HP-UX Reference
Vol. 2: Sections 1M and 2

Manual Reorder No. 09000-90008

Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

July 1985...Edition 1. This manual replaces HP-UX Reference Manual 09000-90007 and documents HP-UX release 5.0 for HP 9000 Series 200, 300 and 500.

Release 5.0 reflects the following system changes:

- Sections 2 and 3 converged more closely to System V,
- Native Language Support added,
- Real-time features added,
- Process accounting added to Series 500,
- LAN, Virtual memory and window management added for Series 200 and 300,
- LAN option available for Series 200 and 300.
- Starbase graphics library added.

TABLE OF CONTENTS

1. Commands

Table of Contents

Table of Contents

1M. System Maintenance Utilities

Table of Contents

2. System Calls

Table of Contents

3. Subroutines

Table of Contents

4. Special Files

5. File Formats

6. Games

No games are currently supported.

7. Miscellaneous Facilities

9. Glossary

NAME

intro - introduction to system maintenance commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands that are used chiefly for system mainte
nance and administration purposes. The commands in this section should be used in conjunction
with other sections of the *HP-UX Reference* as well as the *HP-UX System Administrator Manual*
for your system.

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 indi cating the standard input.

SEE ALSO

getopt(1). getopt(3C).
HP UX Reference.
HP-UX System Administrator Manual.

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 cus-
tomarily 0 for successful execution and non zero to indicate troubles such as erroneous parame
ters, bad or inaccessible data, or other inability to cope with the task at hand. It is called vari-
ously "exit code", "exit status", or "return code", and is described only where special conventions
are involved.

BUGS

Unfortunately, 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

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral PC.

Native Language Support:
8-bit file names and data, customs, messages.

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
    acctdisk, acctdusg, accton, acctwtmp - overview of accounting and miscellaneous accounting com
    mands

SYNOPSIS
    /usr/lib/acct/acctdisk

    /usr/lib/acct/acctdusg [**-u** file] [**-p** file]

    /usr/lib/acct/accton [file]

    /usr/lib/acct/acctwtmp "reason"

HP-UX COMPATIBILITY
    Level: HP-UX/EXTENDED

    Origin: System V

DESCRIPTION
    Accounting software is structured as a set of tools (consisting of both C programs and shell pro
    cedures) that can be used to build accounting systems. *Acctsh*(1M) describes the set of shell pro
    cedures built on top of C programs.

    Connect time accounting is handled by various programs that write records into **/etc/wtmp**, as
    described in *utmp*(5). The programs described in *acctcon*(1M) convert this file into session and
    charging records, which are then summarized by *acctmerg*(1M).

    Process accounting is performed by the HP-UX system kernel. Upon termination of a process,
    one record per process is written to a file (normally **/usr/adm/pacct**). The programs in
    acctprc(1M) summarize this data for charging purposes; *acctcms*(1M) is used to summarize com-
    mand usage. Current process data may be examined using *acctcom*(1).

    Process accounting and connect time accounting (or any accounting records of the format
    described in *acct*(5)) can be merged and summarized into total accounting records by *acctmerg*
    (see **tacct** format in *acct*(5)). *Prtacct* (see *acctsh*(1M)) is used to format any or all accounting
    records.

    Acctdisk reads lines that contain user ID, login name, and number of disc blocks and convert
    them to total accounting records that can be merged with other accounting records.

    Acctdusg reads its standard input (usually from **find / -print**) and computes disc resource con
    sumption by login. If **-u** is given, records consisting of those file names for which *acctdusg*
    charges no one are placed in *file* (a potential source for finding users trying to avoid disc charges).
    If **-p** is given, *file* is the name of the password file. This option is not needed if the password file
    is **/etc/passwd.** (See *diskusg*(1M) for more details.)

    Accton with no optional file specified turns process accounting off. If *file* is given, it must be the
    name of an existing file to which the kernal appends process accounting records (see *acct*(2) and
    acct(5)).

    Acctwtmp writes a *utmp*(5) record to its standard output. The record contains a character string
    that describes the *reason* for writing the record. A record type of ACCOUNTING is assigned (see
    utmp(5)). *Reason* must be a string of 11 or less characters, numbers, $, or spaces. For example,
    the following are suggestions for use in reboot and shutdown procedures, respectively:

        acctwtmp `uname` >> /etc/wtmp
        acctwtmp "file save" >> /etc/wtmp

FILES
    /etc/passwd   used for login-name to user conversions
    /usr/lib/acct   holds all accounting commands listed in
                      sub-class 1M of this manual
    /usr/adm/pacct   current process accounting file

/ctc/wtmp   login/logoff history file

SEE ALSO

acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), diskusg(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).

Chapter 6, "System Accounting," of the *HP–UX System Administrator Manual.*

NAME

acctcms - command summary from per-process accounting records

SYNOPSIS

/usr/lib/acct/acctcms [options] files

HP–UX COMPATIBILITY

Level: HP–UX/EXTENDED

Origin: System V

DESCRIPTION

Acctcms reads one or more *files*, normally in the form described in *acct*(5). It adds all records for processes that executed identically–named commands, sorts them, and writes them to the standard output, normally using an internal summary format. The *options* are:

-a   Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore–minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, "hog factor", characters transferred, and blocks read and written, as in *acctcom*(1). Output is normally sorted by total kcore minutes.

-c   Sort by total CPU time, rather than total kcore minutes.

-j   Combine all commands invoked only once under "***other".

-n   Sort by number of command invocations.

-s   Any file names encountered hereafter are already in internal summary format.

-t   Process all records as total accounting records. The default internal summary format splits each field into prime and non–prime time parts. This option combines the prime and non–prime time parts into a single field that is the total of both, and provides upward compatibility with old style **acctcms** internal summary format records.

The following options may be used only with the –a option.

-p   Output a prime–time–only command summary.

-o   Output a non–prime (offshift) time only command summary.

When **-p** and **-o** are used together, a combination prime and non- prime time report is produced. All the output summaries will be total usage except number of times executed, CPU minutes, and real minutes which will be split into prime and non–prime.

A typical sequence for performing daily command accounting and for maintaining a running total is:

    acctcms file ... >today
    cp total previoustotal
    acctcms -s today previoustotal >total
    acctcms -a -s today

SEE ALSO

acct(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acctcom(1), acct(2), acct(5), utmp(5).

BUGS

Unpredictable output results if **-t** is used on new style internal summary format files, or if it is not used with old style internal summary format files.

NAME

acctcon1, acctcon2 - connect time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]

/usr/lib/acct/acctcon2

HP-UX COMPATIBILITY

Level: HP-UX/EXTENDED

Origin: System V

DESCRIPTION

Acctcon1 converts a sequence of login/logoff records read from its standard input to a sequence of records, one per login session. Its input should normally be redirected from **/etc/wtmp**. Its output is ASCII, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time. The *options* are:

-**p**   Print input only, showing line name, login name, and time (in both numeric and date/time formats).

-**t**   *Acctcon1* maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The -**t** flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.

-**l** *file*   *File* is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Hang-up, termination of *login*(1) and termination of the login shell each generate logoff records, so that the number of logoffs is often three to four times the number of sessions. See *init*(1M) and *utmp*(5).

-**o** *file*   *File* is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

Acctcon2 expects as input a sequence of login session records and converts them into total accounting records (see **tacct** format in *acct*(5)).

EXAMPLES

These commands are typically used as shown below. The file **ctmp** is created only for the use of *acctprc*(1M) commands:

acctcon1 -t -l lineuse -o reboots <wtmp | sort +1n +2 >ctmp

acctcon2 <ctmp | acctmerg >ctacct

FILES

/etc/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), init(1M), login(1), runacct(1M), acct(2), acct(5), utmp(5).

BUGS

The line usage report is confused by date changes. Use *wtmpfix* (see *fwtmp*(1M)) to correct this situation.

NAME
acctmerg - merge or add total accounting files

SYNOPSIS
/usr/lib/acct/acctmerg [options] [file] . . .

HP–UX COMPATIBILITY
Level: HP–UX/EXTENDED

Origin: System V

DESCRIPTION
Acctmerg reads its standard input and up to nine additional files, all in the **tacct** format (see *acct*(5)) or an ASCII version thereof. It merges these inputs by adding records whose keys (nor–mally user ID and name) are identical, and expects the inputs to be sorted on those keys. *Options* are:

-**a**   Produce output in ASCII version of **tacct**.
-**i**   Input files are in ASCII version of **tacct**.
-**p**   Print input with no processing.
-**t**   Produce a single record that totals all input.
-**u**   Summarize by user ID, rather than user ID and name.
-**v**   Produce output in verbose ASCII format, with more precise notation for floating point numbers.

EXAMPLES
The following sequence is useful for making ''repairs'' to any file kept in this format:

           acctmerg -v -a <file1 >file2
                   edit file2 as desired ...
           acctmerg -i <file2 >file1

SEE ALSO
acct(1M),   acctcms(1M),   acctcom(1),   acctcon(1M),   acctprc(1M),   acctsh(1M),   fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).

NAME

acctprc1, acctprc2 - process accounting

SYNOPSIS

/usr/lib/acct/acctprc1 [ctmp]

/usr/lib/acct/acctprc2

HP–UX COMPATIBILITY

Level: HP–UX/EXTENDED

Origin: System V

DESCRIPTION

Acctprc1 reads input in the form described by *acct*(5), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non–prime CPU time (tics), and mean memory size (in memory segment units). If **ctmp** is given, it is expected to contain a list of login sessions, in the form described in *acctcon*(1M), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in **ctmp** helps it distinguish among different login names that share the same user ID.

Acctprc2 reads records in the form written by *acctprc1*, summarizes them by user ID and name, then writes the sorted summaries to the standard output as total accounting records.

These commands are typically used as shown below:

        acctprc1 ctmp </usr/adm/pacct | acctprc2 >ptacct

FILES

/etc/passwd

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctsh(1M), cron(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).

BUGS

Although it is possible to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from *cron*(1M), for example. More precise conversion can be done by faking login sessions on the console via the *acctwtmp* program in *acct*(1M).

HARDWARE DEPENDENCIES

Series 500:

On the Series 500, memory segment units contain 512 bytes each; therefore, memory usage statistics are rounded up to 512–byte units.

NAME

chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, shu-
tacct, startup, turnacct - shell procedures for accounting

SYNOPSIS

/usr/lib/acct/chargefee login–name number

/usr/lib/acct/ckpacct [blocks]

/usr/lib/acct/dodisk [–o] [files ...]

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct number

/usr/lib/acct/nulladm file

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily [–l] [–c] [mmdd]

/usr/lib/acct/prtacct file ["heading"]

/usr/lib/acct/runacct [mmdd [state]]

/usr/lib/acct/shutacct ["reason"]

/usr/lib/acct/startup

/usr/lib/acct/turnacct on | off | switch

HP–UX COMPATIBILITY

Level: HP–UX/EXTENDED

Origin: System V

DESCRIPTION

Chargefee can be invoked to charge a *number* of units to *login-name*. A record is written to
/usr/adm/fee, to be merged with other accounting records during the night.

Ckpacct should be initiated via *cron*(1M). It periodically checks the size of **/usr/adm/pacct**. If
the size exceeds *blocks*, 1000 by default, *turnacct* will be invoked with argument *switch*. If the
number of free disk blocks in the **/usr** file system falls below 500, *ckpacct* will automatically turn
off the collection of process accounting records via the **off** argument to *turnacct*. When at least
this number of blocks is restored, the accounting will be activated again. This feature is sensitive
to the frequency at which *ckpacct* is executed, usually by *cron*.

Dodisk should be invoked by *cron* to perform the disk accounting functions. By default, it will do
disk accounting on the special files in **/etc/checklist**. If the **-o** flag is used, it will do a slower
version of disk accounting by login directory. *Files* specify the one or more filesystem names
where disk accounting will be done. If *files* are used, disk accounting will be done on these filesys-
tems only. If the **-o** flag is used, *files* should be mount points of mounted filesystem. If omitted,
they should be the special file names of mountable filesystems.

Lastlogin is invoked by *runacct* to update **/usr/adm/acct/sum/loginlog**, which shows the last
date on which each person logged in.

Monacct should be invoked once each month or each accounting period. *Number* indicates which
month or period it is. If *number* is not given, it defaults to the current month (01-12). This
default is useful if *monacct* is to executed via *cron*(1M) on the first day of each month. *Monacct*
creates   summary   files   in   **/usr/adm/acct/fiscal**   and   restarts   summary   files   in
/usr/adm/acct/sum.

Nulladm creates *file* with mode 664 and insures that owner and group are **adm**. It is called by
various accounting shell procedures.

Prctmp can be used to print the session record file (normally **/usr/adm/acct/nite/ctmp** created by *acctcon1* (see *acctcon*(1M)).

Prdaily is invoked by *runacct* to format a report of the previous day's accounting data. The report resides in **/usr/adm/acct/sum/rprt**mmdd where *mmdd* is the month and day of the report. The current daily accounting reports may be printed by typing *prdaily*. Previous days' accounting reports can be printed by using the *mmdd* option and specifying the exact report date desired. The -l flag prints a report of exceptional usage by login id for the specifed date. Previous daily reports are cleaned up and therefore inaccessible after each invocation of *monacct*. The **-c** flag prints a report of exceptional resource usage by command and may be used on current day's accounting data only.

Prtacct can be used to format and print any total accounting (**tacct**) file.

Runacct performs the accumulation of connect, process, fee, and disk accounting on a daily basis. It also creates summaries of command usage. For more information, see *runacct*(1M).

Shutacct should be invoked during a system shutdown (usually in **/etc/shutdown**) to turn process accounting off and append a "reason" record to **/etc/wtmp**.

Startup should be called by **/etc/rc** to turn the accounting on whenever the system is brought up.

