***X* above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **−S** will force a no response to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.

−t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **−t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **−t** flag, *fsck* will prompt the operator for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

−q Quiet *fsck*. Do not print size-check messages in Phase 1. Unreferenced **fifos** will silently be removed. If *fsck* requires it, counts in the superblock will be automatically fixed and the free list salvaged.

−D Directories are checked for bad blocks. Useful after system crashes.

−f Fast check. Check block and sizes (Phase 1) and check the free list (Phase 5). The free list will be reconstructed (Phase 6) if it is necessary.

If no *file-systems* are specified, *fsck* will read a list of default file systems from the file **/etc/checklist**.

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:
   Incorrect number of blocks.
   Directory size not 16-byte aligned.
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 than 65536 inodes.
   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, if the files are nonempty. The user will be notified if the file or directory is empty or not. If it is empty, *fsck* will silently remove them. *Fsck* will force the reconnection of nonempty directories. The name assigned is the inode number. The only restriction is that the directory **lost+found** must preexist in the root of the file system 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 and should be used with everything but the *root* file system.

Dfsck

Dfsck allows two file system checks on two different drives simultaneously. *options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A − is the separator between the file system groups.

The *dfsck* program permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsck* will print the file system name for each message to the operator. When answering a question from *dfsck*, the operator must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

Do not use *dfsck* to check the *root* file system.

FILES

/etc/checklist	contains default list of file systems to check.
/etc/checkall	optimizing *dfsck* shell file.

SEE ALSO

checkall(1M), clri(1M), ncheck(1M), checklist(4), fs(4), crash(8).
Setting up UNIX in the *UNIX System Administrator's Guide*.

BUGS

Inode numbers for . and .. in each directory should be checked for validity.

DIAGNOSTICS

The diagnostics produced by *fsck* are intended to be self-explanatory.

NAME

fsdb - file system debugger

SYNOPSIS

/etc/fsdb special [−]

DESCRIPTION

Fsdb can be used to patch up a damaged file system after a crash. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the file system tree.

Fsdb contains several error checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking *fsdb* with the optional − argument or by the use of the **O** symbol. (*Fsdb* reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

Fsdb reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by *fsdb* are:

#	absolute address
i	convert from i-number to i-node address
b	convert to block address
d	directory slot offset
+,−	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=−	decremental assignment
="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically

sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes

The **f** symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. Inodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a#	data block numbers (0 - 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current working i-node.
ln=4	changes the link count for the working i-node to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working i-node.

2i.fd	prints the first 32 directory entries for the root i-node of this file system.
d5i.fc	changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.
512B.p0o	prints the superblock of this file system in octal.
2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
d7.nm=" name"	changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.
a2b.p0d	prints the third block of the current inode as directory entries.

SEE ALSO

fsck(1M), dir(4), fs(4).

NAME

fuser - identify processes using a file or file structure

SYNOPSIS

/etc/fuser [−**ku**] files [−] [[−**ku**] files]

DESCRIPTION

Fuser lists the process IDs of the processes using the *files* specified as arguments. For block special devices, all processes using any file on that device are listed. The process ID is followed by **c**, **p** or **r** if the process is using the file as its current directory, the parent of its current directory (only when in use by the system), or its root directory, respectively. If the −**u** option is specified, the login name, in parentheses, also follows the process ID. In addition, if the −**k** option is specified, the **SIGKILL** signal is sent to each process. Only the super-user can terminate another user's process (see *kill*(2)). Options may be respecified between groups of files. The new set of options replaces the old set, with a lone dash canceling any options currently in force.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

EXAMPLES

fuser -ku /dev/dsk1?

will terminate all processes that are preventing disk drive one from being unmounted if typed by the super-user, listing the process ID and login name of each as it is killed.

fuser -u /etc/passwd

will list process IDs and login names of processes that have the password file open.

fuser -ku /dev/dsk1? -u /etc/passwd

will do both of the above examples in a single command line.

Note that the above device names for disks are generic to the 3B20S and may be different on other processors.

FILES

/unix	for namelist
/dev/kmem	for system image
/dev/mem	also for system image

SEE ALSO

mount(1M), ps(1), kill(2), signal(2).

NAME

getty - set terminal type, modes, speed, and line discipline

SYNOPSIS

/etc/getty [**−h**] [**−t** timeout] line [speed [type [linedisc]]]
/etc/getty **−c** file

DESCRIPTION

Getty is a program that is invoked by *init*(1M). It is the second process in the series, (*init-getty-login-shell*) that ultimately connects a user with UNIX. Initially *getty* generates a system identification message from the values returned by the *uname*(2) system call. Then, if */etc/issue* exists, it outputs this to the user's terminal, followed finally by the login message field for the entry it is using from **/etc/gettydefs**. *Getty* reads the user's login name and invokes the *login*(1) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used.

Line is the name of a tty line in **/dev** to which *getty* is to attach itself. *Getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing. Unless *getty* is invoked with the **−h** flag, *getty* will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The **−t** flag plus *timeout* in seconds, specifies that *getty* should exit if the open on the line succeeds and no one types anything in the specified number of seconds. The optional second argument, *speed*, is a label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* what speed to initially run at, what the login message should look like, what the inital tty settings are, and what speed to try next should the user indicate that the speed is inappropriate. (By typing a *<break>* character.) The default *speed* is 300 baud. The optional third argument, *type*, is a character string describing to *getty* what type of terminal is connected to the line in question. *Getty* understands the following types:

none	default
vt61	DEC vt61
vt100	DEC vt100
hp45	Hewlett-Packard HP45
c100	Concept 100

The default terminal is **none**; i.e., any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition. The optional fourth argument, *linedisc*, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, **LDISC0**.

When given no optional arguments, *getty* sets the *speed* of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either

parity allowed, newline characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

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 *ioctl*(2)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

In addition to the standard UNIX erase and kill characters (**#** and **@**), *getty* also understands **\b** and **^U**. If the user uses a **\b** as an erase, or **^U** as a kill character, *getty* sets the standard erase character and/or kill character to match.

Getty also understands the "standard" ESS2 protocols for erasing, killing and aborting a line, and terminating a line. If *getty* sees the ESS erase character, **_**, or kill character, **$**, or abort character, **&**, or the ESS line terminators, **/** or **!**, it arranges for this set of characters to be used for these functions.

Finally, *login* is called with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which will place them in the environment (see *login*(1)).

A check option is provided. When *getty* is invoked with the **−c** option and *file*, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *ioctl*(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

ct(1C), init(1M), login(1), ioctl(2), gettydefs(4), inittab(4), tty(7).

BUGS

While *getty* does understand simple single character quoting conventions, it is not possible to quote the special control characters that *getty* uses to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. Therefore it is not possible to login via *getty* and type a **#**, **@**, **/**, **!**, **_**, backspace, **^U**, **^D**, or **&** as part of your login name or arguments. They will always be interrepted as having their special meaning as described above.

NAME

init, telinit - process control initialization

SYNOPSIS

/etc/init [0123456SsQq]

/etc/telinit [0123456sSQqabc]

DESCRIPTION

Init

Init is a general process spawner. Its primary role is to create processes from a script stored in the file **/etc/inittab** (see *inittab*(4)). This file usually has *init* spawn *getty*'s on each line that a user may log in on. It also controls autonomous processes required by any particular system.

Init considers the system to be in a *run-level* at any given time. A *run-level* can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by *init* for each of these *run-levels* is defined in the *inittab* file. *Init* can be in one of eight *run-levels*, **0**–**6** and **S** or **s**. The *run-level* is changed by having a privileged user run **/etc/init** (which is linked to */etc/telinit*). This user spawned *init* sends appropriate signals to the orginal *init* spawned by the operating system when the system was rebooted, telling it which *run-level* to change to.

Init is invoked inside UNIX as the last step in the boot procedure. The first thing *init* does is to look for **/etc/inittab** and see if there is an entry of the type *initdefault* (see *inittab*(4)). If there is, *init* uses the *run-level* specified in that entry as the initial *run-level* to enter. If this entry is not in *inittab* or *inittab* is not found, *init* requests that the user enter a *run-level* from the virtual system console, **/dev/syscon.** If an **S** (**s**) is entered, *init* goes into the *SINGLE USER* level. This is the only *run-level* that doesn't require the existence of a properly formated *inittab* file. If **/etc/inittab** doesn't exist, then by default the only legal *run-level* that *init* can enter is the *SINGLE USER* level. In the *SINGLE USER* level the virtual console terminal **/dev/syscon** is opened for reading and writing and the command **/bin/su** is invoked immediately. To exit from the *SINGLE USER run-level* one of two options can be elected. First, if the shell is terminated (via an end-of-file), *init* will reprompt for a new *run-level.* Second, the *init* or *telinit* command can signal *init* and force it to change the *run-level* of the system.

When attempting to boot the system, failure of *init* to prompt for a new *run-level* may be due to the fact that the device **/dev/syscon** is linked to a device other than the physical system teletype (**/dev/systty**). If this occurs, *init* can be forced to relink **/dev/syscon** by typing a delete on the system teletype which is co-located with the processor.

When *init* prompts for the new *run-level*, the operator may only enter one of the digits **0** through **6** or the letters **S** or **s**. If **S** is entered *init* operates as previously described in *SINGLE USER* mode with the additional result that **/dev/syscon** is linked to the user's terminal line, thus making it the virtual system

console. A message is generated on the physical console, **/dev/systty**, saying where the virtual terminal has been relocated.

When *init* comes up initially and whenever it switches out of *SINGLE USER* state to normal run states, it sets the *ioctl*(2) states of the virtual console, **/dev/syscon**, to those modes saved in the file **/etc/ioctl.syscon**. This file is written by *init* whenever *SINGLE USER* mode is entered. If this file doesn't exist when *init* wants to read it, a warning is printed and default settings are assumed.

If a **0** through **6** is entered *init* enters the corresponding *run-level*. Any other input will be rejected and the user will be reprompted. If this is the first time *init* has entered a *run-level* other than *SINGLE USER*, *init* first scans *inittab* for special entries of the type *boot* and *bootwait*. These entries are performed, providing the *run-level* entered matches that of the entry before any normal processing of *inittab* takes place. In this way any special initialization of the operating system,such as mounting file systems, can take place before users are allowed onto the system. The *inittab* file is scanned to find all entries that are to be processed for that *run-level*.

Run-level **2** is usually defined by the user to contain all of the terminal processes and daemons that are spawned in the multi-user environment.

In a multi-user environment, the *inittab* file is usually set up so that *init* will create a process for each terminal on the system.

For terminal processes, ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as the result of hanging up. When *init* receives a child death signal, telling it that a process it spawned has died, it records the fact and the reason it died in **/etc/utmp** and **/etc/wtmp** if it exists (see *who*(1)). A history of the processes spawned is kept in **/etc/wtmp** if such a file exists.

To spawn each process in the *inittab* file, *init* reads each entry and for each entry which should be respawned, it forks a child process. After it has spawned all of the processes specified by the *inittab* file, *init* waits for one of its descendant processes to die, a powerfail signal, or until *init* is signaled by *init* or *telinit* to change the system's *run-level*. When one of the above three conditions occurs, *init* re-examines the *inittab* file. New entries can be added to the *inittab* file at any time; however, *init* still waits for one of the above three conditions to occur. To provide for an instantaneous response the **init Q** or **init q** command can wake *init* to re-examine the *inittab* file.

If *init* receives a *powerfail* signal (*SIGPWR*) and is not in *SINGLE USER* mode, it scans *inittab* for special powerfail entries. These entries are invoked (if the *run-levels* permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions whenever the operating system experiences a power failure.

When *init* is requested to change *run-levels* (via *telinit*), *init* sends the warning signal (**SIGTERM**) to all processes that are undefined in the target *run-level*. *Init* waits 20 seconds before forcibly terminating these processes via the kill signal (**SIGKILL**).

Telinit

Telinit, which is linked to */etc/init*, is used to direct the actions of *init*. It takes a one character argument and signals *init* via the kill system call to perform the appropriate action. The following arguments serve as directives to *init*.

0–6 tells *init* to place the system in one of the *run-levels* **0–6**.

a,b,c tells *init* to process only those **/etc/inittab** file entries having the **a**, **b** or **c** *run-level* set.

Q,q tells *init* to re-examine the **/etc/inittab** file.

s,S tells *init* to enter the single user environment. When this level change is effected, the virtual system teletype, **/dev/syscon**, is changed to the terminal from which the command was executed.

Telinit can only be run by someone who is super-user or a member of group **sys**.

FILES

/etc/inittab
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/dev/syscon
/dev/systty

SEE ALSO

getty(1M), login(1), sh(1), who(1), kill(2), inittab(4), utmp(4).

DIAGNOSTICS

If *init* finds that it is continuously respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string, and generate an error message on the system console, and refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in the *inittab*.

NAME

iv - initialize and maintain volume

SYNOPSIS

iv [**-iustdwmv**] *special* [*descriptionfile*]

DESCRIPTION

Iv initializes and maintains a PC7300 disk volume. *Special* and *descriptionfile* specify the disk and a description file for it; these are described below. *Iv* does one of five operation, specified by the following options:

-i completely initialize a volume. This consists of five phases:

1. Initialize *iv's* internal Volume Home Block, based on *descriptionfile* and the disk type. If the disk can support bad block handling (all types except floppies), create an internal Bad Block Table. Put bad block data from *descriptionfile* and volume's existing Bad Block Table (if any) in internal Bad Block Table.
2. Format medium.
3. Perform a surface check. If the disk can support bad block handling, add bad blocks to the Bad Block Table. If the disk cannot support bad block handling, the first bad spot causes the disk to be rejected.
4. Write out the Volume Home Block. This has the effect of dividing the volume into slices (partitions).
5. Allocate and write out the files that share the Reserved Area (slice 0) with the Volume Home Block. If the disk can support bad block handling, one of these files is the Bad Block Table. Other files are specified in *descriptionfile.*

-u

Update the volume home block. This is the same as **-i** except that the second and third phases (medium formatting and surface check) are skipped.

-s

Surface test. Any bad blocks discovered are added to the bad block table.

-t

Tell volume description. Display volume home block in human-readable form. No description file is needed. The volume's contents are not affected.

-d

Description file display. A description file that describes the current state of the volume is written to the standard output. If the Reserved Area contains a loader, the loader keyword's value is writtten as /usr/lib/iv/loader. If the Reserved Area contains a down load image area, the Down Load Area Description lists files whose names are of the form

/usr/lib/iv/wsxxx.yyy

Where *xxx* is the numeric device identification; and *yyy* is **422** if *xxx* is even, **232** if *xxx* is odd.

The **−f** option, equivalent to **−u** is provided for compatibility with older versions of *iv*. It should not be used, as it may disappear in future releases.

In addition to the single operation option (**−i, −u, −s, −t,** or **−d)** you can specify any or all of the following options:

−v Verbose display output. If the display includes The Volume Home Block, also include the bad block table.

−l A normal surface test consists of a single pass over the disk; **−l** specifies ten passes.

−w A normal surface test pass consists of a read pass; **− w** specifies a write pass before each read pass.

File Parameters

Special is the character special file for slice zero on the volume. This name takes the form **/dev/rfp0** *t* **0,** where *t* is 0 for winchester, 1 for cartridge, 2 for floppy.

Descriptionfile is a text file that describes the volume. It is required by the **−i** and **−u** option. The description file consists of four parts:

general disk description
reserved area files description
bad blocks description
partition table description

The four descriptions are separated from each other by four lines, each of which contains only a single dollar sign ($). Specifics for each of the five descriptions are given under separate headings below.

General Description

Each line in the General Description begins with a keyword. Some keywords are followed by values; the value is separated from the keyword by spaces or tabs. For example:

hitech
cylinders 25

Each keyword is only used once. Here are the valid keywords.

type Mandatory, unless the volume is already initialized in PC7300 format. Value is disktype: " HD" for winchester, " SY" for hard disk cartridge, and " FD" for floppy.

Mandatory, unless the volume is already initialized in PC7300 format. Value is the volume name. Any characters except spaces or tabs are permitted in the volume name. The actual name in the Volume Home Block is always exactly six characters; *iv* right truncates names that are too long and right pads with nulls names that are too short.

cylinders
Mandatory, unless the volume is already initialized in PC7300 format. Value is the number of cylinders on the disk. This must be a positive number not greater than 1024.

heads Mandatory, unless the volume is already initialized in PC7300 format. Value is the number of heads on the disk. This must be a positive number not greater than 15.

sectors
Mandatory, unless the volume is already initialized in PC7300 format. Value is the number of physical sectors per track.

steprate
Mandatory, unless the volume is already initialized in PC7300 format. Value is a number that is passed to the disk controller. Currently this number must be 0.

exchangeable
If this keyword is present, the disk can be removed from its drive (floppy or hard disk cartridge).

hitech
If this keyword is present, write precompensation is not required on the disk. See the disk manufacturer's documentation for further information.

precomp
The value is *c* /16, where *c* is the cylinder at which precomposition should start. See the disk manufacturer's documentation for further information.