Turnacct is an interface to *accton* (see *acct*(1M)) to turn process accounting **on** or **off**. The **switch** argument turns accounting off, moves the current **/usr/adm/pacct** to the next free name in **/usr/adm/pacct**incr (where *incr* is a number starting with **1** and incrementing by one for each additional **pacct** file), then turns accounting back on again. This procedure is called by *ckpacct* and thus can be taken care of by the *cron* and used to keep **pacct** to a reasonable size.

FILES

/usr/adm/fee	accumulator for fees
/usr/adm/pacct	current file for per–process accounting
/usr/adm/pacct*	used if pacct gets large and during execution of daily accounting procedure
/etc/wtmp	login/logoff summary
/usr/lib/acct/ptelus.awk	contains the limits for exceptional usage by login id
/usr/lib/acct/ptecms.awk	contains the limits for exceptional usage by command name
/usr/adm/acct/nite	working directory
/usr/lib/acct	holds all accounting commands listed in sub–class 1M of this manual
/usr/adm/acct/sum	summary directory, should be saved

HARDWARE DEPENDENCIES

Series 500:

The system's process accounting routine silently enforces a 5000–block limit on the size of the process accounting file. Therefore, setting the maximum size of */usr/adm/pacct* larger than 5000 blocks will prevent *ckpacct* from automatically invoking *turnacct switch*, since the file size will never reach the specified limit. See *acct*(2).

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), cron(1M), diskusg(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).

NAME
       backup - backup or archive file system

SYNOPSIS
       /etc/backup [-archive] [-fsck]

HP–UX COMPATIBILITY
       Level: HP–UX/NON–STANDARD

       Origin: HP

       Remarks: This manual page describes *backup* as it is implemented on the Series 200 computer.
                   Refer to other *backup*(1M) manual pages for information valid for other implementa-
                   tions.

DESCRIPTION
       Backup uses *find*(1) and *cpio*(1) to save a *cpio* archive of all files which have been modified since
       the modification time of **/etc/archivedate** on the default tape drive (**/dev/rct**). *Backup* should
       be periodically invoked to ensure adequate file backup.

       The **-archive** option causes *backup* to save all files, regardless of their modification date, and then
       update **/etc/archivedate** using *touch*(1).

       Backup prompts you to mount a new tape and continue if there is no more room on the current
       tape. Note that this prompting does not occur if you are running *backup* from *cron*(1M).

       The **-fsck** option causes *backup* to start a file system consistency check (without correction) after
       the backup is complete. For correct results, it is important that the system be effectively single-
       user while *fsck* is running, especially if **-fsck** is allowed to automatically fix whatever inconsisten-
       cies it finds. *Backup* does not ensure that the system is single–user.

       You may edit **/etc/backup** to "customize" it for your system. For example, *backup* uses *tcio*(1)
       with *cpio* to backup your files on an HP Command Set 80 disc's streaming tape. You will need to
       modify *backup* to use *cpio*(1) if you want to access a standard HP Tape Drive.

       Several local values are used which can be customized:

       backupdirs   specifies which directories to recursively back up (usually /, meaning all direc-
                        tories);

       backuplog   file name where start and finish times, block counts, and error messages are
                        logged;

       archive   file name whose date is the date of the last archive;

       remind   file name that is checked by **/etc/profile** to remind the next person who logs
                        in to change the backup tape;

       outdev   specifies the output device for the backed up files;

       fscklog   file name where start and finish times and *fsck* output is logged.

       You may want to make other changes, such as whether or not *fsck* does automatic correction
       (according to its arguments), where *cpio* output is directed, other information logging, etc.

       In all cases, the output from *backup* is a normal *cpio* archive file (or volume) which can be read
       using *tcio* and *cpio* with the **c** option.

FILES
       /etc/archivedate
       parameterized file names

SEE ALSO
       cpio(1), find(1), touch(1), cron(1M), fsck(1M).

BUGS

Refer to **BUGS** in *cpio*(1).

When *cpio* runs out of tape, it sends an error to *stderr* and demands a new special file name from /dev/tty.

To continue, rewind the tape, mount the new tape, type the name of the new special file at the system console, and press **RETURN**.

If *backup* is left running overnight and the tape runs out, *backup* terminates, leaving the *find* process still waiting. You need to kill this process when you return.

NAME
backup - backup or archive file system

SYNOPSIS
/etc/backup [**-archive**] [**-fsck**]

HP-UX COMPATIBILITY
Level: HP-UX/STANDARD

Origin: HP

Remarks: This manual page describes *backup* as it is implemented on the Series 500 computers. Refer to other *backup*(1M) manual pages for information valid for other implementations.

    Not supported in the Integral Personal Computer.

DESCRIPTION
Backup uses *find*(1) and *cpio*(1) to save on the default tape drive (**/dev/rmt79***xx*) a *cpio* archive of all files which have been modified since the modification time of **/etc/archivedate**. *Backup* should be periodically invoked by *cron*(1M) at night, or when the system is otherwise idle.

The **-archive** option causes *backup* to save all files, regardless of their modification date, and then update **/etc/archivedate** using *touch*(1).

Backup prompts you to mount a new tape and continue if there is no more room on the current tape. Note that this prompting does not occur if you are running *backup* from *cron*(1M).

The **-fsck** option causes *backup* to start a file system consistency check (without correction) after the backup is complete. This is the normal mode of nightly operation. For correct results, it is important that the system be effectively single–user while *fsck* is running, especially if it is allowed to automatically fix whatever inconsistencies it finds. *Backup* does not ensure that the system is single–user.

You should edit **/etc/backup** to "customize" it for your system. For example, *backup* uses *tcio*(1) by default. You will need to modify *backup* to use *cpio*(1) if you want to access a raw device.

Several parameters are used which can be customized:

backupdirs	specifies which directories to recursively back up (usually /, meaning all directories);
backuplog	file name where start and finish times, block counts, and error messages are logged;
archive	file name whose date is the date of the last archive;
remind	file name that is checked by **/etc/profile** to remind the next person who logs in to change the backup tape;
rootdev	list of places for *fsck* (usually a character special file that points to the root device);
fscklog	file name where start and finish times and *fsck* output is logged.

You may want to make other changes, such as whether or not *fsck* does automatic correction (according to its arguments), where *cpio* output is directed, other information logging, etc.

In all cases, the output from *backup* is a normal *cpio* archive file (or volume) which can be read using *tcio* (if used to generate the backup) and *cpio* with the **-c** option.

FILES
/etc/archivedate
parameterized file names

SEE ALSO

    cpio(1), find(1), touch(1), cron(1M), fsck(1M).

BUGS

    Refer to **BUGS** in *cpio*(1).

    When *cpio* runs out of tape, it sends an error to *stderr* (which is logged, so it does not appear on your CRT), and demands a new special file name from /dev/tty. To continue, rewind the tape, mount the new tape, type the name of the new special file at the system console, and press **RETURN**.

    If *backup* is left running overnight and the tape runs out, *backup* terminates, leaving the *find* process still waiting. You need to kill this process when you return.

NAME
brc, bcheckrc, rc, powerfail - system initialization shell scripts

SYNOPSIS
/etc/brc

/etc/bcheckrc

/etc/rc

/etc/powerfail

HP-UX COMPATIBILITY
Level: HP–UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral PC.

DESCRIPTION
Except for *powerfail*, these shell procedures are executed via entries in */etc/inittab* by *init*(1M) when the system is changed out of *SINGLE USER* mode. *Powerfail* is executed whenever a system power failure is detected.

The *brc* procedure clears the mounted file system table, **/etc/mnttab** (see *mnttab*(4)), and loads any programmable micro–processors with their appropriate scripts.

The *bcheckrc* procedure performs all the necessary consistency checks to prepare the system to change into multi–user mode. It will prompt to set the system date and to check the file systems with *fsck*(1M).

The *rc* procedure starts all system daemons before the terminal lines are enabled for multi–user mode. In addition, file systems are mounted and accounting, error logging, system activity logging and the Remote Job Entry (RJE) system are activated in this procedure.

The *powerfail* procedure is invoked when the system detects a power failure condition. Its chief duty is to reload any programmable micro–processors with their appropriate scripts, if suitable. It also logs the fact that a power failure occurred.

SEE ALSO
fsck(1M), init(1M), shutdown(1M), inittab(5), mnttab(5).

NAME
catman - create the cat files for the manual

SYNOPSIS
/etc/catman [**-p**] [**-n**] [**-w**] [sections]

HP-UX COMPATIBILITY
Level: HP UX/STANDARD

Origin: UCB

Remarks: Not supported on the Integral PC.

DESCRIPTION
Catman creates the preformatted versions of the on–line manual from the *nroff* input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, *catman* will recreate the **/usr/lib/whatis** database.

If there is one parameter not starting with a '-', it is taken to be a list of manual sections to look in. For example

catman 123

will cause the updating to only happen to manual sections 1, 2, and 3.

Options:

-n   prevents creation of **/usr/lib/whatis**.

-p   prints what would be done instead of doing it.

-w   causes only the **/usr/lib/whatis** database to be created. No manual reformatting is done.

FILES
/usr/man/man?/*.*	raw (*nroff* input) manual sections
/usr/man/cat?/*.*	preformatted manual pages
/usr/lib/mkwhatis	commands to make whatis database

SEE ALSO
man(1).

NAME

chroot - change root directory for a command

SYNOPSIS

/etc/chroot newroot command

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral PC.

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.

Command includes both the command name and any arguments.

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

Command cannot be in a shell script.

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

Chroot does not search **PATH** for the location of *command*, so the absolute path name of *command* must be given.

NAME
chsys - change to different operating system or version

SYNOPSIS
/etc/chsys sysname

HP-UX COMPATIBILITY

Level: HP–UX/NON–STANDARD

Origin: HP

Remarks: *Chsys* is implemented on the Series 500 only.

DESCRIPTION
Chsys is a shell script that enables you to boot a different operating system, or a different version of the same operating system, using only one boot area on one disc. *Sysname* is one of a number of operating system names defined within *chsys*. *Chsys* uses *oscp*(1M) to rebuild the boot area on */dev/rhd* with the selected system, reading from ordinary files containing operating system code. *Chsys* then invokes *osck*(1M) to confirm that the new system is "healthy". (Note that *osck* performs a redundant check, so its invocation in *chsys* may be removed if you want to save time.)

Chsys invokes *oscp* as quietly as possible. *Chsys* causes *oscp* to read the new system ID string from a file selected by the *sysname* given, and redirects the output from *oscp* to */dev/null.* If *oscp* and *osck* are successful, *chsys* calls *reboot*(1M) to switch to the new operating system. Note that *oscp* and *osck* together can take longer than a minute to run. During this time, *chsys* keeps you informed as to what actions are being taken.

If you simply want to re-boot the operating system already in the boot area, do **not** use *chsys*. Instead, invoke *reboot*(1M) directly.

If you want to allocate and use several boot areas on several discs, see *osmgr*(1M).

You should modify *chsys* to localize it for your system. You may want to add or delete available *sysnames*, change the names or meanings of *sysnames*, change the name of the character special file (*/dev/rhd*) which points to the boot volume, etc. *Chsys* recognizes four default *sysnames*. They stand for:

> HP–UX Model 520 single–user minimal system;
> HP–UX Model 520 single–user complete system;
> BASIC minimal system;
> BASIC complete system.

These *sysnames* serve as examples for any others you may want to add. They may or may not be useful to you.

Chsys should only be invoked by the effective super user unless both of the following are true:

> the special file which points to the boot device must be readable and writable by whoever invokes *chsys*;

> the *reboot* command must be owned by root and have the set–user–ID bit set.

If either of the above are not true, either the *oscp* or the *reboot* command will fail.

Chsys must be invoked with a **$PATH** that includes the directories containing the *oscp, osck, reboot,* and *echo* commands.

RETURN VALUES
If any of the invoked commands fails, *chsys* writes a message to standard error and exits with the same return value as that returned by the unsuccessful command. *Chsys* returns 1 if invoked improperly.

SEE ALSO
sh(1), osmgr(1M), shutdown(1M), stopsys(1M), sync(1M).

WARNINGS

    Chsys does not check that the system is idle, and it does not notify all users that the system is going down. You should usually execute *shutdown*(1M) before executing *chsys*.

    Chsys does not ask you to confirm that the intended operating system or version has been selected before the system is re-booted. However, *osck* ensures that the system is rebootable, and *reboot* performs a *sync*(1M). Note that new operating systems built in the boot area by *oscp* are always marked as loadable (see *osmark*(1M)).

NAME

       clri - clear i-node

SYNOPSIS

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

HP-UX COMPATIBILITY

       Level: HP-UX/NUCLEUS

       Origin: HP

       Remarks: *Clri* is implemented only on those systems using the Bell file system.

                    Clri is currently implemented on Series 200 and the Integral PC only.

DESCRIPTION

       Clri writes zeros on the 128 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 com-
mand 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 direc-
tory. If it is used to zero out 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

    clrsvc - clear x25 switched virtual circuit

SYNOPSIS

    /usr/lib/uucp/X25/clrsvc line pad–type

HP–UX COMPATIBILITY

    Level: HP–UX/NON–STANDARD

    Origin: HP

DESCRIPTION

    Clrsvc clears any virtual circuit that might be established on the **line** specified. The **pad–type**
    indicates to *clrsvc* what *opx25* script to run from */usr/lib/uucp/X25*. HP2334A is the only PAD
    supported at this time, and results in an *opx25* execution of *HP2334A.clr*.

    A typical invokation would be:

        /usr/lib/uucp/X25/clrsvc /dev/x25.1 HP2334A

SEE ALSO

    Getx25(1C), opx25(1C), getty(1M), login(1), uucp(1C)

AUTHOR

    Radek Linhart

NAME

    config – configure an HP–UX system

SYNOPSIS

    /etc/config [–t] [–m master] [–c file] [– l file] [–a file] dfile

HP-UX COMPATIBILITY

    Level: HP–UX/NON–STANDARD

    Origin: HP

    Remarks: *Config* is implemented on the Series 200 only.

DESCRIPTION

    Config enables the user to configure the following parts of the operating system:

        1. device switch drivers and I/O cards

        2. root and swap devices

        3. selected system parameters

        4. kernel code that handles messages, semaphores, and shared memory

    It takes as input, a user–provided description of an HP-UX system (*dfile*) and always generates two files, with an optional third file. The first file is a C program that defines the configuration tables for the various devices on the system. The second file is a *makefile* script that will compile the C program produced and relink the newly configured system. The third file (if specified) contains a *mknod* command for each device specified in *dfile*.

    The options available:

    –t   gives a short table of major device numbers for the character and block devices named in *dfile*. This can facilitate the creation of special files.

    –m *master* specifies that the file *master* contains all the information regarding supported devices. The default file name is */etc/master*. This file is supplied with the HP–UX system and **should not be modified unless the user fully understands its construction**.

    –c *file*   specifies the name of the configuration table file produced by running the user-- data file, *dfile*, through *config*(1M). The default file name is *conf.c*

    –l *file*   specifies the name of the *makefile* script that will compile the configuration pro- gram and relink the newly configured system. The default file name is *config.mk*

    –a *file*   serves two functions:

        1. When specified without *dfile*, a *mkdev* script of templates is produced.

        2. If *dfile* is given, this indicates that the user will supply addresses for devices so that *config* can produce a script that contains both the *mkdev* templates and a list of *mknod* commands for each device specified in *dfile*. If this option is chosen, all devices must have addresses. Zero (0) as a dummy address, is valid and necessary for many of the devices, e.g., the card drivers. The default file name is *mkdev*.

The only required argument is either *dfile* or −**a**. If *dfile* is given it must contain device information for the user's system. This file is divided into two parts. The first part contains physical device and driver specifications; the second part contains system-dependent information. Any line with an asterisk (*) in column 1 is a comment.

The following devices are **not** configurable and should not be specified in the system descrition file, *dfile*:

> ; l l.   swap   cons tty   sy   mm   ite200   iomap   graphics r8042   hil nim-
> itz

Part 1 of dfile:

This part of *dfile* allows you to configure:

1. device switch drivers

2. I/O cards

3. pseudo drivers, e.g., *ieee802, pty*

Each line contains 1, 2, or 3 fields, delimited by blanks and/or tabs in the following format:

> *devname* [*address*] [*specialfilename*]

where:

> *devname*   is the driver name for the device (e.g., cs80 for the HP7912 64MB disc drive) or card (98629 for the SRM card) or the name of the pseudo-driver (e.g., iee802 for the ieee802 protocol) you wish to configure.
>
> *address*   is the minor number for that device as given to mknod or the select code of the card if addressing checking is desired. For pseudo-drivers, i.e., iee802, pty, ethernet, the address field is 0. (in hexadecimal, without the preceding 0x).
>
> *specialfilename*
>   is what you want the device's special file to be called in the afile.

For example, to specify a 7914 disc at select code 14, bus address 0 with mknod name */dev/hd*:

> cs80 0E0000 hd

The complete list of configurable devices, cards, and pseudo-drivers is given in the **EXAMPLE** section.

It is not necessary to specify the *address* field, but if you do specify this field and use the −**a** option, *config* will produce a file containing a *mknod* command for each device you specify. It will also check for the unique use of addresses. The −**a** option allows you to name this file.

Part 2 of dfile:

The second part contains four different types of lines; none of these specifications are required.

1. Root device specification lines which have the following form:

> root *devname address*

where *devname* is the product number (without the suffix) of the device you wish to configure, e.g., cs80 for the HP7912 64MB disc drive, and *address* is the minor device number (in hexadecimal, without the preceding 0x).

2. Swap device specification lines:/fR

If you want the system to auto configure the swap device but you want to specify the swap size, then use:

swapsize <#*blocks*>

If you want to specify both the swap device location and its size then the specification line has the following form:

swap *devname address swplo* [*nswap*]

where:

devname   is the product number (with the suffix) of the device you wish to configure, e.g., cs80 for the HP7912 64MB disc drive (in hexadecimal).

address   is the minor device number (in hexadecimal)

swplo   is the location (decimal) of the swap area

nswap   is the number of disc blocks (decimal) in the swap area. Only the *nswap* parameter is optional. Zero is the default for auto–configuration.

swplo:

A negative value (typically –1) for *swplo* specifies that a file system is expected on the device. At boot–up, the super block will be read to determine the exact size of the file system, and this value will be put in *swplo*. If the swap device is auto–configured, this is the mechanism used. If the super block doesn't appear valid, the entry will be skipped, so that the case of a corrupted super block won't later cause the entire file system to be corrupted by configuring the swap area on top of it.

A positive (including zero) value for *swplo* specifies that at least that much area must be reserved. Zero obviously means to reserve no area at the head of the device. The case for *swplo* pointing beyond the end of the device is gracefully handled.

nswap:

If *nswap* is zero, the entire remainder of the device is automatically configured in as swap area.

If *nswap* is non–zero, it's absolute value is treated as an upper bound for the size of the swap area. Then, for the case that the swap area size has actually been cut back, the sign of *nswap* determines whether *swplo* remains as is, resulting in the swap area being adjacent to the reserved area, or whether *swplo* is bumped by the size of the unused area, resulting in the swap area being adjacent to the tail of the device.

3. Parameter specification

These parameters should not be modified unless the user fully understands the ramifications of doing so. See the *System Administrator's Manual* for more detail on each parameter.

The format: lines of two fields each (number is decimal). Each line is independent and optional.

System Parameters:

center; 1 1. maxusers   number or formula timezone   number or formula
dst   number or formula procs number or formula inodes number or formula
files   number or formula nbuf   number or formula ncallout number or formula
texts   number or formula unlockable_mem   number or formula nflocks number or for-
mula npty   number or formula maxuprc   number or formula dmmin number or formula
dmmax number   or   formula dmtext number or   formula dmshm number   or   formula
maxdsiz number   or   formula maxssiz number or   formula maxtsiz number   or   formula
shmmaxaddr   number or formula

System V code: messages (*mesg*), semaphores (*sema*) and shared memory (*shmem*) capability

If *mesg*, *sema*, *shmem*= 1, the kernel code for these features will be included (default); if they = 0, the kernel code will not be included: they are independent. If they are included any of the parameters listed below may be modified. center; 1 1. mesg   1 msgmap   number or formula msgmax number or formula msgmnb   number or formula msgmni number or formula msgssz number or formula msgtql number or formula msgseg number or formula center; 1 1. sema   1 semmap number or formula semmni number or formula semmns number or formula semmnu   number or formula semmsl number or formula semvmx number or formula semaem number   or   formula center; 1 1. shmem 1 shmmax   number or formula shmmin number   or   formula shmmni number or   formula shmseg number   or   formula shmbrk number or formula

EXAMPLE

The *dfile* below will configure an HP–UX system with all the drivers that are currently supported on the Series 200 Release 5.0. The tunable parameters given are the system defaults.

```
* drivers
cs80
flex
amigo
tape
printer
stape
srm
ptymas
ptyslv
ieee802
ethernet
hpib
gpio
ciper
* cards
98624
98625
98626
98628
98642
* reconfigure the swap area to occupy an entire CS/80 drive at
* select code 14 bus address 01
```

```
            swap                        cs80    0E0100   0       0
            * tunable parameters
            maxusers                    8
            timezone                    420
            dst                         1
            procs                       (20+8*MAXUSERS)
            inodes                      ((NPROC+16+MAXUSERS)+32)
            files                       (16*(NPROC+16+MAXUSERS)/10+32+2*NETSLOP)
            nbuf                        0       /* configure based on memory */
            ncallout                    (16+NPROC)
            texts                       (24+MAXUSERS+NETSLOP)
            unlockable_mem              50
            nflocks                     200
            npty                        96
            maxuprc                     25
            dmmin                       16
            dmmax                       2048
            dmtext                      1365
            dmshm                       512
            maxdsiz                     0x01000000
            maxssiz                     0x01000000
            maxtsiz                     0x01000000
            shmmaxaddr                  0x00ffffff
            * configure in messages, semas, and shared memory
            mesg                        1
            msgmap                      (msgtql + 3)
            msgmax                      8192
            msgmnb                      16384
            msgmni                      50
            msgssz                      1
            msgtql                      40
            msgseg                      16384
            sema                        1
            semmap                      10
            semmni                      64
            semmns                      128
            semmnu                      30
            semmsl                      25
            semvmx                      32767
            semaem                      16384
            shmem                       1
            shmmax                      (2048*1024)
            shmmin                      1
            shmmni                      100
            shmseg                      10
            shmbrk                      16
```

FILES

/etc/master	default input master device table
conf.c	default output configuration table
config.mk	default makefile script
mkdev	default mknod script

SEE ALSO
    master(5)

NAME

      cpset - install object files in binary directories

SYNOPSIS

      cpset [–**o**] object directory [mode [owner [group]]]

HP–UX COMPATIBILITY

      Level: HP–UX/STANDARD

      Origin: System V

DESCRIPTION

      Cpset is used to install the specified *object* file in the given *directory*. The *mode*, *owner*, and *group*, of the destination file may be specified on the command line. If this data is omitted, two results are possible:

            If the user of *cpset* has administrative permissions (that is, the user's numerical ID is less than 100), the following defaults are provided:

            mode - 0755

            owner - bin

            group - bin

            If the user is not an administrator, the default, owner, and group of the destination file will be that of the invoker.

      An optional argument of **-o** will force *cpset* to move *object* to **OLD***object* in the destination directory before installing the new object.

      For example:

            cpset echo /bin 0755 bin bin

            cpset echo /bin

            cpset echo /bin/echo

      All the examples above have the same effect (assuming the user is an administrator). The file **echo** will be copied into **/bin** and will be given **0755, bin, bin** as the mode, owner, and group, respectively.

      Cpset utilizes the file **/usr/src/destinations** to determine the final destination of a file. The locations file contains pairs of pathnames separated by spaces or tabs. The first name is the "official" destination (for example: **/bin/echo**). The second name is the new destination. For example, if *echo* is moved from **/bin** to **/usr/bin**, the entry in **/usr/src/destinations** would be:

            /bin/echo   /usr/bin/echo

      When the actual installation happens, *cpset* verifies that the "old" pathname does not exist. If a file exists at that location, *cpset* issues a warning and continues. This file does not exist on a dis-- tribution tape; it is used by sites to track local command movement. The procedures used to build the source will be responsible for defining the "official" locations of the source.

    Cross Generation

      The environment variable **ROOT** will be used to locate the destination file (in the form **$ROOT/usr/src/destinations**). This is necessary in the cases where cross generation is being done on a production system.

SEE ALSO

      install(1M), make(1).

NAME
>   cron – clock daemon

SYNOPSIS
>   **/etc/cron**

HP–UX COMPATIBILITY
>   Level: HP–UX/STANDARD
>
>   Origin: System V
>
>   Native Language Support:
>   >   8-bit filenames.
>
>   Remarks: Not supported on the Integral PC.

DESCRIPTION
>   *Cron* executes commands at specified dates and times. Regularly scheduled commands can be
>   specified according to instructions found in crontab files; users can submit their own crontab file
>   via the *crontab* command. Commands which are to be executed only once may be submitted via
>   the *at* command. Since *cron* never exits, it should only be executed once. This is best done by
>   running *cron* from the initialization process through the file **/etc/rc** (see *init*(1M)).
>
>   *Cron* only examines crontab files and at command files during process initialization and when a
>   file changes. This reduces the overhead of checking for new or changed files at regularly scheduled
>   intervals.

FILES
>   /usr/lib/cron   main cron directory
>   /usr/lib/cron/log   accounting information
>   /usr/spool/cron   spool area

SEE ALSO
>   at(1), crontab(1), sh(1), init(1M).

DIAGNOSTICS
>   A history of all actions taken by cron are recorded in **/usr/lib/cron/log.**

NAME

    devnm - device name

SYNOPSIS

    /etc/devnm [names]

HP–UX COMPATIBILITY

    Level: HP–UX/STANDARD

    Origin: System V

    Remarks: *Devnm*(1m) is currently implemented on the Series 200 and the Integral PC only.

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 are 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 *brc*(1M)) to construct a mount table entry for the **root** device.

EXAMPLE

    The command:

        /etc/devnm /usr

    produces

        dsk/0s1 /usr

    if **/usr** is mounted on **/dev/dsk/0s1**.

FILES

    /dev/dsk/*

    /etc/mnttab

SEE ALSO

    brc(1M), setmnt(1M).

NAME
df - report number of free disk blocks

SYNOPSIS
df [**-t**] [**-f**] [file–systems]

HP–UX COMPATIBILITY
Level: HP–UX/EXTENDED

Origin: HP

DESCRIPTION
Df prints out the number of free 512–byte blocks and free i–nodes available for on–line file sys-
tems by examining the counts kept in the super-block(s); *file–systems* may be specified either by
device name (e.g., **/dev/dsk/0s1**) 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.

The file space reported is the space available to the ordinary user. For more details see MINFREE
in *fs*(5).

FILES
/dev/dsk/*
/etc/mnttab

SEE ALSO
du(1), fsck(1M), fs(5), mnttab(5).

NAME
diskusg – generate disk accounting data by user ID

SYNOPSIS
diskusg [options] [files]

HP–UX COMPATIBILITY
Level: HP–UX/EXTENDED

Origin: System V

DESCRIPTION
Diskusg generates intermediate disk accounting information from data in *files,* or the standard input if omitted. *Diskusg* output lines on the standard output, one per user, in the following format:

          uid login #blocks

where

uid –   the numerical user ID of the user.

login –   the login name of the user; and

#blocks –   the total number of disk blocks allocated to this user.

Diskusg normally reads only the i–nodes of file systems for disk accounting. In this case, *files* are the special filenames of these devices.