Reserved Area Description

The Reserved Area Description describes the files that share slice zero with the volume home block. Each line in the Reserved Area Description consists of a keyword followed by one or more parameters; one or more tabs or spaces separates keywords and parametetrs from each other. Here are the valid keywords and their meanings. (A logical block is 1024 bytes long.)

loader
Describes the loader area. The first, mandatory, parameter is the full pathname of an a.out file to put in the loader area. The second, optional, parameter is the size of the loader area in logical blocks. If the second parameter is missing, the size of the a.out file is used.

badblocktable
Describes the bad block table. The first, mandatory, parameter is the size of the bad block table in logical

blocks. The second, optional, parameter is only used when an existing bad block table contains errors; this parameter is "empty" to clear the bad block table, missing otherwise.

dump Describes area to contain dump after crash. Only, mandator, parameter, specifies the size of the dump area in logical blocks.

All lines valid for the Reserved Area Files Description are optional. However, the bad block table is mandatory on a volume which supports bad block handling; the loader area is mandatory on a volume which is to hold and operating system; and a dump area is used on a volume which is to hold an operating system. A system volume cannot have a bootable program area.

Bad Blocks Description

The Bad Block Description explicitly specifies up to 255 bad blocks to be added to the bad blocks table. *Iv* merges specified bad block information with information already in the bad block table (if there already is one) and bad block information discovered through the surface test.

Each bad block entry is a single line. There are two forms:

s

Where *s* is a sector number; *c h b* where *c* is a cylinder number, *h* is a head number, and *b* is a byte number. Both forms condemn a single sector, the second the sector that contains the specified byte.

The last sector on each track serves as a bad block alternate. *Iv* chooses to alternates in a way that minimizes extra seeking for alternate blocks.

Partition Table Description

The Partition Table Description shows where the slices (partitions) are to begin. Each line in the file consists ofa track number, the starting track of a slice. Slices are listed in ascending numeric order and begin on successively higher tracks. The beginning of a slice defines the end of the previous slice. The first track number is always 0, since slice zero always begins on a track zero.

There can be at most 16 slices on a disk. It is a fatal error to specify a slice one that doesn't leave enough room in slice zero for the Volume Home Block and the slice zero files.

EXAMPLES

Here is an example of a disk description file.

```
        #    iv description file for standard floppy disk
        type FD
        name Data
        cylinders 80
heads   2
sectors 8
        steprate        0
$
$
$
0
1
```

FILES

/dev/rfp* - disk character special files
/usr/lib/iv/* - prototype description files

SEE ALSO

dismount(1), update(1), fp(7).

WARNINGS

The **-i, -u,** and **-s** operations are dangerous or fatal to existing volume data. Always precede these operations with a backup.

When a new bad block is itself an alternate block, *iv* may produce messages that appear spurious but are actually correct. If the bad block is already in use as an alternate, the " added bad block" message can appear twice for one block.

NAME

killall - kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

Killall is is a procedure used by **/etc/shutdown** to kill all active processes not directly related to the shut down procedure.

Killall is chiefly used to terminate all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

Killall sends *signal* (see *kill*(1)) to all remaining processes not belonging to the above group of exclusions. If no *signal* is specified, a default of **9** is used.

FILES

/etc/shutdown

SEE ALSO

kill(1), ps(1), shutdown(1M), signal(2).

NAME

login - sign on

SYNOPSIS

login [name [env-var ...]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-d* to indicate an "end-of-file." (See *How to Get Started* at the beginning of this volume for instructions on how to dial up initially.)

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

exec login

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure */etc/profile* is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh*(1)) is initialized, and the file **.profile** in the working directory is excuted, if it exists. These specifications are found in the **/etc/passwd** file entry for the user. The name of the command interpreter is − followed by the last component of the interpreter's pathname (i.e., −**sh**). If this field in the password file is empty, then the default command interpreter, */bin/sh* is used.

The basic *environment* (see *environ*(5)) is initialized to:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
SHELL=*last-field-of-passwd-entry*
MAIL=/usr/mail/*your-login-name*
TZ=*timezone-specification*

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your login name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as

L*n*=xxx

NAME

lpadmin - configure the LP spooling system

SYNOPSIS

/usr/lib/lpadmin **−p** printer [options]
/usr/lib/lpadmin **−x** dest
/usr/lib/lpadmin **−d**[dest]

DESCRIPTION

Lpadmin configures LP spooling systems to describe printers, classes and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs and to change the system default destination. *Lpadmin* may not be used when the LP scheduler, *lpsched*(1M), is running, except where noted below.

Exactly one of the **−p**, **−d** or **−x** options must be present for every legal invocation of *lpadmin*.

−d[*dest*] makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when *lpsched*(1M) is running. No other *options* are allowed with **−d**.

−x*dest* removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. No other *options* are allowed with **−x**.

−p*printer* names a *printer* to which all of the *options* below refer. If *printer* does not exist then it will be created.

The following *options* are only useful with **−p** and may appear in any order. For ease of discussion, the printer will be refered to as *P* below.

−c*class* inserts printer *P* into the specified *class*. *Class* will be created if it does not already exist.

−e*printer* copies an existing *printer's* interface program to be the new interface program for *P*.

−h indicates that the device associated with *P* is hardwired. This *option* is assumed when creating a new printer unless the **−l** *option* is supplied.

−i*interface* establishes a new interface program for *P*. *Interface* is the path name of the new program.

−l indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*, disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.

−m*model* selects a model interface program for *P*. *Model* is one of the model interface names supplied with the LP software (see *Models* below).

−r*class* removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then the *class* will be removed.

−v*device* associates a new *device* with printer *P*. *Device* is the pathname of a file that is writable by the LP administrator, *lp*. Note that there is nothing to stop an administrator from associating the same *device* with more than one *printer*. If only the **−p** and **−v** *options* are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions.

When creating a new printer, the **−v** option and one of the **−e**, **−i** or **−m** options must be supplied. Only one of the **−e**, **−i** or **−m** options may be supplied. The **−h** and **−l** keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters **A-Z**, **a-z**, **0-9** and **_** (underscore).

Models.

Model printer interface programs are supplied with the LP software. They are shell procedures which interface between *lpsched* and devices. All models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **−m**. Alternatively, LP administrators may modify copies of models and then use *lpadmin* **−i** to associate them with printers. The following list describes the *models* and lists the options which they may be given on the *lp* command line using the **−o** keyletter:

dumb interface for a line printer without special functions and protocol. Form feeds are assumed. This is a good model to copy and modify for printers which do not have models.

1640 Diablo 1640 terminal running at 1200 baud, using XON/XOFF protocol. Options:

−12 12-pitch (10-pitch is the default)
−f don't use the *450*(1) filter. The output has been pre-processed by either *450*(1) or the *nroff* 450 driving table.

hp Hewlett Packard 2631A line printer at 2400 baud. Options:

−c compressed print
−e expanded print

prx Printronix P300 printer using XON/XOFF protocol at 1200 baud.

EXAMPLES

1. Assuming there is an existing Hewlett Packard 2631A line printer named *hp2*, it will use the **hp** model interface after the command:

```
/usr/lib/lpadmin -php2 -mhp
```

2. To obtain compressed print on *hp2*, use the command:

   ```
   lp -dhp2 -o-c files
   ```

3. A Diablo 1640 printer called *st1* can be added to the LP configuration with the command:

   ```
   /usr/lib/lpadmin -pst1 -v/dev/tty20 -m1640
   ```

4. An *nroff* document may be printed on *st1* in any of the following ways:

   ```
   nroff -T450 files | lp -dst1 -of
   nroff -T450-12 files | lp -dst1 -of
   nroff -T37 files | col | lp -dst1
   ```

5. The following command prints the password file on *st1* in 12-pitch:

   ```
   lp -dst1 -o12 /etc/passwd
   ```

   NOTE: the **−12** option to the **1640** model should never be used in conjunction with *nroff*.

FILES

/usr/spool/lp/*

SEE ALSO

450(1), accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1).