Diskusg recognizes the following options:

−**s**   the input data is already in *diskusg* output format. *Diskusg* combines all lines for a single user into a single line.

−**v**   verbose. Print a list on standard error of all files that are charged to no one.

−**i** *fnmlist*   ignore the data on those file systems whose file system name is in *fnmlist. Fnmlist* is a list of file system names separated by commas or enclose within quotes. *Diskusg* compares each name in this list with the file system name stored in the volume ID (see *labelit(1M)).*

−**p** *file*   use *file* as the name of the password file to generate login names. **/etc/passwd** is used by default.

−**u** *file*   write records to *file* of files that are charged to no one. Records consist of the special file name, the i–node number, and the user ID.

The output of *diskusg* is normally the input to *acctdisk* (see *acct*(1M)) which generates total accounting records that can be merged with other accounting records. *Diskusg* is normally run in *dodisk* (see *acctsh*(1M)).

EXAMPLES
The following will generate daily disk accounting information:

```
for i in /dev/rhd* ; do
        diskusg $i > dtmp.`basename $i` &
done
wait
diskusg –s dtmp.* | sort +0n +1 | acctdisk > disktacct
```

FILES
/etc/passwd   used for user ID to login name conversions

SEE ALSO
acct(1M), acctsh(1M), acct(5)

NAME
fsck - file system consistency check and interactive repair

SYNOPSIS
/etc/fsck -p [file system ...]
/etc/fsck [-b block#][-y][-n][-q][file system   ...]

HP–UX COMPATIBILITY
Level: Large Machine/SVID/HFS

Origin: HP

Remarks: This version of *fsck* applies to the HFS file system. See other *fsck* manual pages for
other file systems.

DESCRIPTION
fsck audits and interactively repairs inconsistent conditions for HP–UX file systems. If the file
system is consistent then the number of files on that file system and the number of used and free
blocks are reported. If the file system is inconsistent then *fsck* provides a mechanism to fix these
inconsistencies depending on which form of fsck command is used.

fsck checks a default set of file systems or the file systems specified in the command line. If *file
system* is not specified, *fsck* reads the table in **/etc/checklist** to determine which file systems to
check.

If the **–p** option is used without specifying a **file–system,** *fsck* reads the specified pass numbers
in **/etc/checklist** to inspect groups of disks in parallel taking maximum advantage of I/O over-
lap to preen the file systems as quickly as possible. The **–p** option is normally used in the script
/etc/rc during automatic reboot. Normally, the root file system will be checked on pass 1, other
"root" ("0" section) file systems on pass 2, other small file systems on separate passes (e.g. the
"section 4" file systems on pass 3 and the "section 7" file systems on pass 4), and finally the large
user file systems on the last pass, e.g. pass 5. A pass number of 0 or a type which is neither "rw"
nor "ro" in checklist causes a file system not to be checked. If the optional fields are not present
on a line in */etc/checklist*, or the pass number is –1, *fsck* will preen the file system on such lines
sequentially after all eligible file systems with positive pass numbers have been preened.

Below are the inconsistencies which *fsck* with the **-p** option will correct; if it encounters other
inconsistencies it exits with an abnormal return status. For each corrected inconsistency, one or
more lines will be printed identifying the file system on which the correction will take place, and
the nature of the correction. These inconsistencies are limited to the following:
            Unreferenced inodes
            Unreferenced pipes and fifos
            Link counts in inodes too large
            Missing blocks in the free list
            Blocks in the free list also in files
            Counts in the super-block wrong

fsck without –**p** option, prompts for concurrence before each correction is attempted when the file
system is inconsistent. It should be noted that some corrective actions will result in a 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. The following options in
the second form are interpreted by *fsck.*

-b   Use the block specified immediately after the flag as the super block for the file system. An
        alternate super block will always be found at block ((SBSIZE + BBSIZE)/DEV_BSIZE),
        typically block 16.

-y   Assume a yes response to all questions asked by *fsck;* this should be used with great cau-
        tion as this is a free license to continue after essentially unlimited trouble has been

encountered.

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

- **-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 and cylinder groups will be automatically fixed.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.

2. Blocks claimed by an inode or the free list outside the range of the file system.

3. Incorrect link counts.

4. Size checks:
    Directory size not of proper format.

5. Bad inode format.

6. Blocks not accounted for anywhere.

7. Directory checks:
    File pointing to unallocated inode.
    Inode number out of range.

8. Super Block checks:
    More blocks for inodes than there are in the file system.

9. Bad free block list format.

10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. The only restriction is that the directory **lost+found** must preexist in the root of the filesystem being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

After *fsck* has checked and fixed the file system, it will store the correct magic number in the super block if it is not already there. For a non–root file system, FS_CLEAN will be stored there. For the root file system, which is mounted at the time of the fsck, if there were no problems found and if FS_OK was already set, then no changes are required to the super block.

Checking the raw device is almost always faster.

HARDWARE DEPENDENCIES
    Series 200:

       Series 200 5.0 release supports only one section per volume.

FILES
    /etc/checklist   contains default list of file systems to check.

SEE ALSO
    checklist(5), fs(5), fsclean(1M), newfs(1M), mkfs(1M),

NAME

fsck - file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [-y] [-n] [-s] [-d] [*file system*]

HP–UX COMPATIBILITY

Level: HP–UX/NUCLEUS

Origin: HP

Remarks: This manual page describes *fsck* as implemented on the Series 500 (SDF file system). Refer to other *fsck*(1M) manual pages for information valid for other file systems.

Not supported on the Integral PC.

DESCRIPTION

Fsck checks and interactively repairs inconsistent conditions for SDF file systems. If the file system is consistent, then the number of files, the number of blocks used, the number of blocks free, and the percent volume unused are reported. If the file system is inconsistent, the operator is prompted for concurrence before each operation is attempted. Note that many corrective actions will result in some loss of data. The amount and severity of the loss may be determined from the diagnostic output. The default action for each inconsistency 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 makes multiple passes over the file system, so care should be taken to ensure that the system is quiescent. You should unmount the file system being checked, if possible. At the least, the system should be single–user, and spurious processes (such as *cron*) should be killed.

The following flags are interpreted by *fsck*:

-y   Assume a **yes** response to all questions asked.

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

-s   Ignore the actual free list and unconditionally reconstruct a new one. This option is useful in correcting multiply claimed blocks when one of the claimants is the free list. When using this option, the number of unclaimed blocks reported by *fsck* includes all the blocks in the free map. This can produce extensive output if **-d** is also selected.

   -s should only be selected after a previous *fsck* indicates a conflict between a file and the free map. After *fsck -s* has executed, the integrity of the conflicting file(s) should be checked.

   If **-s** is used to correct a problem on a virtual memory device, there is a high probability that the final step in *fsck* will fail, and you will be forced to reboot. Should this occur, and appropriate error message will be printed. No damage should occur.

-d   Dump additional information. The more **d**'s that are present, the more information that is dumped. You may specify up to five **d**'s. However, using more than two can result in an overwhelming amount of output.

Fsck also recognizes, and ignores, the **-S** and **-t** options found in other versions of *fsck*. An appropriate warning is printed.

File system is a device file name describing the device on which the file system to be checked resides (that is, **/dev/rhd**). If no *file system*(s) are specified, *fsck* will read a list of default file systems from the file **/etc/checklist**.

Error messages from *fsck* are written to *stderr*. Information generated because of the **-d** option and normal output is written to *stdout*. Both are unbuffered.

Inconsistencies checked include:

1. Blocks claimed by more than one i–node, or by the free list;

2. Blocks claimed by an i–node or the free list outside the range of the file system;

3. Incorrect link counts;

4. Blocks not accounted for anywhere;

5. Bad i–node format;

6. Directory checks:
   Files pointing to unallocated i–nodes,
   I–node numbers out of range,
   Multiply linked directories,
   Link to the parent directory.

Orphaned files (allocated but unreferenced) with non–zero sizes are, with the operator's con–currence, reconnected by placing them in the *lost+found* directory. The name assigned is the i–node number. The only restriction is that *lost+found* must exist in the root of the file system being checked, and must have empty slots in which entries can be made. This is accomplished by creating *lost+found*, copying a number of files to it, and then removing them (before *fsck* is exe–cuted).

Orphaned directories and files with zero size, with the operator's concurrence, are returned directly to the free list. This will also happen if the *lost+found* directory does not exist.

You should run a backup prior to running *fsck* for repairs.

FILES

/etc/checklist   contains the default list of file systems to check

SEE ALSO

checklist(5), fs(5).

Series 500 HP–UX System Administrator Manual.

DIAGNOSTICS

The diagnostics are intended to be self–explanatory.

BUGS

All file systems must be described by a character special device file.

Do not redirect *stdout* or *stderr* to a file on the device being checked. This includes pipes when checking the root volume.

Fsck cannot check devices with a logical block size greater than 1024.

NAME

    fsclean - determine shutdown status of specified file system

SYNOPSIS

    /etc/fsclean special

HP-UX COMPATIBILITY

    Level: HP–UX/NUCLEUS

    Origin: HP

DESCRIPTION

    Fslcean determines the shutdown status of the the file system specified by *special*. *Fsclean* reads the super block to determine whether the file system's last shutdown was done correctly. If it was, then *fsclean* returns 0. If it was not, then *fsclean* returns 1. All other errors, such as "cannot open the specified device file," return 2.

    Fsclean can be used in */etc/rc* to determine whether *fsck* should be run on the file system before continuing with the normal boot of the HP–UX system.

SEE ALSO

    rc(1M), reboot(1M).

NAME

    fsdb - file system debugger

SYNOPSIS

    /etc/fsdb special [**-b**   block#] [**-**]

HP-UX COMPATIBILITY

    Level: HP–UX/NON–STANDARD

    Origin: HP

    Remarks: This version of *fsdb*(1M) is implemented only on those machines using the HFS file systems. Refer to other *fsdb*(1M) manual pages for information valid for other implementations.

              Always execute *fsck*(1M) after done with *fsdb*.

DESCRIPTION

    Fsdb can be used to patch up a damaged file system after a crash. It normally uses the first super block for the file system located at the beginning of the disk section as the effective super block. If the *–b* flag is used, the block specified immediately after the flag will be used as the super block for the file system. An alternate super block will always be found at block ((SBSIZE + BBSIZE)/DEV_BSIZE), typically block 16.

    Fsdb deals with the file system in terms of block fragments, which are the unit of addressing in the file system and the minimum unit of space allocation. To avoid possible confusion, *fragment* is used to mean that, and *block* is reserved for the larger true block. *Fsdb* has conversions to translate fragment numbers 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 fragment addresses. These can be disabled if necessary by invoking *fsdb* with the optional - argument or by the use of the **O** symbol.

    Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. Hexa decimal numbers must be prefixed with **0x**. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

    Fsdb reads a fragment at a time. A buffer management routine is used to retain commonly used fragment of data in order to reduce the number of read system calls. All assignment operations result in an immediate write–through of the corresponding fragment.

    The symbols recognized by *fsdb* are:

#	absolute address
i	convert from i–number to i–node address
b	convert from fragment number to disk address (historically "block")
d	directory slot offset
+,-	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=-	decremental assignment
="	character string assignment
X	hexadecimal flip flop
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. Octal numbers are prefixed with a zero. Hexadecimal numbers are prefixed with 0x. 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 inter-rupt character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect fragment 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 fragment are printed. The print options available are:

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

The **f** symbol is used to print data fragments associated with the current i-node. If followed by a number, that fragment of the file is printed. (Fragments are numbered from zero). The desired print option letter follows the fragment number, if present, or the **f** symbol. This print facility works for small as well as large files except for special files such as fifos, and device special files.

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 (hexadecimal if X toggle is used) address followed by the value in octal (hexadecimal if X toggle is used) 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 charac-ter representation of the entry name. I-nodes are printed with labeled fields describing each ele-ment.

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 in byte unit
a#	data block numbers (0 - 14)
at	time last accessed
mt	time last modified
ct	last time inode changed
maj	major device number
min	minor device number

The following mnemonics are used for directory examination:

di	i-number of the associated directory entry
nm	name of the associated directory entry

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current work-ing 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, fragment zero of the file associated with the working i–node.
2i.fd	prints the first fragment–size piece of 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 fragment's worth of bytes of the file are then printed in ASCII.
1b.px	prints the first fragment of the superblock of this file system in hexadecimal.
2i.a0b.d7.di=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 if the first character of the name field is alphabetic.
a2b.p0d	prints the third fragment of the current i–node as directory entries.

SEE ALSO

fsck(1M), dir(5), fs(5).

WARNING

The use of *fsdb* should be limited to experienced *fsdb* users.

NAME

fsdb - examine/modify file system

SYNOPSIS

fsdb file–system

HP–UX COMPATIBILITY

Level: HP–UX/NON–STANDARD

Origin: HP

Remarks: This manual entry describes *fsdb* as implemented on the Series 500 computers. Refer to other *fsdb*(1M) entries for information valid for other implementations. Note that *fsdb* on the Series 500 is an experimental utility which could change in future releases.

DESCRIPTION

Fsdb provides you with the ability to perform the following functions for each specified SDF *file–system*:

1.   Find the inode number of a file, given its full path name. The *file–system* must be the root file system, or must be mountable to use this feature.

2.   Examine and modify the contents of the superblock (volume header).

3.   Examine and modify the contents of any inode or other file attribute file record.

Integer input to *fsdb* may be entered in decimal (default), octal (with a preceding "0"), or hexadecimal (with a preceding "0x").

File–system is a raw or block special file describing the device on which the file system is located.

Fsdb may be executed only by the super–user.

Fsdb execution is largely self–explanatory. Prompts consist of questions requesting the needed information. When execution begins, *fsdb* displays the following menu:

1 - find inode numbers.
2 - examine superblock.
3 - examine inodes.
q - quit.

after which you are requested to enter one of the options shown. Typing **1** causes *fsdb* to accept full pathnames of files, in return for which it prints the corresponding inode number. Typing **q** returns you to the main menu.

Typing **2** displays the contents of each record in the superblock. Each record is numbered. If a right parenthesis ")" follows the number, then the record can be modified. If a right curly bracket "}" follows the number, then the record cannot be modified. You are then asked whether or not you want to modify the superblock. An answer beginning with **n** sends you back to the menu; an initial **y** causes *fsdb* to ask for the record number to be modified. If the record number specified cannot be modified, you are told about it, and prompted for another record number. If you specify a record number which can be changed, you are prompted for the new data. Typing **q** returns you to the main menu.

Typing **3** causes *fsdb* to prompt you for a file attribute record number. Upon receipt of a valid number, the contents of that record are displayed, and you are prompted for the information you want to change. Parentheses and curly brackets have the same meanings as described above. Typing **q** returns you to the main menu.

Typing **q** at the main menu level terminates the command.

A word of caution: *fsdb* is deceptively easy to use, and therefore should be used with extreme care. Be sure you know what you are doing before you enter too deeply into options **2** or **3**. You are given the opportunity to abort (by typing **q**) any operation before you have changed anything,

so consider carefully what you are about to do before you do it. *Fsdb* does not provide an "undo" function - the changes you make are immediate.

SEE ALSO

fsck(1M).

BUGS

If *fsdb* changes a field that is duplicated in an in–memory OS data structure, the change may be undone by the OS. Forcing a reboot while still in *fsdb* sometimes circumnavigates this problem. Changes to inodes 0 and 1 always fall into this category.

NAME
fwtmp, wtmpfix - manipulate connect accounting records

SYNOPSIS
/usr/lib/acct/fwtmp [-ic]
/usr/lib/acct/wtmpfix [files]

HP–UX COMPATIBILITY
Level: HP–UX/EXTENDED

Origin: System V

DESCRIPTION
Fwtmp
Fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in **wtmp** to formatted ASCII records. The ASCII version is useful to enable editing, via *ed*(1), bad records or general purpose maintenance of the file.

The argument **-ic** is used to denote that input is in ASCII form, and output is to be written in binary form. (The arguments **i** and **c** are independent, respectively specifying ASCII input and binary output, thus **-i** is an ASCII to ASCII copy and **-c** is a binary to binary copy).

Wtmpfix
Wtmpfix examines the standard input or named files in **wtmp** format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A - can be used in place of *files* to indicate the standard input. If time/date corrections are not performed, *acctcon1* will fault when it encounters certain date–change records.

Each time the date is set, a pair of date change records are written to **/usr/adm/wtmp**. The first record is the old date denoted by the string **old time** placed in the line field and the flag **OLD_TIME** placed in the type field of the **<utmp.h>** structure. The second record specifies the new date and is denoted by the string **new time** placed in the line field and the flag **NEW_TIME** placed in the type field. *Wtmpfix* uses these records to synchronize all time stamps in the file. *Wtmpfix* nullifies date change records when writing to the standard output by setting the time field of the **<utmp.h>** structure in the old date change record equal to the time field in the new date change record. In this way, *wtmpfix* and *acctcon1* will not factor in a date change record pair more than once.

In addition to correcting time/date stamps, *wtmpfix* will check the validity of the name field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to **INVALID** and write a diagnostic to the standard error. In this way, *wtmpfix* reduces the chance that *acctcon1* will fail when processing connect accounting records.

FILES
/etc/wtmp
/usr/include/utmp.h

SEE ALSO
acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), runacct(1M), ed(1), acct(2), acct(5), utmp(5).

DIAGNOSTICS
Wtmpfix generates these diagnostics:

Cannot make temporary: xxx
        failed to make temp file

Input truncated at offset: xxx
        missing half of date pair

New date expected at offset: xxx
       missing half of date pair

Cannot read from temp: xxx
       some error reading

Bad file at offset: xxx
       ut_type out of range character only checked)

Out of core
       malloc fails. (Saves table of date changes)

No dtab
       software error (not seen yet)

BUGS

       Fwtmp generates no errors, even on garbage input.

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

SYNOPSIS
      /etc/getty [-h] [-t timeout] line [speed]
      /etc/getty -c file

HP–UX COMPATIBILITY
      Level: HP–UX/NUCLEUS

      Origin: System V

      Remarks: Not supported on the Integral PC.

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 the HP–UX system. Initially, if **/etc/issue** exists, *getty* prints its contents to the user's terminal, followed 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* at what speed to initially run, what the login message should look like, what the initial 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.

      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, new–line 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 HP–UX system 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(5), inittab(5), termio(4).

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, ˆU, ˆD, or & as part of your login name or arguments. They will always be interpreted as having their special meaning as described above.

NAME
getx25 - get x25 line

SYNOPSIS
/etc/getx25 line speed pad–type

HP–UX COMPATIBILITY
Level: HP–UX/NON–STANDARD

Origin: HP

DESCRIPTION
Getx25 is very similar to *getty* in function, but is used only for incoming lines that are connected to an X.25 PAD. It performs special functions such as setting up an initial PAD configuration. It also logs the number of the caller in */usr/spool/uucp/X25LOG*. The third parameter is the name of the PAD being used. HP2334A is the only one supported at this time.

A typical invokation would be:

/etc/getx25 x25.1 2 HP2334A

SEE ALSO
getty(1M), login(1), uucp(1C)

NAME

init, telinit - process control initialization

SYNOPSIS

/etc/init [0123456SsQq]

/etc/telinit [0123456sSQqabc]

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral PC.

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 the HP–UX system 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 *init-tab*(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 formatted *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 tele-type (**/dev/systty**). If this occurs, *init* can be forced to relink **/dev/syscon** by typing a delete on the system teletype which is collocated with the processor.

When *init* prompts for the new *run-level*, the operator may enter only 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 trys to send messages to **/dev/syscon**, 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 does not 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 re prompted. 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
install - install commands

SYNOPSIS
/etc/install [-c dira] [-f dirb] [-i] [-n dirc] [-o] [-s] file [dirx ...]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral PC.

DESCRIPTION

Install is a command most commonly used in "makefiles" (see *make*(1)) to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

Install is useful for installing new commands, or new versions of existing commands, in the standard directories (i.e. /bin, /etc, etc.).

If no options or directories (*dirx* ...) are given, *install* will search a set of default directories (/bin, /usr/bin, /etc, /lib, and /usr/lib, in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file* (the new version), and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx* ...) are specified after *file*, those directories will be searched before the directories specified in the default list.

The meanings of the options are:

-c *dira*
    Installs a new command (*file*) in the directory specified by *dira*, only if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the -s option.

-f *dirb*
    Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to 755 and bin, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the -o or -s options.

-i
    Ignores default directory list, searching only through the given directories (*dirx* ...). May be used alone or with any other options other than -c and -f.

-n *dirc*
    If *file* is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file will be set to 755 and bin, respectively. May be used alone or with any other options other than -c and -f.

-o
    If *file* is found, this option saves the "found" file by copying it to OLD*file* in the directory in which it was found. This option is useful when installing a normally text busy file such as /bin/sh or /etc/getty, where the existing file cannot be removed. May be used alone or with any other options other than -c.

-s
    Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

cpset(1M), make(1).

BUGS

Install cannot create alias links for a command (for example, *vi*(1) is an alias link for *ex*(1)).

NAME

kermit - KERMIT protocol file transfer program

SYNOPSIS

/usr/contrib/bin/kermit [options] files

HP–UX COMPATIBILITY

Level: HP–UX/NON-STANDARD

Origin: Public Domain

DESCRIPTION

Kermit is a file transfer program in common use on MS–DOS systems. It can also be used to tranfer files between two HP–UX systems when used in conjunction with *cu*. *Kermit* is invoked as follows:

usage: kermit c [lbe line baud escapechar]   to connect
         kermit s [diflb line baud] file ...   to send files
         kermit r [diflb line baud]   to receive files

where: c=connect, s=send, r=receive, d=debug, i=image mode, f=no filename conversion, l=tty line, b=baud rate, and e=escape char.

For remote *Kermit*, the format is either **kermit r** to receive files, or **kermit s** *file ...* to send files.

A typical *kermit* file transfer in conjunction with *cu* would look something like this:

```
$ cu –lculb0 –qm dir
Connected
% ls
% kermit r
~&kermit slb /dev/culb0 9600 file1 file2
Kermit: Sending file1 as FILE1
Kermit: Sending file2 as FILE2
Kermit: done.
&
% ls
file1   file2
% ~.
Disconnected
$
```

SEE ALSO

umodem(1M), cu(1C), uucp(1C).

NAME

      killall - kill all active processes

SYNOPSIS

      /etc/killall [signal]

HP–UX COMPATIBILITY

      Level: HP–UX/STANDARD

      Origin: System III

      Remarks: Not supported on the Integral PC.

DESCRIPTION

      Killall is a procedure used by **/etc/shutdown** to kill all active processes not directly related to the shutdown 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 the specified *signal* to all user processes in the system, with the following exceptions:

            the *init* process;

            all processes (including background processes) associated with the terminal from which *killall* was invoked;

            any *ps -ef* process, if owned by *root*;

            any *sed -e* process, if owned by *root*;

            any *shutdown* process;

            any *killall* process;

            any */etc/rc* process.

      Killall obtains its process information from *ps*(1), and thus may not be able to perfectly identify which processes to signal.

      If no *signal* is specified, a default of **9** (kill) is used.

      Killall is invoked automatically by *shutdown*(1M). The use of *shutdown* is recommended over using *killall* by itself.

FILES

      /etc/shutdown

SEE ALSO

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

NAME
link, unlink - exercise link and unlink system calls

SYNOPSIS
/etc/link file1 file2

/etc/unlink file

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System III

Remarks: Not supported on the Integral PC.

DESCRIPTION
Link and *unlink* perform their respective system calls on their arguments, abandoning most error checking. These commands may only be executed by the super–user.

RETURN VALUE
0 – successful *link*.

1 – input syntax error.

2 – *link* call failed (*unlink* will never report failure).

SEE ALSO
rm(1), link(2), unlink(2).

NAME

    lpadmin - configure the LP spooling system

SYNOPSIS

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

HP–UX COMPATIBILITY

    Level: HP–UX/STANDARD

    Origin: System V

    Native Language Support:
        8-bit file names, 8-bit and 16-bit data, customs, messages

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**.

    -pic*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 referred 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 pathname of the new program.

    -l   indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*(1M), 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, C programs, or other executable programs which interface between *lpsched (1M)* and devices. All models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* -**m**. Models should have 644 permission if owned by lp and bin, or 664 permission if owned by bin and bin. Alternatively, LP administrators may modify copies of models and then use *lpadmin* -**i** to associate them with printers. See *mklp*(1M) for details of the printer models provided with your HP-UX system.

The LP model interface program does the actual printing on the device that is currently associated with the printer. The LP spooler sets standard in to **/dev/null** and standard out and standard error to the device specified in the -**v** option of *lpadmin*. The interface program is invoked then for printer **P** from the directory **/usr/spool/lp** as follows:

> interface/P id user title copies options file ...

id   is the request returned by *lp*.

user   is the logname of the user who made the request.

title   is the optional title specified with the -**t** option of *lp*.

copies   is the number of copies to be printed.

options   is a blank separated list of class -dependent or printer–dependent options specified with the -**o** option of *lp*.

file   is the full pathname of the file to be printed.

Given the command line arguments and the output directed to the device, interface programs may format their output in any way they choose.

When the printing is completed, it is the responsibility of the interface program to exit with a code indicative of the success of the print job. A return value of **0** indicates that the job completed successfully. Values of **1** to **127** indicate that some error was encountered. This problem will not effect future print jobs. *lpsched* notifies users by mail that there was an error in printing the request. When problems are detected which are likely to effect future print jobs, the interface program would be well to disable the printer so that print requests are not lost.

EXAMPLES

1.   Assuming there is an existing Hewlett–Packard 2934A line printer named *lp2*, it will use the **hp2934a** model interface after the command:

> /usr/lib/lpadmin -plp2 -mhp2934a

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1), mklp(1M), nroff(1).

NAME

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

SYNOPSIS

       /usr/lib/lpsched [-v]

       /usr/lib/lpshut

       /usr/lib/lpmove requests dest

       /usr/lib/lpmove dest1 dest2

HP–UX COMPATIBILITY

       Level: HP–UX/STANDARD

       Origin: System V

       Native Language Support:

              8-bit file names, 8-bit and 16-bit data, customs, messages

DESCRIPTION

       Lpsched schedules requests taken by *lp*(1) for printing on line printers. *Lpsched*(1M) is typically invoked in **/etc/rc**. This creates a process which runs in the background until *lpshut*(1M) is executed. The activity of the process is recorded in **/usr/spool/lp/log**. If the **-v** option is invoked, a verbose record of the *lpsched* process is captured.

       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*(1). The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp (1)* 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
    makekey - generate encryption key

SYNOPSIS
    /usr/lib/makekey

HP-UX COMPATIBILITY
    Level: HP-UX/STANDARD

    Origin: System V

DESCRIPTION
    Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

    The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and upper- and lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

    The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

    Makekey is intended for programs that perform encryption. Usually, its input and output will be pipes.

SEE ALSO
    passwd(5).

NAME

    mkdev – make device files

SYNOPSIS

    /etc/mkdev

HP–UX COMPATIBILITY

    Level: HP–UX/STANDARD

    Origin: HP

    Remarks: This command is implemented as a shell script, and will differ between the different implementations of HP–UX. This description applies to all versions.

               Not supported on the Integral PC.

DESCRIPTION

    This shell script helps the superuser install and maintain an HP–UX system. It consists of a machine–dependent list of commands which create one of each possible type of device file, with suggested default device addresses. It also creates mount directories for mountable volumes and changes permissions as appropriate for the device files.

    This command makes it easier to build (or rebuild) special files all at once.