NAME

lpsched, lpshut, lpmove - start/stop the LP request scheduler and move requests

SYNOPSIS

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove requests dest
/usr/lib/lpmove dest1 dest2

DESCRIPTION

Lpsched schedules requests taken by *lp*(1) for printing on line printers.

Lpshut shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after *lpsched* is started again. All LP commands perform their functions even when *lpsched* is not running.

Lpmove moves requests that were queued by *lp*(1) between LP destinations. This command may be used only when *lpsched* is not running.

The first form of the command moves the named *requests* to the LP destination, *dest*. *Requests* are request ids as returned by *lp*. The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp* will reject requests for *dest1*.

Note that *lpmove* never checks the acceptance status (see *accept*(1M)) for the new destination when moving requests.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpstat(1).

NAME

mkfs - construct a file system

SYNOPSIS

/etc/mkfs special blocks[:inodes] [gap blocks/cyl]
/etc/mkfs special proto [gap blocks/cyl]
/etc/mkfs special

DESCRIPTION

Mkfs constructs a file system by writing on the special file according to the directions found in the remainder of the command line. If the second argument is given as a string of digits, *mkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* disk blocks the file system will occupy. The boot program is left uninitialized. If the optional number of inodes is not given, the default is the number of *logical* blocks divided by 4.

If the second argument is a file name that can be opened, *mkfs* assumes it to be a prototype file *proto*, and will take its directions from that file. The prototype file contains tokens separated by spaces or new-lines. The first token is the name of a file to be copied onto block zero as the bootstrap program. The second token is a number specifying the size of the created file system in *physical* disk blocks. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of inodes in the file system. The maximum number of inodes configurable is 65500. The next set of tokens comprise the specification for the root file. 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 path name 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, *mkfs* 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:

/stand/*diskboot*
4872 110
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
        $
$
```

The third form of the command syntax is recommended, since it needs no parameters, just special file.

The *default* will be used if the supplied *gap* and *blocks/cyl* are considered illegal values or if a short argument count occurs.

SEE ALSO

dir(4), fs(4).

BUGS

If a prototype is used, it is not possible to initialize a file larger than 64K bytes, nor is there a way to specify links.

NAME

mknod - build special file

SYNOPSIS

/etc/mknod name **c** | **b** major minor
/etc/mknod name **p**

DESCRIPTION

Mknod makes a directory entry and corresponding i-node for a special file. The first argument is the *name* of the entry. In the first case, 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), which may be either decimal or octal.

The assignment of major device numbers is specific to each system. They have to be dug out of the system source file **conf.c**.

Mknod can also be used to create fifo's (a.k.a named pipes) (second case in *SYNOPSIS* above).

SEE ALSO

mknod(2).

NAME

mount, umount - mount and dismount file system

SYNOPSIS

/etc/mount [special directory [**−r**]]

/etc/umount special

DESCRIPTION

Mount announces to the system that a removable file system is present on the device *special*. The *directory* must exist already; it becomes the name of the root of the newly mounted file system.

These commands maintain a table of mounted devices. If invoked with no arguments, *mount* prints the table.

The optional last argument indicates that the file is to be mounted read-only. Physically write-protected and magnetic tape file systems must be mounted in this way or errors will occur when access times are updated, whether or not any explicit write is attempted.

Umount announces to the system that the removable file system previously mounted on device *special* is to be removed.

FILES

/etc/mnttab mount table

SEE ALSO

setmnt(1M), mount(2), mnttab(4).

DIAGNOSTICS

Mount issues a warning if the file system to be mounted is currently mounted under another name.

Umount complains if the special file is not mounted or if it is busy. The file system is busy if it contains an open file or some user's working directory.

BUGS

Some degree of validation is done on the file system, however it is generally unwise to mount garbage file systems.

NAME

ncheck - generate names from i-numbers

SYNOPSIS

/etc/ncheck [**−i** numbers] [**−a**] [**−s**] [file-system]

DESCRIPTION

Ncheck with no argument generates a path name 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

fsck(1M), sort(1).

DIAGNOSTICS

When the file system structure is improper, **??** denotes the "parent" of a parentless file and a path name beginning with ... denotes a loop.

NAME

rc - system initialization shell script

SYNOPSIS

/etc/rc

DESCRIPTION

This shell procedure is executed via */etc/inittab* by *init*(1M) when the system state is changed.

The *rc* procedure clears the mounted file system table, */etc/mnttab* (see *mnttab*(4)), performs all the necessary consistency checks to prepare the system to change into multi-user mode.

The *rc* procedure starts all system daemons before the terminal lines are enabled. In addition, file systems are mounted and accounting, window, status, and telephony management is started.

SEE ALSO

init(1M), shutdown(1M), inittab(4).

NAME

reboot - reboot the system

SYNOPSIS

/etc/reboot

DESCRIPTION

Reboot causes the processor to enter its system bootstrap code thereby rebooting the system.

Reboot will enter the boot sequence immediately, without flushing the internal system buffers. It must be used with extreme caution.

SEE ALSO

sync(1M).

NAME

setmnt - establish mount table

SYNYOPSIS

/etc/setmnt

DESCRIPTION

Setmnt creates the /etc/mnttab table (see mnttab(4)), which is needed for both the mount(1M) and umount commands. Setmnt reads standard input and creates a mnttab entry for each line. Input lines have the format:

filesys node

where filesys is the name of the file system's special file (e.g., "rfp*") and node is the root name of that file system. Thus filesys and node become the first two strings in the mnttab(4) entry.

FILES

/etc/mnttab

SEE ALSO

mnttab(4).

BUGS

Evil things will happen if filesys or node are longer than 10 characters.

Setmnt silently enforces an upper limit on the maximum number of mnttab entries.

NAME

shutdown - terminate all processing

SYNOPSIS

/etc/shutdown

DESCRIPTION

Shutdown is a shell script that accomplishes the following:

1. Kills all processes (killall)
2. Stops the lp scheduler (lpshut)
3. Transfers control to /etc/profile (init), which recognizes that it is in single-user mode and reboots.

SEE ALSO

mount(1M), sync(1).

NAME

uuclean - uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [options]

DESCRIPTION

Uuclean will scan the spool directory for files with the specified prefix and delete all those which are older than the specified number of hours.

The following options are available.

−d*directory* Clean *directory* instead of the spool directory.

−p*pre* Scan for files with *pre* as the file prefix. Up to 10 **−p** arguments may be specified. A **−p** without any *pre* following will cause all files older than the specified time to be deleted.

−n*time* Files whose age is more than *time* hours will be deleted if the prefix test is satisfied. (default time is 72 hours)

−w*file* The default action for *uuclean* is to remove files which are older than a specified time (see **−n** option). The **−w** option is used to find those files older than *time* hours, however, the files are not deleted. If the argument *file* is present the warning is placed in *file*, otherwise, the warnings will go to the standard output.

−s*sys* Only files destined for system *sys* are examined. Up to 10 **−s** arguments may be specified.

−m*file* The **−m** option sends mail to the owner of the file when it is deleted. If a *file* is specified then an entry is placed in *file*.

This program is typically started by *cron*(1M).

FILES

/usr/lib/uucp	directory with commands used by *uuclean* internally
/usr/spool/uucp	spool directory

SEE ALSO

cron(1M), uucp(1C), uux(1C).

NAME

uusub - monitor uucp network

SYNOPSIS

/usr/lib/uucp/uusub [options]

DESCRIPTION

Uusub defines a *uucp* subnetwork and monitors the connection and traffic among the members of the subnetwork. The following options are available:

−**a***sys*	Add *sys* to the subnetwork.
−**d***sys*	Delete *sys* from the subnetwork.
−**l**	Report the statistics on connections.
−**r**	Report the statistics on traffic amount.
−**f**	Flush the connection statistics.
−**u***hr*	Gather the traffic statistics over the past *hr* hours.
−**c***sys*	Exercise the connection to the system *sys*. If *sys* is specified as **all**, then exercise the connection to all the systems in the subnetwork.

The meanings of the connections report are:

sys #call #ok time #dev #login #nack #other

where *sys* is the remote system name, *#call* is the number of times the local system tries to call *sys* since the last flush was done, *#ok* is the number of successful connections, *time* is the latest successful connect time, *#dev* is the number of unsuccessful connections because of no available device (e.g. ACU), *#login* is the number of unsuccessful connections because of login failure, *#nack* is the number of unsuccessful connections because of no response (e.g. line busy, system down), and *#other* is the number of unsuccessful connections because of other reasons.

The meanings of the traffic statistics are:

sfile sbyte rfile rbyte

where *sfile* is the number of files sent and *sbyte* is the number of bytes sent over the period of time indicated in the latest *uusub* command with the −**u***hr* option. Similarly, *rfile* and *rbyte* are the numbers of files and bytes received.

The command:

uusub -c all -u 24

is typically started by *cron*(1M) once a day.

FILES

/usr/spool/uucp/SYSLOG	system log file
/usr/lib/uucp/L_sub	connection statistics
/usr/lib/uucp/R_sub	traffic statistics

SEE ALSO

uucp(1C), uustat(1C).

NAME

volcopy, labelit - copy file systems with label checking

SYNOPSIS

/etc/volcopy [options] fsname special1 volname1 special2 volname2

/etc/labelit special [fsname volume [**−n**]]

DESCRIPTION

Volcopy makes a literal copy of the file system using a blocksize matched to the device. *Options* are:

−a invoke a verification sequence requiring a positive operator response instead of the standard 10 second delay before the copy is made,

−s (default) invoke the **DEL if wrong** verification sequence.

Other *options* are used only with tapes:

−bpidensity bits-per-inch (i.e., **800/1600/6250**),
−feetsize size of reel in feet (i.e., **1200/2400**),
−reelnum beginning reel number for a restarted copy,
−buf use double buffered I/O.

The program requests length and density information if it is not given on the command line or is not recorded on an input tape label. If the file system is too large to fit on one reel, *volcopy* will prompt for additional reels. Labels of all reels are checked. Tapes may be mounted alternately on two or more drives.

The *fsname* argument represents the mounted name (e.g.: **root**, **u1**, etc.) of the filsystem being copied.

The *special* should be the physical disk section or tape (e.g.: **/dev/rdsk15**, **/dev/rmt0**, etc.).

The *volname* is the physical volume name (e.g.: **pk3**, **t0122**, etc.) and should match the external label sticker. Such label names are limited to six or fewer characters. *Volname* may be − to use the existing volume name.

Special1 and *volname1* are the device and volume from which the copy of the file system is being extracted. *Special2* and *volname2* are the target device and volume.

Fsname and *volname* are recorded in the last 12 characters of the superblock (**char fsname[6], volname[6];**).

Labelit can be used to provide initial labels for unmounted disk or tape file systems. With the optional arguments omitted, *labelit* prints current label values. The **−n** option provides for initial labeling of new tapes only (this destroys previous contents).

FILES

/etc/log/filesave.log a record of file systems/volumes copied

SEE ALSO

fs(4).

BUGS

Only device names beginning **/dev/rmt** (on DEC systems) or

/dev/rtp (on 3B20S systems) are treated as tapes.

NAME

wall - write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message to all currently logged in users preceded by:

Broadcast Message from ...

It is used to warn all users, typically prior to shutting down the system.

The sender must be super-user to override any protections the users may have invoked (see *mesg*(1)).

FILES

/dev/tty*

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

"Cannot send to ..." when the open on a user's tty file fails.

NAME

whodo - who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

Whodo produces merged, reformatted, and dated output from the *who*(1) and *ps*(1) commands.

SEE ALSO

ps(1), who(1).

NAME

intro - introduction to special files

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals and UNIX device drivers. The names of the entries are generally derived from names for the hardware, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding UNIX device driver are discussed where applicable.

BUGS

While the names of the entries *generally* refer to vendor hardware names, in certain cases these names are seemingly arbitrary for various historical reasons.

NAME

err - error-logging interface

DESCRIPTION

Minor device 0 of the *err* driver is the interface between a process and the system's error-record collection routines. The driver may be opened only for reading by a single process with superuser permissions. Each read causes an entire error record to be retrieved; the record is truncated if the read request is for less than the record's length.

FILES

/dev/error special file

SEE ALSO

errdemon(1M).

NAME

escape - output escape codes for bitmap windows

DESCRIPTION

This lists the escape sequences honored by the /dev/window device (see window(7)). Pn refers to a numeric parameter (which defaults to 1). Ps refers to a selective parameter which defaults to zero.

Name Sequence Operation

Cursor Positioning

BS\010Backspace 1 column. No-op in column 1 HT\011Move to next multiple of 8 column LF\012Down 1 line, scrolling as necessary VT\013See LF (above) FF\014See LF (above) CR\015Cursor to column 1

NELESC EPosition to column 1 of next line RIESC MNegative line feed (scroll down at top) INDESC DSee LF (above)

CUUESC [Pn ACursor up Pn lines CUDESC [Pn BCursor down Pn lines CUFESC [Pn CCursor forward Pn columns CUBESC [Pn DCursor backward Pn columns CHAESC [Pn GMove cursor to column Pn CUPESC [P1 ; Pc HCursor position to Pl, Pc (1,1 = home) HVPESC [P1 ; Pc fSee CUP (above)

Scrolling, Deleting, Inserting, and Erasing

SUESC [Pn SScroll entire display up Pn lines SDESC [Pn TScroll entire display down Pn lines

DCHESC [Pn PDelete Pn Positions ICHESC [Pn @Insert Pn Positions

DLESC [Pn MDelete Pn lines ILESC [Pn LInsert Pn lines

ELESC [Ps KErase parts of line EL0ESC [0 KErase cursor to EOL EL1ESC [1 KErase BOL to cursor EL2ESC [2 KErase entire line

EDESC [Ps JErase parts of display ED0ESC [0 JErase cursor to EOD ED1ESC [1 JErase BOD to cursor ED2ESC [2 JErase entire display (clear)

Select Graphic Rendition

SGRESC [Ps;Ps;...mSelect graphic rendition (attribute) SGR0ESC [0 mSelect normal attribute SGR2ESC [2 mSelect faint attribute SGR4ESC [4 mSelect underline attribute SGR7ESC [7 mSelect reverse video attribute SGR9ESC [9 mSelect struck-out attribute (ISO) SGR10ESC [10 mSelect the G0 character set SGR11ESC [11 mSelect the G1 character set SGR12ESC [12 mSelect the G2 character set SGR17ESC [17 mSelect G7

Select Character Set

SGR10ESC [10 mSelect G0 character set SGR11ESC [11 mSelect G1 character set SGR12ESC [12 mSelect G2 character set SGR17ESC [17 mSelect G7

Cursor Visibility

CTVISESC [= Pv CSelect cursor visibility and anchoring
CTVIS0 ESC [= 0 CNormal (visible cursor, window follows)
CTVIS1 ESC [= 1 CInvisible (invisible cursor, window follows)

SEE ALSO

window(7), kbd(7), ANSI Specification X3.64.

NAME

gd - general driver for moving-head disks

DESCRIPTION

Gd provides a general interface to the RM05, RM80, RP04, RP05, RP06, and RP7 moving head disks. In addition to the capability of mixing these mediums on the same controller, the driver will handle up to four controllers.

The driver reads the disk hardware drive-type register to determine access partitioning and other drive dependent attributes. Thus, the manual entries describing the above disk drives should be used for information regarding that particular drive.

The configuration name of *disk* should be specified when generating a system with *config(1M)*.

FILES

/dev/rp*, /dev/rrp*

SEE ALSO

config(1M), master(4), hm(7), rm80(7), rp(7).

NAME

kbd - keyboard codes

DESCRIPTION

The following table gives the sequence of bytes sent for each key pressed or the system console.

" Legend" gives the keycap legend, " X" gives the sequence sent when that key alone is pressed, " s-x" when shift is pressed, " c-x" when ctrl (control) is pressed with it.

The " Type" field identifies the key type. SYS = no repeating, no caps lock, no num lock (e.g. EXIT)

REPT = repeating, no caps lock, no num lock (e.g. DLETE CHAR) ALPHA = repeating, caps lock, no num lock (e.g. A) NUM = no repeating, no caps lock, num lock (e.g. HOME)

NUMREPT = repeating, no caps lock, num lock (e.g. NEXT)

In the sequences sent, \E means ESC (0x1B),
means system special key class " n" . Following the digit " n" is the sequence, thus: BXY means special class 2, send ESC X Y. The classes are established via the WICCSYS window ioctl (see window (7)).

ILLK refers to an illegal key combination.

Legend	X	s-X	c-X	Flags
ClearLine	EOa	EOA	EOA	SYS
Rstrt/Ref	EOb	EOB	EOB	SYS
F1	EOc	EOC	ILLK	SYS
F2	EOd	EOD	ILLK	SYS
F3	EOe	EOE	ILLK	SYS
F4	EOf	EOF	ILLK	SYS
F5	EOg	EOG	ILLK	SYS
F6	EOh	EOH	ILLK	SYS
F7	EOi	EOI	ILLK	SYS
F8	EOj	EOJ	ILLK	SYS
Exit	EOK	EOK	EOK	SYS
Msg	EOl	EOL	EOL	SYS
Help	EOm	EOM	EOM	SYS
Creat	EOn	EON	EON	SYS
Save	EOo	EOO	EOO	SYS
Suspd	EOp	EOP	EOP	SYS
Rsume	EOq	EOQ	EOQ	SYS
Opts	EOr	EOR	EOR	SYS
Undo	EOS	EOS	EOS	SYS
Redo	EOt	EOT	EOT	SYS
Esc/DEL	33	177	ILLK	REPT
1	1	!	EPa	REPT
2	2	@	EPb	REPT
3	3	#	EPc	REPT
4	4	$	EPd	REPT
5	5	%	EPe	REPT
6	6	^	EPf	REPT
7	7	&	EPg	REPT
8	8	*	EPh	REPT

9		9	(	EPi	REPT
0		0	)	EPj	REPT
-		-	-	EPk	REPT
+	EPl REPT				
BACKSPACE		10	10	10	REPT
RESET/BREAK		ILLK	Ec	Ec	SYS
CMD		EOu	EOU	EOU	SYS
CLOSE/OPEN		EOv	EOV	EOV	SYS
CANCL		EOw	EOW	EOW	SYS
FIND		EOx	EOX	EOX	SYS
REPLAC		EOy	EOY	EOY	SYS
TAB		11	EOZ	EOZ	REPT
Q		q	Q	21	ALPHA
W		w	W	27	ALPHA
E		e	E	05	ALPHA
R		r	R	22	ALPHA
T		t	T	24	ALPHA
Y		y	Y	31	ALPHA
U		u	U	25	ALPHA
I		i	I	11	ALPHA
O		o	O	17	ALPHA
P		p	P	20	ALPHA
[		[	{	33	REPT
]		]	}	35	REPT
			|	34	REPT
`		`	~	00	REPT
PRINT		EOz	EOZ	EOZ	NUM
CLEAR/RFHSH		ENa	EOJ	EOJ	NUM
PAGE		ECU	ECV	ECV	NUM
MOVE		ENc	ENC	ENC	SYS
COPY		ENd	END	END	SYS
CAPLOCK		ILLK	ILLK	ILLK	SYS
A		a	A	01	ALPHA
S		s	S	23	ALPHA
D		d	D	04	ALPHA
F		f	F	06	ALPHA
G		g	G	07	ALPHA
H		h	H	10	ALPHA
J		j	J	12	ALPHA
K		k	K	13	ALPHA
L		l	L	14	ALPHA
;		;	:	ILLK	REPT
'		'	"	ILLK	REPT
RETURN		15	15	15	REPT
BEG		E9	ENB	ENB	NUM
HOME		ECH	ENM	ENM	NUM
END		EO	ENN	ENN	NUM
DLETE		ENe	ENE	ENE	SYS
DLETECHAR		ENf	ENF	ENF	REPT
L-SHIFT		ILLK	ILLK	ILLK	SYS
Z		z	Z	32	ALPHA
X		x	X	30	ALPHA
C		c	C	03	ALPHA
V		v	V	26	ALPHA

B	b	B	02	ALPHA
N	n	N	16	ALPHA
M	m	M	15	ALPHA
,	,	<	ILLK	REPT
/	/	?	ILLK	REPT
R-SHIFT	ILLK	ILLK	ILLK	SYS
ENTER	12	12	12	SYS
PREV	ENg	ENG	ENG	NUMREPT
ROLL/UP	ECA	ECT	ECT	NUMREPT
NEXT	ENH	ENH	ENH	NUMREPT
SLECT/MARK	ENi	ENI	ENI	SYS
INPUT/MODE	ENj	ENJ	ENJ	SYS
CTRL	ILLK	ILLK	ILLK	SYS
SPACE	40	40	40	REPT
CTRL	ILLK	ILLK	ILLK	SYS
NUMLOCK	ILLK	ILLK	ILLK	SYS
<-	ECQ	ENK	ENK	NUMREPT
ROLL/ON	ECS	ECS	ECS	NUMREPT
->	EOC	ENL	ENL	NUMREPT

NAME

lp – line printer
rawlp – raw line printer- raw line printer

DESCRIPTION

Lp provides the interface to any standard Centronics line printer. When it is opened or closed, a suitable number of page ejects is generated. Bytes written are printed.

An internal parameter within the driver determines whether or not the device is treated as having a 96- or 64-character set. In 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. A new-line that extends over the end of a page is turned into a form-feed. The default line length is 132 characters, no indentation and lines per page is 66. Lines longer than the line length minus the indent (i.e. 132 characters, using the above defaults) are truncated.

Two *ioctl*(2) system calls are available:

```
#include <sys/lprio.h>
ioctl (fildes, command, arg)
struct lprio *arg;
```

The *commands* are:

LPRGET Get the current indent, columns per line, and lines per page and store in the *lprio* structure referenced by **arg**.

LPRSET Set the current indent, columns per line, and lines per page from the structure referenced by **arg**.

Thus, indent, page width and page length can be set with an external program.

Rawlp provides a direct interface to the parallel printer with no modification of the data sent.

FILES

/dev/lp
/dev/raw/lp

SEE ALSO

lp(1).

NAME

mem, kmem - core memory

DESCRIPTION

Mem is a special file that is an image of the core 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 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 the PDP-11, the I/O page begins at location 0160000 of *kmem* and per-process data for the current process begins at 0140000.

FILES

/dev/mem, /dev/kmem

BUGS

On the PDP-11, memory files are accessed one byte at a time, an inappropriate method for some device registers.

NAME

null - the null file

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

termio - general terminal interface

DESCRIPTION

All of the asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files; they are opened by *getty* and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(2). A process can break this association by changing its process group using *setpgrp*(2).

A terminal associated with one of these files 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 full, 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. When the input limit is reached, all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, 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, erase and kill processing is normally done. By default, the character # erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character @ kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The erase and kill characters may be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII DEL) generates an *interrupt* signal which is sent to all processes with the associated

control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).

QUIT (Control-| or ASCII FS) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.

ERASE (#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (@) deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF (Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.

NL (ASCII LF) is the normal line delimiter. It can not be changed or escaped.

EOL (ASCII NUL) is an additional line delimiter, like NL. It is not normally used.

STOP (Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START (Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a *hangup* signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any subsequent read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If 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.

Several *ioctl*(2) system calls apply to terminal files. The primary calls use the following structure, defined in <**termio.h**>:

```
#define   NCC       8
struct    termio {
          unsigned  short   c_iflag;      /* input modes */
          unsigned  short   c_oflag;      /* output modes */
          unsigned  short   c_cflag;      /* control modes */
          unsigned  short   c_lflag;      /* local modes */
          char              c_line;       /* line discipline */
          unsigned  char    c_cc[NCC];    /* control chars */
};
```

The special control characters are defined by the array *c_cc*. The relative positions and initial values for each function are as follows:

0	INTR	DEL
1	QUIT	FS
2	ERASE	BS
3	KILL	@
4	EOF	EOT
5	EOL	NUL
6	reserved	
7	reserved	

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map upper-case to lower-case on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three character sequence:

0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character, will restart output which has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is all bits clear.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:

VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

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. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2 four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud

B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
B19200	0000016	19200 baud
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise modem control is assumed.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g. 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

for:	*use*:
`	\'
|	\!
~	\^
\	\\
{	\(
}	\)

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is

useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The primary *ioctl*(2) system calls have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are:

TCGETA — Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.

TCSETA — Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.

TCSETAW — Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.

TCSETAF — Wait for the output to drain, then flush the input queue and set the new parameters.

Additional *ioctl*(2) calls have the form:

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

The commands using this form are:

TCSBRK — Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).

TCXONC — Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.

TCFLSH — If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

FILES

/dev/tty*

SEE ALSO

stty(1), ioctl(2), window(7).

NAME

tty - controlling terminal interface

DESCRIPTION

The file **/dev/tty** is, in each process, a synonym for the control terminal or window associated with the process group of that process, if any. It is useful for programs or shell sequences 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 the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

FILES

/dev/tty
/dev/tty*

SEE ALSO

termio(7), window(7).

NAME

window - bitmap windows

DESCRIPTION

Windows are opened and closed via open(2) and close(2). To open a new window, the program opens the device "/dev/window". The kernel will bind a new window to the returned file descriptor. The window number can be obtained in the minor device field of a subsequent stat(2) call on the file descriptor.

The opening of a window creates a dimensionless window which does not occupy any screen space. The window size is subsequently established in one of two ways:

Implicitly. If the program does a read(2), write(2), or certain ioctl(2) calls on the window, the kernel will automatically set the window size to a default (currently full-screen) and then proceed with the particular system call.

Explicitly. If the program does an ioctl(wd,WIOCSETD,&params), then the window size is taken from the params given (see WIOCSETD, below). This is the preferred mechanism for establishing a window's size as it permits the creation of windows of arbitrary dimensions.

In addition, a program may open "/dev/wN" where "N" is the window number (minor device number) of an already-existing window. This permits multiple applications to open the same window.

When a window is created, it automatically becomes the current, active window. As soon as dimensions are established, it will be displayed at the front of the screen, unobscured by any other windows. In addition, the default system font is loaded into font slot 0.

Any window may be closed via the close(2) system call. When the last of potentially multiple programs closes the window, its space on the display is removed.

read and write calls are used to perform I/O on the window. read reads characters from the keyboard and returns them to the process. write writes characters to the display.

The data to be read or written is in ANSI X3.64 / 7-bit ASCII code. Output sequences exist to control cursor motion, insertion, deletion, erasure, fonts, and various mode settings. Input consists of characters and control sequences.

In addition, all the standard facilities of the unix "tty" driver are available, control of echo, raw mode, new-line, padding, interrupt/quit/kill characters, etc.

Any uncovered window can write to the display without blocking, regardless of whether or not it is the active window. For example, this allows status processes to output system update messages even though they are not the current window.

Reading does not explicitly block non-active windows. Rather, they are allowed to read any input data which was accumulated (typed ahead) when they were last active. When this data is

exhausted, the process will block on the next read.

In addition to all tty ioctls (TIOCxxxx), the window device supports its own ioctls which control window functions:

ioctl(wd,WIOCGETD,&uwdata)
ioctl(wd,WIOCSETD,&uwdata)

These calls allow the program to get and set (respectively) parameters about the window. The uwdata structure has the following form:

```
struct uwdata          /* user window          */
{
unsigned shortuw_x;/* upper-left-corner x) */
unsigned shortuw_y;/* upper-left-corner y  */
unsigned shortuw_width;/* width (pixels)     */
unsigned shortuw_height;/* height (pixels)   */
unsigned shortuw_uflags;/* various flags     */
unsigned charuw_hs;/* horizontal size (RO) */
unsigned charuw_vs;/* vertical size (RO)   */
unsigned charuw_baseline;/* baseline (RO)   */
};
```

Where uw_x and uw_y are the pixel coordinates of the upper left-hand corner of the window. If the window has borders (see below), then uw_x and uw_y specify the upper left-hand corner of the border. uw_width and uw_height are the width and height of the window (in pixels). The width and height never include the border.

uw_vs is the vertical spacing for characters in pixels. uw_hs is the horizontal spacing for characters. uw_baseline is the vertical offset from 0,0 to the baseline position of a character. The baseline is an imaginary line around which characters are drawn - much like the ruling on a conventional lined note pad.

uw_vs, uw_hs, and uw_baseline are read-only parameters (they are ignored on a WIOCSETD call). They are computed dynamically by the kernel based on the most extreme character loaded into any font in the window's palette (see WIOCLFONT). All character-oriented cursor motion uses these values to translate character addresses to pixel addresses. In addition, if the VCWIDTH flag is off in uw_uflags (see below), all characters are displayed in an imaginary cell which is uw_hs X uw_vs pixels. In this mode, existing unix programs (such as vi) can readily address the display as a " normal" character terminal, even if multiple " fancy" proportional-space fonts are used instead of the nominal 9 X 12 system font. Smarter applications use pixel addressing for glyph placement and set the VCWIDTH flag enabling proportional glyph placement.

In addition, uw_hs and uw_vs control the size of the character cursor in the window. In the future, applications will have far more control over the appearance and size of both the character and mouse cursor.

The uw_uflags field contains flags:

```
#define NBORDER0x1/* borderless          */
#define VCWIDTH0x2/* variable chr spacing */
#define BORDHSCROLL0x4/* border hscroll icons */
#define BORDVSCROLL0x8/* border vscroll icons */
#define BORDHELP0x10/* border help patch    */
#define BORDCANCEL0x20/* border cancel patch  */
#define BORDRESIZE0x40/* border re-size patch */
#define BORDWNUM0x80/* border window num     */
#define UNCOVERED0x100/* uncovered (RO)       */
#define KBDWIN0x200/* keyboard (RO)         */
#define NOCLEAR0x400/* don't clear window   */
```

NBORDER turns off the window's borders by forcing the four margin parameters to zero. This bit vestigial and will probably be deleted shortly. Callers should explicitly zero the four margin parameters to eliminate window borders.

VCWIDTH, when set, enables proportional character placement. When set, the cursor is advanced by the displayed character's horizontal increment, rather than the window-wide maximum character width (readable as uw_hs).

The BORDxxxx flags enable the corresponding border icons. HSCROLL enables the horizontal scrolling icons, VSCROLL the vertical. HELP, CANCEL and RESIZE enable the help, cancel and window re-sizing icons. When the mouse is clicked on an enabled icon, the corresponding keyboard sequence is transmitted.

UNCOVERED is set whenever the window is totally visible. This flag is read-only.

KBDWIN is set when this window is the current (keyboard) window. This flag is read-only.

Setting NOCLEAR prvents the system from clearing out the contents of the window upon its creation. This flag is intended for use only by window management system software.

ioctl(wd,WIOCSELECT)

This ioctl causes the window " wd" to become the current keyboard (active) window. This call is normally issued only by window management software, not by applications.

ioctl(wd,WIOCREAD,&pixmap)

This ioctl causes the pixel image of the entire display to be " dumped" into the memory at pixmap. pixmap should be 15660 unsigned shorts arranged as 348 rows each containing 45 unsigned shorts. The least significant bit of the first short in the array contains the upper-left-hand display pixel.

ioctl(wd,WIOCSETTEXT,&ut)
ioctl(wd,WIOCGETTEXT,&ut)

This ioctl allows the application to associate textual data with a window's two screen-labeled key (SLK) lines, command line, and prompt line. These four lines are the bottom for on the display and switch with the selected window. In addition, the application may program the window's label line (the top border) and a

non-displayed " user" line which is generally used to describe the window to window management software.

The ut (user text) structure has the following form:

```
struct utdata          /* user text data      */
{
short   ut_num;        /* number (see above)  */
char    ut_text[WTXTLEN]; /* text             */
};
```

ut_num is the text item number (WTXTSLK1, WTXTSLK2, WTXTPROMT, WTXTCMD, WTXTLABEL, WTXTUSER) and ut_text contains the null-terminated data.

ioctl(wd,WIOCSYS,num)
ioctl(wd,WIOCGSYS,num)

The WIOCSYS ioctl declares window " wd" to be system window number " num". WIOCGSYS returns the process group associated with an existing system window number " num". There are currently three system windows. Each system window " owns" a number of keys on the keyboard, regardless of the currently selected window. If one of these keys is struck, it is queued for reads in the appropriate system window and is not sent to the currently active one. No other action is taken (specifically, the system window is not selected). The following table lists the special system keys:

SYSWMGR(0) Window Manager:
Suspd, s-Suspd, Rsume, s-Rsume, s-Print

SYSPMGR(1) Phone Manager:
All shifted function keys (F1-F8)

SYSSMGR(2) Status Manager:
Msg, s-Msg

ioctl(wd,WIOCGETMOUSE,&umdata)
ioctl(wd,WIOCSETMOUSE,&umdata)

These ioctl control the mouse. Once enabled, the mouse sends " reports" to the application in the same stream as keyboard input. If the mouse has not been enable, no reports are sent.

The umdata structure is as follows:

```
#define MSDOWN0x1/* buttons go down      */
#define MSUP0x2/* buttons go up         */
#define MSIN0x4/* mouse is in rectangle*/
#define MSOUT0x8/* mouse is outside rect*/

structumdata/* mouse data          */
{
charum_flags;/* wakeup flags        */
shortum_x;/* motion rectangle      */
shortum_y;
```