    Mkdev automatically changes the working directory (using *cd*) to **/dev** before starting execution.

    Mkdev is specifically intended for modification before (each) use. Command lines for non–desired devices should be commented out with "#" so that they are still available for later use. You may want to use shorter device names than those suggested, especially for default devices. For HP–UX naming conventions, see *intro*(4).

DIAGNOSTICS

    Each command line in **mkdev** is echoed as it is executed. Error messages, if any, are generated by the commands invoked.

    Since the super–user must modify this script before using it the first time, an error is given if it has not been modified.

SEE ALSO

    mknod(1m), mkdir(1), chmod(1).

NAME

mkfs - construct a file system

SYNOPSIS

/etc/mkfs special size [nsect ntrack blksize fragsize ncpg minfree rps nbpi]

/etc/mkfs special proto [nsect ntrack blksize fragsize ncpg minfree rps nbpi]

HP-UX COMPATIBILITY

Level: HP–UX STANDARD

Origin: HP

DESCRIPTION

HFS file systems are normally created with the *newfs*(1M) command.

Mkfs constructs a file system by writing on the special file *special*. *size* specifies the number of DEV_BSIZE blocks in the file system. *Mkfs* builds a file system with a root directory and a *lost+found* directory. (see *fsck*(1M)) The FS_CLEAN magic number for the file system is stored in the super block.

The optional arguments allow fine tune control over the parameters of the file system. **Nsect** specifies the number of sectors per track on the disk. **Ntrack** specifies the number of tracks per cylinder on the disk. **Blksize** gives the primary block size for files on the file system. It must be a power of two, currently selected from 4096 or 8192. **Fragsize** gives the fragment size for files on the file system. The **fragsize** represents the smallest amount of disk space that will be allo-cated to a file. It must be a power of two currently selected from the range DEV_BSIZE to MAXBSIZE. **Ncpg** specifies the number of disk cylinders per cylinder group. This number must be in the range 1 to 32. **Minfree** specifies the minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the super–user is allowed to allocate disk blocks. The default value is 10%. If a disk does not revolve at 60 revolutions per second, the **rps** parameter may be specified. **nbpi** specifies the number of data bytes (amount of user file space) per i–node slot. The number of i–nodes is calculated as a function of the file system size. If **nbpi** is not valid, its value defaults to 2048.