```
shortum_w;
shortum_h;
structicon *um_icon;/* ptr to icon        */
};
```

The um_flags field contains flags which are used to determine when mouse reports shouls be sent. MSUP and MSDOWN cause reports to be sent when buttons go up or down, respectively. MSIN and MSOUT cause reports to be sent when the mouse is located within (MSIN) or outside (MSOUT) the rectangular region specified with um_x, um_y, um_w, um_h (x,y, width and height, in pixels).

um_icon is an optional pointer to an icon structure (see font(4)). This icon will be used as the mouse-track cursor. If um_icon is zero, the standard system mouse-track is used.

Mouse reports take the form:

ESC [? {x-pos} ; {y-pos} ; {buttons} ; {reason} M

Where ESC is the ASCII "escape" character (033) followed by left square-bracket, question mark and four ASCII decimal numbers separated by semicolons. The sequence is terminated by a capitol 'M' character.

{x-pos} and {y-pos} are the x and y positions of the mouse-track relative to the window. {buttons} is single digit character in the range "0" (060) to "7" (067) representing three mouse buttons as bits. The most significant bit is the left-most mouse button.

{reason} is an ASCII decimal string explaining what event caused the mouse report. The number consists of combinations of the MSUP, MSDOWN, MSIN, and MSOUT bits (above). Whenever a mouse report is generated due to MSIN or MSOUT, the enable bit for the condition is clear. These wakeup conditions are one-shot. Whenever a WIOCSETMOUSE ioctl is issued with the MSIN or MSOUT bits set in um_flags, a check is made to see whether an immediate report is necessary because the mouse already satisfies the wakeup condition.

Some typical mouse reports are:

ESC [? 100 ; 20 ; 1 ; 1 M
: The reason is MSDOWN (1), the button state is 1 (right-most button down, others up). The mouse-track is at 100,20.

ESC [? 10 ; 54 ; 0 ; 4 M
: The reason is MSIN (4), there are no buttons down (0), the mouse-track is at 10,54 which is within the bounds defined by the rectangle in the last WIOCSETMOUS ioctl.

ioctl(wd,WIOCRASTOP,&urdata)

The WIOCRASTOP ioctl provides user programs with direct access to a window's pixel data. This " raster operation" ioctl is controlled by the urdata structure:

```
structurdata/* user rastop data     */
{
unsigned short*ur_srcbase;/* ptr to source data   */
unsigned shortur_srcwidth;/* number bytes/row     */
unsigned short*ur_dstbase;/* ptr to dest data     */
unsigned shortur_dstwidth;/* number bytes/row     */
unsigned shortur_srcx;/* source x                 */
unsigned shortur_srcy;/* source y                 */
unsigned shortur_dstx;/* destination x            */
unsigned shortur_dsty;/* destination y            */
unsigned shortur_width;/* width                   */
unsigned shortur_height;/* height                 */
charur_srcop;/* source operation     */
charur_dstop;/* destination operation*/
unsigned short*ur_pattern;/* pattern pointer     */
};

/* rastop source operators      */
#define SRCSRC0/* source                 */
#define SRCPAT1/* pattern                */
#define SRCAND2/* source and pattern     */
#define SRCOR3/* source or pattern       */
#define SRCXOR4/* source xor pattern      */

/* rastop destination operators */
#define DSTSRC0/* srcop(src)             */
#define DSTAND1/* srcop(src) and dst     */
#define DSTOR2/* srcop(src) or dst       */
#define DSTXOR3/* srcop(src) xor dst     */
#define DSTCAM4/* not(srcop) and dst     */
```

The first four members of the structure determine the memory addresses of the source and destination planes. srcbase and dstbase may point to the address of the first short of an arbitrarily-sized array of shorts. Each row of pixels consists of srcwidth (or dstwidth) number of bytes from this array. Thus, the first pixel row exists from srcbase to ((char *)srcbase) + srcwidth. Within each short, the least significant bit is the left-most when displayed on the screen.

Alternatively, srcbase and/or dstbase may contain 0, in which case the source or destination is assumed to be the window specified by the first arg to the ioctl (wd). The caller need not supply any value for the srcwidth if srcbase is 0, nor dstwidth if dstbase is zero. It is therefore possible to perform raster operations from user space to user space, user space to screen, screen to user space, or screen to screen.

The next four members of the urdata struct contain pixel addresses within the specified pixel plane. 0,0 is always the upper-left-hand corner of the display. Note that raster operation are completely aware of the problems associated with overlapping rectangles: the memory operations will be done front to back or back to front as necessary.

The width and height parameters give the rectangle's width and height in pixels.

The srcop (source operation) and dstop (destination operation) fields together determine the algorithm which will be applied to the two rectangles. The basic behavior of rastop conforms to the following vector description:

dst = dstop(srcop(src,pattern))

Where srcop and dstop are vector functions. There are five source operations. SRCSRC is the identify function whose value is the unmodified source rectangle itslef. SRCPAT's value is that of the " pattern" (see below) and bears no relationship to the source. SRCOR is the inclusive or of the source and the pattern, SRCAND, the and, SRCXOR, the exclusive or.

DSTSRC is the identify function, returning the result of the source operation unchanged. DSTAND is the and of the destination with the result of the source, DSTOR is the inclusive or, and DSRXOR the exclusive or. DSTCAM and's the one's-complement of the source operation into the destination. DSTCAM the inverse of DSTOR: where DSTOR would turn on pixels, DSTCAM will turn them off.

The pattern field is required for SRCPAT, SRCAND, SRCOR, and SRCXOR operations only. It points to an array of 16 X 16 pixels arranged as 16 consecutive shorts. As with source and destination rectangles, the LSB of the first short in the vector corresponds to the upper-left-hand pixel of the pattern. Patterns are automatically aligned with the destination.

Since WIOCRASTOP is really an output operation, the process is blocked until the window is exposed. In addition, the raster operation waits for previously-output characters to appear on the screen before commencing.

ioctl(wd,WIOCLFONT,&ufdata)
ioctl(wd,WIOCUFONT,&ufdata)

These two ioctls control the loading and unloading of fonts for a particular window. Each window has 8 font " slots" which are addressable with ANSI X3.64 character strings (SGR, SI, SO, SS2, etc.). The WIOCLFONT call loads a particular font into a particular slot, automatically unloading any font previously loaded there. WIOCUFONT explicitly unloads a font from a slot. Loaded fonts tie up system resources (although the kernel will

automatically " share" identical fonts across multiple windows) so it is good practice to unload fonts when they are no longer needed. Note that the font in slot #0 is known as the " system font". At the time of this writing, the font slot #0 may be replaced by any valid font, but this is almost certain to change shortly as some software (including the kernel) may depend on being able to locate a fixed-pitch, ASCII, 9 X 12 font in a fixed location. If the font file is malformed, a -1 is returned from the loctl and errno is set to EBFONT.

The ufdata structure is very simple:

```
#define FNSIZE  60      /* font name size       */

struct ufdata           /* user font data       */
{
short   uf_slot;        /* slot number          */
char    uf_name[FNSIZE];/* font name (file name)*/
};
```

Where uf_slot is the font slot number (0-7) and uf_name is the path name where a suitably-formatted font file can be found. See section 5 for more information about fonts.

When a new font is loaded, the kernel checks to see if it contains any character more extreme than the one reflected in the current uw_hs, uw_vs, and uw_baseline variables. If it does, the three values are updated. When a font is unloaded, the kernel recomputes new values for uw_hs, uw_vs, and uw_baseline.

ioctl(wd,WIOCPGRP,dummy)

This ioctl sets the window's controlling process group to that of the process issuing the ioctl. It is especially useful in those cases where the parent has open'ed a new window which it wishes to give to the child. If the child does a setpgrp(2) call, it will be isolated from the parent's process group. If the parent does not issue this ioctl, the signals generated by interacting with child (e.g. SIGINTR) will go to the process group that opened the window (the parent), but the child will not see these signals becuase it has done the setpgrp(2) call.

The recommended sequence is:

```
open the window
fork()

        CHILD:
        setpgrp()
        dups and closes for stdin, out, err
        ioctl(0,WIOCPRGP)
        exec

        PARENT:
```

wait(), etc.

FILES

/dev/window*
/usr/include/sys/window.h

SEE ALSO

termio(7), font(4), tam(3T), wrastop(3T)

BUGS

The VCWIDTH flag is inoperative on the PC7300.