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 (usually /etc/BOOT). If the name of a file is "" then it is ignored. The second token is a number specifying the number of DEV_BSIZE byte blocks in the file system. The next tokens comprise the specification for the root directory. File specifications consist of tokens giving the mode, the user–id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters -**bcd** specify regular, block special, character special and directory files respec-tively.) 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 giv-ing the owner, group, and other read, write, execute permissions, see *chmod*(1).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, *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:

```
/etc/BOOT
4872
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

SEE ALSO
chmod(1), dir(5), fs(5), fsck(1M), fsclean(1M), newfs(1M).

BUG
No way to specify links in the **proto** file.

NAME
mklp   configure the LP spooler subsystem

SYNOPSIS
/etc/mklp

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: HP

Remarks: This command is implemented as a shell script, and will differ between the different implementations of HP–UX. This description applies to all versions, and further details will be found in the commentary in the script.

DESCRIPTION
This shell script helps the superuser configure the printers into the LP spooler which are supported on the particular HP UX system. The administration of all printers in the LP spooler subsystem is similar, however in general there are *options* made available by the printer model which differ from printer to printer. These are described within the **mklp** script itself.

This command makes is easier to configure the LP spooler all at once. If desired, it can also be used to rebuild the subsystem.

While the **mklp** script gives some indication as to how the device special files are to be defined, the **mkdev** script should also be used in determining the major and minor number.

Mklp is specifically intended for modification before (each) use. Command lines for printers which will not be used should be commented out with "#" so that they are still available for later use.

SEE ALSO
lp(1), lpadmin(1M), mkdev(1M), mknod(1M).

DIAGNOSTICS
Each command line in **mklp** is echoed as it is executed. Error messages, if any, are generated by the commands invoked.

Since the super user must modify this script before using it the first time, an error is given if it has not been modified.

NAME

mknod - build special file

SYNOPSIS

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

HP–UX COMPATIBILITY

Level: HP–UX/NUCLEUS

Origin: System V

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. **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).

The *System Administrator's Manual* for your system.

NAME

      mount, umount - mount and dismount file system

SYNOPSIS

      /etc/mount [special directory [**-r**] [**-f**]]

      /etc/mount −**a**

      /etc/umount special

      /etc/umount −**a**

HP−UX COMPATIBILITY

      Level: HP−UX/EXTENDED

      Origin: System V

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. *Directory* must be given as an absolute path name.

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

      The optional argument **-r** indicates that the file system is to be mounted read−only. Physically write−protected 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.

      The **-f** option indicates that the file system should be mounted even if the file system clean flag indicates that the file system should be fsck'ed before mounting.

      If the **-a** option is present for either *mount* or *umount* , and all of the optional fields in */etc/checklist* are included and supported, all of the file systems described in */etc/checklist* are attempted to be mounted or dismounted. In this case, *special* and *directory* are taken from */etc/checklist*. The *special* file name used is the block special name from */etc/checklist*.

HARDWARE DEPENDENCIES

      Series 500:

            Warning: if virtual memory is brought up on a volume other than the root volume, and if that volume is then mounted, it cannot be dismounted.

FILES

      /etc/mnttab   mount table

      /etc/checklist file system table

SEE ALSO

      fsclean(1M), mount(2), mnttab(5), checklist(5).

DIAGNOSTICS

      Attempts to mount a currently−mounted volume under another name will result in an error [EBUSY].

      special and *directory* names recorded in */etc/mnttab* are truncated to MNTLEN bytes.

      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

mvdir - move a directory

SYNOPSIS

/etc/mvdir dirname   name

HP-UX COMPATIBILITY

Level: HP–UX/NUCLEUS

Origin: System V

Native Language Support:
8-bit filenames.

Remarks: Not supported on the Integral PC.

DESCRIPTION

Mvdir moves and/or renames directories within a file system. *Dirname* must be a directory; *name* must not exist. Neither name may be a sub-set of the other (**/x/y** cannot be moved to **/x/y/z**, nor vice versa).

The directory specified by *name* cannot be a subdirectory of that specified by *dirname*. The directory specified by *dirname* may be a subdirectory of that specified by *name*, but the notations . and .. must be used in naming the directories, because *mvdir* does not allow the *names* of the directories to have the property that one is a subdirectory of the other.

Only the super–user can use *mvdir*.

EXAMPLE

The following moves the directory specified by **/x/y/z** to **/a/b/c**:

mvdir /x/y/z /a/b/c

SEE ALSO

mkdir(1).

BUGS

The restriction on names is intended to prevent creation of a (cyclic) sub-tree that cannot be reached from the root. The test is strictly by name, thus creating such a sub-tree is still possible. The super–user is cautioned to be very careful in his use of the names . and .. while moving directories.

NAME

ncheck - generate names from i numbers

SYNOPSIS

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

HP–UX COMPATIBILITY

Level: HP–UX/NUCLEUS

Origin: HP

Remarks: *Ncheck* is implemented only on systems supporting the Berkeley file system.

Not supported on the Integral PC.

DESCRIPTION

Ncheck with no argument generates a path–name vs. i–number list of all files on the volumes specified by the file **/etc/checklist.** Names of directory files are followed by **/..** The options are as follows:

-i   reduces the report to only those files whose i–numbers are specified on the command line in the *numbers* list.

-a   allows printing of the names . and .., which are ordinarily suppressed.

-s   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), checklist(5).

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
newfs - construct a new file system

SYNOPSIS
/etc/newfs [**−v**] [**−n**] [**mkfs−options**] **special disk−type**

HP−UX COMPATIBILITY
Level: HP−UX/EXTENDED

Origin: UCB

DESCRIPTION
Newfs is a "friendly" front-end to the *mkfs*(1M) program. *Newfs* will look up the type of disk a file system is being created on in the disk description file */etc/disktab*, calculate the appropriate parameters to use in calling *mkfs*, then build the file system by forking *mkfs* and, if the file system is a root section, install the necessary bootstrap programs in the initial 8192 bytes of the device. The **-n** option prevents the bootstrap programs from being installed. **special** is the character special file for the disc and **disk−type** is the type of the disc as specified in /etc/disktab.

If the **-v** option is supplied, *newfs* will print out its actions, including the parameters passed to *mkfs*.

Options which may be used to override default parameters passed to *mkfs* are:

-s size   The size of the file system in DEV__BSIZE blocks.

-b block−size
The block size of the file system in bytes.

-f frag−size
The fragment size of the file system in bytes.

-t #tracks/cylinder
The number of tracks per cylinder.

-c #cylinders/group
The number of cylinders per cylinder group in a file system. The default value used is 16.

-m free space %
The percentage of space reserved from normal users; the minimum free space threshold. The default value used is 10%.

-r revolutions/minute
The speed of the disk in revolutions per minute (normally 3600).

-i number of bytes per inode
This specifies the density of inodes in the file system. The default is to create an inode for each 2048 bytes of data space. If fewer inodes are desired, a larger number should be used; to create more inodes a smaller number should be given.

FILES
/etc/disktab   for disk geometry and file system section information

SEE ALSO
disktab(5), fs(5), fsck(1M), mkfs(1M), tunefs(1M).

NAME

opx25 - execute HALGOL programs

SYNOPSIS

opx25 [-f scriptname] [-c char] [-o file-descriptor] [-i file-descriptor] [-n string] [-d] [-v]

HP–UX COMPATIBILITY

Level: HP–UX/NON–STANDARD

Origin: HP

DESCRIPTION

HALGOL is a simple language for communicating with devices such as modems and X.25 PADs. It has simple statements like **send xxx** and **expect yyy** which are descibed below.

Options:

-f *script*

Causes *opx25* to read *script* as the input program. If −f is not specified, then *opx25* reads *stdin* for the script.

-c *char* Causes *opx25* to use *char* as the first character in the input stream instead of actually reading it from the input descriptor. This is sometimes useful when the program that calls *opx25* is forced to read a character but then cannot "unread" it.

-o *number*

Causes *opx25* to use *number* for the output file descriptor (i.e., the device to use for **send**). The default is 1.

-i *number*

Causes *opx25* to use *number* for the input file descriptor (i.e., the device to use for **expect**). The default is 0.

-n *string*

Causes *opx25* to save this string for use when "\#" is encountered in a **send** command.

-d   Causes *opx25* to turn on debugging mode.

-v   Causes *opx25* to turn on verbose mode.

An *opx25* script file contains lines of the following type:

(empty)

Empty lines are ignored.

/   Lines beginning with a slash "/" are ignored (comments)

ID   ID denotes a label. ID is limited to alphanumerics or "_".

send STRING

STRING must be surrounded by double quotes. The text is sent to the device specified by the −o option. Non–printable characters are represented as in C, i.e., as \DDD, where DDD is the octal ASCII character code. "\#" in a **send** string is the string that followed the −n option.

break   Send a break "character" to the device.

expect NUMBER STRING

Here NUMBER is how many seconds to wait before giving up. 0 means wait forever, but this isn't advised. Whenever STRING appears in the input within the time allotted, the command succeeds. Thus, it isn't necessary to specify the entire string. For example, if you know that the PAD will send several lines followed by a "@" prompt, you could just use "@" as the string.

run *program args*

The program (*sleep*, *date*, or whatever) is run with the *args* specified. Don't use quotes

here. Also, the program is invoked directly (with *execp*), so wild cards, redirection, etc. are not possible.

error ID

If the most recent **expect** or **run** encountered an error, go to the label ID.

exec *program args*

Like **run**, but doesn't fork.

echo STRING

Like **send**, but goes to *stderr* instead of to the device.

set debug

Sets the program in debug mode. It echoes each line to */tmp/opx25.log*, as well as giving the result of each **expect** and **run**. This can be useful for writing new scripts. The command "set nodebug" will turn off this feature.

set log Sends subsequent incoming characters to */usr/spool/uucp/X25LOG*. This can be used in the *.in file as a security measure, since part of the incoming data stream contains the number of the caller. There is a similar feature in *getx25*: it writes the time and the login name into the same logfile. The command "set nolog" will turn off this feature.

set numlog

Like "set log," but better in some cases, because it sends **only** digits to the log file. The command "set nonumlog" will turn off this feature.

timeout NUMBER

Sets a global timeout value. Each **expect** uses time in the timeout reservoir; when this time is gone, the program gives up (exit 1). If this command isn't used, there is no global timeout. Also, the global timeout can be reset any time, and a value of 0 turns it off.

exit NUMBER

Exits with this value. 0 is success, anything else is failure.

You can crudely test configuration files by running *opx25* by hand, using the argument −f followed by the name of the script file. The program in this case **send**s to, and **expect**s from, standard output and input, so you can type the input, observe the output, and see messages with the "echo" command. See the file */usr/lib/uucp/X25/ventel.out* for a good example of HALGOL programming.

SEE ALSO

getx25(1C), uucp(1C).
Serial Network Communications Guide

NAME
osck - check integrity of OS in SDF boot area(s)

SYNOPSIS
/etc/osck [**-v**] volume

HP–UX COMPATIBILITY
Level: HP UX/NON–STANDARD

Origin: HP

Remarks: *Osck* is implemented on the Series 500 only.

DESCRIPTION
Osck checks one operating system in the boot area on the volume specified by *volume* (a character special file).

The OSF must be the first section of an n-section operating system. If n is greater than one, *osck* prompts for additional volumes as needed. The volumes must be mounted in order.

The **-v** (verbose) option causes *osck* to print additional information about each volume and each code segment as they are encountered. If **-v** is not specified, it is silent except for warnings, errors, and prompts for new volumes.

Osck checks the following:

       OSF headers are valid and consistent across multiple volumes;

       the first code segment is a power up segment;

       the code segment chain contains correct headers and lengths;

       all segment checksums are correct;

       the system terminates correctly after the last segment.

SEE ALSO
oscp(1M), osmark(1M), osmgr(1M), sdfinit(1M).

DIAGNOSTICS
Osck gives an appropriate error message and returns a non–zero value if *volume* cannot be accessed or is not an SDF volume, there is no boot area, or the boot area contents appear invalid. Error messages are also given if any integrity violation is found. See *osmgr*(1M) for a complete list of return values.

NAME

oscp - copy, create, append to, split operating system

SYNOPSIS

/etc/oscp [**-o**] [**-v**] fromvolume tovolume
/etc/oscp -m [**-v**] file ... tovolume
/etc/oscp -a [**-v**] file ... tovolume
/etc/oscp -s [**-v**] fromvolume
/etc/oscp -f [**-v**] fromvolume tofile

HP–UX COMPATIBILITY

Level: HP–UX/NON–STANDARD

Origin: HP

Remarks: *Oscp* is implemented on the Series 500 only.

DESCRIPTION

Oscp enables you to perform:

boot–to–boot copy
Copy an operating system from the boot areas on one or more SDF volumes to the boot area on one SDF volume;

files–to–boot copy (**-m**, **-a** options)
Create a new operating system or append to an existing operating system from a list of ordinary files, and put the resulting system in one boot area;

boot–to–files copy (**-s** option)
Split up the segments in an operating system from one or more boot areas to one or more ordinary files.

boot–to–file copy (**-f** option)
Split up the segments in an operating system from one or more boot areas to a single ordinary file.

Fromvolume and *tovolume* are usually character special files.

Boot–to–Boot Copy

If **-m**, **-a**, **-s**, and **-f** are not specified, *oscp* does boot–to–boot copy. For normal, multi–volume boot–to–boot copy, *oscp* requires that the OSF on the first *fromvolume* be the first section of an *n*–section operating system. If *n* is greater than one, *oscp* prompts you for additional volumes as required. The additional volumes must be mounted in order.

Before starting the copy, *oscp* clears the OSF header on *tovolume*. The OSF header values are corrected on *tovolume* after the copy is done. This new header may include a new system ID string that you enter when you are prompted (the same ID string displayed by the boot loader).

The **-o** (one volume only) option tells *oscp* to copy only one OSF (which may be part or all of a system) from *fromvolume* to *tovolume*, without changing the OSF header.

The **-v** (verbose) option tells *oscp* to print additional information about each volume as it is encountered. Otherwise, *oscp* is silent except for warnings, errors, and prompts for new volumes and new system ID strings.

Files–to–Boot Copy

If the **-m** (merge) option is given, *oscp* does a files–to–boot copy from the specified *files*. The source files may be BASIC/9000 BIN files or HP–UX ordinary files. The *files* must all be accessible and contain valid code segments. The code segments must all be of the same system type. The last code segment in each file must be followed by two null bytes.

Note that segments of unknown type, and old power–up segments (before February 1983) are "generic donors", and may be merged with any other type. Also note that, when creating a new

Hewlett–Packard   – 1 –   June 28, 1985

system, *oscp* uses the first OSF header magic number in its internal list (i.e. 0xE9C28206).

Once you enter the new system ID string, *oscp* destroys the old OSF (if any) in the boot area before writing the new system.

The **-a** (append) option allows you to append code segments from ordinary files to an existing OSF on *tovolume*. There must be enough unused space in the boot area after the OSF, and the OSF must be a complete system in itself (i.e. volume 1 of 1). The existing OSF is not invalidated until the last segment is copied to the boot area.

In conjunction with **-m** or **-a**, the **-v** (verbose) option gives you additional information about the boot area and each segment as it is encountered.

Boot–to–Files Copy

The **-s** (split) option allows you to split an operating system into one or more ordinary files (HP–UX ordinary files only, not BASIC BIN files). For each code segment in the operating system, you are prompted for a file name to which the code segment is appended. If you enter a null line, the code segment is appended to the same file as was used in the previous append operation.

If the size of the specified file is greater than zero, *oscp* backs up two bytes from the end of the file to overwrite the previous terminator before appending the code segment to the file.

The **-v** (verbose) option gives you additional information about the boot area and each segment as it is encountered.

Note that the resulting ordinary files may be owned by the owner of the *oscp* command, depending on its permissions.

Boot–to–File Copy

The **-f** option allows you to split an operating system into a single ordinary file (*tofile*), eliminating any user interaction (except possibly to change certain types of media, if that is where the boot area is located). Otherwise, this option behaves exactly like the **-s** option.

Copying to Boot Areas

Before beginning the copy, *oscp* prompts you for the 80–character operating system ID string to use for all volumes.

Before writing to *tovolume*, *oscp* first checks that it contains a boot area with sufficient unused space.

SEE ALSO

osck(1M), osmark(1M), osmgr(1M), sdfinit(1M).

DIAGNOSTICS

Oscp prints an appropriate error message and returns a non–zero value if *fromvolume* or *tovolume* cannot be accessed or is not an SDF volume, there is no boot area, the boot area contents appear invalid, or the source OSF is not section 1 of an *n*–section system.

Errors are also given if:
       fromvolume and *tovolume* are the same (by name);
       *fromvolume*s are mounted out of order;
       a specified ordinary file is inaccessible or has invalid contents;
       the first segment is not a power–up segment;
       any segment has a mismatching system type.

See *osmgr*(1M) for the exact list of return values.

BUGS

Oscp -a checks that all appended segments are mutually compatible, but it does not check them against the segments in the existing OSF.

Performing an *oscp -a* to a boot area with less than 1024 free bytes results in an error before the copy completes.

Before appending, *oscp -s* backspaces over the existing two–null–byte terminator at the end of each ordinary file, but it does not check that the bytes overwritten were actually two null bytes.

A boot area of less the 1024 bytes, at the end of a volume, results in a read error.

.

NAME

osmark - mark SDF volume boot area as loadable/non-loadable

SYNOPSIS

/etc/osmark [-m | -u] [-v] volume

HP-UX COMPATIBILITY

Level: HP-UX/NON-STANDARD

Origin: HP

Remarks: *Osmark* is implemented on the Series 500 only.

DESCRIPTION

Osmark marks an operating system file (OSF) in a boot area as loadable (**-m** option) or non-loadable (**-u** option). *Volume* is usually a character special file specifying the SDF volume on which the boot area is found.

If neither **-m** nor **-u** are specified, *osmark* reports the status of the OSF.

The **-v** (verbose) option causes *osmark* to print additional information about the volume in the same format as that used by *osck* and *oscp*.

When dealing with a multi-volume operating system, be sure that each OSF in the system is properly marked, not just the first.

SEE ALSO

osck(1M), oscp(1M), osmgr(1M).

DIAGNOSTICS

Osmark outputs an appropriate error message and returns a non-zero value if *filespec* cannot be accessed or is not an SDF volume, there is no boot area, or the boot area contents appear invalid. Refer to *osmgr*(1M) for a list of possible return values.

NAME
osmgr - operating system manager package description

HP-UX COMPATIBILITY
Level: HP-UX/NON-STANDARD

Origin: HP

Remarks: This entry describes the operating system manager package, which is implemented on the Series 500 only.

Not supported on the Integral Personal Computer.

DESCRIPTION
This group of three commands helps you manage the operating systems which reside in the boot areas on your Structured Directory Format (SDF) volumes. The package includes:

oscp   copy systems or create them from ordinary files;

osck   check operating system integrity;

osmark   mark an operating system file as loadable or not loadable, or inquire about current state of operating system file.

Oscp, osck, and *osmark* are multiple links to a single program.

Boot Areas:
Each SDF volume has one boot area consisting of zero or more contiguous logical blocks. The boot area is completely outside the file area. Its size is determined when the volume is initialized. To change the size of a boot area, you must re-initialize the volume.

Each boot area may contain at most (one part of) one operating system.

The logical block size for a boot area is the same as that for the rest of the volume (i.e., whatever size you request when you initialize the volume).

Operating Systems:
Every HP 9000 operating system consists of a series of code segments. An operating system may reside in the boot area on one volume, or it may be distributed in sections over several volumes (not necessarily with a whole number of segments per volume).

An operating system can also reside in a number of ordinary files, each containing a whole number of segments, and terminated by two null bytes. This is the same format used for BASIC/9000 BIN files. In this form, the system is not loadable, but its files can be combined into a loadable system by *oscp.*

Operating System Files:
Each boot area contains zero or one operating system files (OSF's). If an operating system resides in sections in several boot areas, each section occupies one OSF on one SDF volume.

Operating System File Headers:
Each OSF starts with a header that includes a "loadable" flag, a volume number, and the total number of volumes over which this operating system is distributed. The loader only boots an OSF if it is marked loadable. If required, it requests additional volumes until it has loaded from all volumes in the set. You should ensure that all parts of a multi-volume operating system are marked loadable.

Each OSF header also includes an 80-character identification string. The loader displays this string before it starts to load from each volume.

RETURN VALUES
The following list contains all the possible return values, mnemonics, and meanings given by OS manager commands:

0   no error;

1	USAGE	bad argument list;
2	FILESYS	error during file system access;
3	VOLSEQ	volumes mounted out of order;
4	VOLCONT	bad volume (not SDF, no boot area, etc.);
5	HEADER	invalid or inconsistent OSF header(s);
6	FIRSTSEG	first segment is not a power-up segment;
7	SEGTYPE	incompatible segment system types or revisions;
8	SEGLEN	segment length out of range or not whole words;
9	CHECKSUM	segment checksum does not match reference value;
10	TERM	system terminator (″-1″ word) missing.

SEE ALSO

osck(1M), oscp(1M), osmark(1M), sdfinit(1M).

NAME

      pwck, grpck - password/group file checkers

SYNOPSIS

      /etc/pwck [file]
      /etc/grpck [file]

HP–UX COMPATIBILITY

      Level: HP–UX/STANDARD

      Origin: System V

      Remarks: Not supported on the Integral PC.

DESCRIPTION

      Pwck scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The criteria for determining a valid login name are taken from *HP–UX System Administrator's Manual* for your system . The default password file is **/etc/passwd**.

      Grpck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group**.

FILES

      /etc/group
      /etc/passwd

SEE ALSO

      group(5), passwd(5).
      The HP–UX System Administrators Manual

DIAGNOSTICS

      Group entries in **/etc/group** with no login names are flagged.

NAME

     reboot - reboot the system

SYNOPSIS

     /etc/reboot [**-h** | **-r**] [**-n** | **-s**] [**-d** device] [**-f** lif_filename]

HP–UX COMPATIBILITY

     Level: HP–UX/NUCLEUS

     Origin: HP

     Remarks: *Reboot* is implemented on the Series 200 only.

DESCRIPTION

     Reboot brings down the system and then halts or re–boots system. Reboot with no argument syncs all disks and does proper shutdown before re–booting the system. The options are as fol-lows:

     -h   shutdown the system and halt.

     -r   shutdown the system and re–boot automatically. (default)

     -n   no sync before shutdown.

     -s   sync before shutdown. (default)

     -d   specified the device will be used when re–boot. The device has to be a lif volume. (can't use with –h)

     -f   the name/id of the system to start. If the it is a NULL string then the powerup search sequence will be made for a system. Otherwise, the filename has to follow lif file name convention. (can't use with –h)

     All the processes excluding proc0, proc1 and reboot itself will be killed before shutdown.

     Reboot can be executed by root only and should be executed in *single–user*state.

SEE ALSO

     init(1), lif(1), stopsys(1M), reboot(2).

NAME
     revck - check internal revision numbers of HP–UX files

SYNOPSIS
     /etc/revck ref_files

HP–UX COMPATIBILITY
     Level: HP–UX/NON–STANDARD

     Origin: HP

     Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION
     Revck checks the internal revision numbers of lists of files against reference lists. Each *ref_file*
     must contain a list of absolute path names (each beginning with "/") and *whatstrings* (revision
     information strings from *what*(1)). Path names begin in column one of a line, and have a colon
     appended to them. Each path name is followed by zero or more lines of *whatstrings*, one per line,
     each indented by at least one tab (this is the same format in which *what*(1) outputs its results).

     For each path name, *revck* checks that the file exists, and that executing *what*(1) on the current
     path name produces results identical to the *whatstrings* in the reference file. Only the first 1024
     bytes of *whatstrings* are checked.

     Ref_files are usually the absolute path names of the *revlist* files shipped with HP–UX. Each
     HP–UX software product includes a file named /system/*product*/revlist (for example,
     /system/97070A/revlist). The *revlist* file for each product is a reference list for the ordinary files
     shipped with the product, plus any empty directories on which the product depends.

FILES
     /system/*product*/revlist   lists of HP–UX files and revision numbers

SEE ALSO
     what(1).

DIAGNOSTICS
     Revck is silent except for reporting missing files or mismatches. If a *ref_file* is not in the right
     format, you will get unpredictable results.

NAME

rootmark - mark/unmark volume as HP–UX root volume

SYNOPSIS

/etc/rootmark [**-m** | **-u**] filespec

HP–UX COMPATIBILITY

Level: HP–UX/NON–STANDARD

Origin: HP

Remarks: *Rootmark* is implemented on the Series 500 only.

Not supported on the Integral PC.

DESCRIPTION

Rootmark enables you to control which mass storage device contains your HP–UX root (/) direc-
tory. The HP–UX operating system searches mass storage devices and uses the first root volume
it finds.

Filespec is usually a character special file which points to a mass storage volume initialized with
Structured Directory Format (SDF). If invoked with no option, *rootmark* tells the current state
of the specified volume. If **-m** is specified, then the specified volume is marked as a root volume.
If **-u** is specified, the specified volume is marked as not a root volume. *Rootmark* is silent if suc-
cessful.

RETURN VALUE

Rootmark sends an error message to standard error and returns a non–zero value if it cannot read
or write a volume, or if a volume is not SDF. *Rootmark* returns 1 for incorrect syntax, 2 for a file
system problem, and 3 for a volume that is not in SDF.

EXAMPLE

The following example makes **/dev/rhd** usable as root; you must super–user to execute the
example:

```
# rootmark /dev/rhd      # check if /dev/rhd is a root volume
/dev/rhd is marked as NOT a root volume.
# rootmark –m /dev/rhd  # mart it as the root volume
# rootmark /dev/rhd      # check results
/dev/rhd is marked as a root volume.
```

SEE ALSO

mount(1), osmgr(1M), sdfinit(1M).

WARNINGS

A volume must not be marked as a root volume unless it contains all the directories and files that
HP–UX requires for system initialization.

Never mark any media shipped from Hewlett–Packard as not a root volume, in case you need to
re–install HP–UX from that media.

NAME

runacct - run daily accounting

SYNOPSIS

/usr/lib/acct/runacct [mmdd [state]]

HP–UX COMPATIBILITY

Level: HP–UX/EXTENDED

Origin: System V

DESCRIPTION

Runacct is the main daily accounting shell procedure. It is normally initiated via *cron*(1M). *Runacct* processes connect, fee, disk, and process accounting files. It also prepares summary files for *prdaily* or billing purposes.

Runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into **active**. When an error is detected, a message is written to **/dev/console**, mail (see *mail*(1)) is sent to **root** and **adm**, and *runacct* terminates. *Runacct* uses a series of lock files to protect against re–invocation. The files **lock** and **lock1** are used to prevent simultaneous invocation, and **lastdate** is used to prevent more than one invocation per day.

Runacct breaks its processing into separate, restartable *states* using **statefile** to remember the last *state* completed. It accomplishes this by writing the *state* name into **statefile**. *Runacct* then looks in **statefile** to see what it has done and to determine what to process next. *States* are executed in the following order:

SETUP	Move active accounting files into working files.
WTMPFIX	Verify integrity of **wtmp** file, correcting date changes if necessary.
CONNECT1	Produce connect session records in **ctmp.h** format.
CONNECT2	Convert **ctmp.h** records into **tacct.h** format.
PROCESS	Convert process accounting records into **tacct.h** format.
MERGE	Merge the connect and process accounting records.
FEES	Convert output of *chargefee* into **tacct.h** format and merge with connect and process accounting records.
DISK	Merge disk accounting records with connect, process, and fee accounting records.
MERGETACCT	Merge the daily total accounting records in **daytacct** with the summary total accounting records in **/usr/adm/acct/sum/tacct**.
CMS	Produce command summaries.
USEREXIT	Any installation–dependent accounting programs can be included here.
CLEANUP	Cleanup temporary files and exit.

To restart *runacct* after a failure, first check the **active** file for diagnostics, then fix up any corrupted data files such as **pacct** or **wtmp**. The **lock** files and **lastdate** file must be removed before *runacct* can be restarted. The argument *mmdd* is necessary if *runacct* is being restarted, and specifies the month and day for which *runacct* will rerun the accounting. Entry point for processing is based on the contents of **statefile**; to override this, include the desired *state* on the command line to designate where processing should begin.

EXAMPLES

To start *runacct*.

    nohup runacct 2> /usr/adm/acct/nite/fd2log &

To restart *runacct*.

    nohup runacct 0601 2>> /usr/adm/acct/nite/fd2log &

To restart *runacct* at a specific *state*.

    nohup runacct 0601 MERGE 2>> /usr/adm/acct/nite/fd2log &

FILES

/etc/wtmp
/usr/adm/pacct*
/usr/adm/acct/nite/active
/usr/adm/acct/nite/daytacct
/usr/adm/acct/nite/lock
/usr/adm/acct/nite/lock1
/usr/adm/acct/nite/lastdate
/usr/adm/acct/nite/statefile
/usr/adm/acct/nite/ptacct*.*mmdd*

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), mail(1), acct(2), acct(5), utmp(5).

Chapter 6, "System Accounting," of the *HP-UX System Administrator Manual*.

BUGS

Normally it is not a good idea to restart *runacct* in the **SETUP** *state*. Run **SETUP** manually and restart via:

    runacct *mmdd* **WTMPFIX**

If *runacct* failed in the **PROCESS** *state*, remove the last **ptacct** file because it will not be complete.

NAME
sdfinit - initialize Structured Directory Format volume

SYNOPSIS
/etc/sdfinit [–i] pathname [blocksize [bootsize [interleave]]]

HP-UX COMPATIBILITY
Level: HP-UX/NUCLEUS

Origin: HP

Remarks: Not supported on the Integral PC.

DESCRIPTION
Pathname refers to a character or block special file, which must be accessible and not mounted.

Blocksize is the number of bytes per logical block. It is rounded up, if necessary, to the next multiple of the physical record size for the volume. If absent or less than one (1), the system sets a reasonable default for you.

Bootsize is the number of bytes to allocate for the boot area on the volume. It is rounded up to a whole number of logical blocks. It defaults to zero (no boot area).

Interleave is the sector interleave factor. It defaults to one (not necessarily the best value for all devices).

The root directory on the newly–initialized volume is always owned by the super–user and has permissions of 777.

The $-i$ option inhibits formatting and certification, so the volume is only initialized. That is, only a directory structure is written. This saves a considerable amount of time in most cases. However, the $-i$ option is **NOT RECOMMENDED** for most removable media, unless it was recently formatted and certified in the same type of drive.

RESTRICTIONS
The effective user ID must be zero (super–user). The disc must not be mounted.

DIAGNOSTICS
Appropriate error messages are given if the argument list is incorrect, pathname cannot be initialized or any other error occurs.

SEE ALSO
osmgr(1M), sections on device drivers

NAME

setmnt - establish mount table mnttab

SYNOPSIS

/etc/setmnt

HP–UX COMPATIBILITY

Level: HP–UX/NUCLEUS

Origin: System V

Remarks: Not supported on the Integral PC.

DESCRIPTION

Setmnt creates the **/etc/mnttab** table (see *mnttab*(5)), 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., "dsk/?s?") and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the *mnttab*(5) entry.

FILES

/etc/mnttab

SEE ALSO

mount(1M), rc(1M), mnttab(5).

BUGS

Filesys and *node* are truncated to MNTLEN bytes. The minimum value for MNTLEN is 32.
Setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.
It is unwise to use *setmnt* to create false entries for *mount*(1) and *umount*(1).

NAME

      setprivgrp - set special attributes for group

SYNOPSIS

      setprivgrp -g | -n | group-name [privileges]

      setprivgrp -f file

HP-UX COMPATIBILITY

      Level: HP-UX/RT

      Origin: HP

DESCRIPTION

      Setprivgrp associates a group with a kernel capability. This allows subsetting of super-user-like privileges for members of a particular group or groups. In the first form, the first argument to *setprivgrp* is either a *group-name*, **-g**, or **-n** which specifies a particular group, all groups, or no groups, respectively. The optional second and subsequent arguments are symbolic names indicating kernel capabilities. In the second form, the **-f** option is used to specify a file, typically */etc/privgroup*, from which group capabilities are set. The group access privileges are changed to reflect the specified kernel capabilities.

      RTPRIO   gives access to the *rtprio*(2) system call for setting real-time priorities.

      MLOCK   gives access to the *plock*(2) system call for locking process text and data into memory, and the SHM_LOCK command used with *shmctl*(2) system call.

      CHOWN   gives access to the *chown*(2) system call.

      Specifying no access privileges removes any privileges that may currently be assigned. Note that capabilities set by this command are not additive. If you wish to add a capability for a particular group, you need to respecify all capabilities that were already set for that group in addition to the new capability.

      The file named using the **-f** option should contain one or more lines in the following format:

            -g | -n | group-name [privileges]

      Only the super user may use this command.

FILES

      /etc/privgroup
      /etc/group

ERRORS

      Setprivgrp returns 1 if caller is not super user, and 2 if there is not enough table space to hold a new privileged group assignment.

SEE ALSO

      getprivgrp(1), getprivgrp(2), rtprio(2), plock(2), shmctl(2), chown(2)

      not found

NAME

        shutdown - terminate all processing

SYNOPSIS

        /etc/shutdown [grace]

HP–UX COMPATIBILITY

        Level: HP–UX/STANDARD

        Origin: System III

        Remarks: Not supported on the Integral PC.

DESCRIPTION

        Shutdown is part of the HP–UX system operation procedures. Its primary function is to terminate all currently running processes in an orderly and cautious manner. The procedure is designed to interact with the operator (i.e., the person who invoked *shutdown*). *Shutdown* may instruct the operator to perform some specific tasks, or to supply certain responses before execution can resume. *Shutdown* goes through the following steps:

        All users logged on the system are notified to log off the system by a broadcasted message. The operator may display his/her own message at this time. Otherwise, the standard file–save message is displayed.

        If the operator wishes to run the file–save procedure, *shutdown* unmounts all file systems.

        All file systems' super blocks are updated before the system is to be stopped (see *sync*(1M)). This must be done before re-booting the system, to insure file system integrity. The most common error diagnostic that will occur is *device busy*. This diagnostic happens when a particular file system could not be unmounted. See *umount*(1M).

        Grace specifies, in seconds, a grace period for users to log off before shutting down. The default is 60 seconds. If *grace* is zero, *shutdown* runs more quickly and gives the user very little time to log out.

HARDWARE DEPENDENCIES

        Series 500:

                A file–save procedure is not implemented.

SEE ALSO

        killall(1M), mount(1M), sync(1M).

NAME

    stopsys - stop operating system with optional reboot

SYNOPSIS

    /etc/stopsys [**-r**]

HP-UX COMPATIBILITY

    Level: HP-UX/NON-STANDARD

    Origin: HP

    Remarks: *Stopsys* is implemented on the Series 500 only.

DESCRIPTION

    Stopsys dumps all system I/O buffers to mass storage volumes (i.e. performs a *sync*(1M)), and shuts down all virtual memory activity. Then, *stopsys* either stops the operating system so that the hardware may be powered down (no option), or it reboots the system (resets the machine's processor(s) to the power-on state) (**-r** option). The reboot (**-r**) option results in the activation of the system boot loader, almost exactly as if the power was just turned on, except that I/O cards are not power-cycled.

    Just before it stops the system, *stopsys* writes a message to /dev/console indicating that the system is stopped and can be safely powered down.

    Stopsys may be invoked only by the effective super-user. However, it may be made public by setting the set-user-ID bit and assigning ownership to *root*.

    Stopsys does not ensure that the system is idle. If any user processes are running, the *sync*(1M) may be ineffective. You should execute *shutdown*(1M), or at least kill all non-essential processes, prior to running *stopsys*.

SEE ALSO

    chsys(1M), killall(1M), shutdown(1M), sync(1M).

DIAGNOSTICS

    Stopsys returns only if a non-fatal error occurs, in which case it writes a message to standard error and returns 1. Non-fatal errors include:

        invocation with improper arguments;
        invocation by other than the effective super-user;
        any failure to stop the system, as long as the system is still usable.

    If *stopsys* fails to stop the system for any reason, but the system is then not in a usable state, *stopsys* writes an error message to /dev/console and then attempts to reboot (if **-r** was specified). If **-r** was not specified, or if the reboot attempt fails, *stopsys* writes "system stopped" on /dev/console, and you must reboot the system yourself (using the power switch or the front panel).

    Note that if the reboot fails it indicates a hardware problem with the HP 9000 Model 20 keyboard on select code 6, or the HP 9000 Model 30/40 system control module on select code 7.

BUGS

    At this time, *stopsys* does not shut down Local Area Net (LAN) activity.

NAME

swapon - enable additional device for paging and swapping

SYNOPSIS

/etc/swapon -a

/etc/swapon name ...

HP–UX COMPATIBILITY

Level: HP–UX/EXTENDED

Origin: UCB

DESCRIPTION

Swapon is used to enable additional devices on which paging and swapping are to take place. The system begins by swapping and paging on only a single device so that only one disk is required at bootstrap time. Calls to *swapon* normally occur in the system multi–user initialization file */etc/rc* making all swap devices available, so that the paging and swapping activity is interleaved across several devices.

Normally, the **-a** argument is given, causing all devices marked as "sw" swap devices in **/etc/checklist** to be made available.

The second form announces individual block devices to be used for paging and swapping. These block devices must have been setup at system configuration time.

HARDWARE DEPENDENCIES

Series 200: Series 200 does not support swapping on multiple devices.

FILES

/dev/dsk/#s#   Normal paging devices.

HARDWARE DEPENDENCIES

Not implemented on Series 500 and Integral PC.

SEE ALSO

swapon(2).

BUGS

There is no way to stop paging and swapping on a device. It is therefore not possible to make use of devices which may be dismounted during system operation.

NAME
tic - terminfo compiler

SYNOPSIS
tic [**-v**[*n*]] file ...

HP–UX COMPATIBILITY
Level: HP–UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral PC.

DESCRIPTION
Tic translates terminfo files from the source format into the compiled format. The results are placed in the directory **/usr/lib/terminfo**.

The **-v** (verbose) option causes *tic* to output trace information showing its progress. If the optional integer is appended, the level of verbosity can be increased.

Tic compiles all terminfo descriptions in the given files. When a **use=** field is discovered, *tic* searches first the current file, then the master file, which is "./terminfo.src".

If the environment variable TERMINFO is set, the results are placed there instead of **/usr/lib/terminfo**.

Some limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

FILES
/usr/lib/terminfo/?/*   compiled terminal capability data base

SEE ALSO
curses(3X), terminfo(5).

BUGS
Instead of searching **./terminfo.src**, it should check for an existing compiled entry.

NAME

tunefs - tune up an existing file system

SYNOPSIS

/etc/tunefs *tuneup-options special|filesys*

HP–UX/COMPATIBILITY

Level: HP–UX/EXTENDED

Origin: UCB

Remarks: *Tunefs*(1m) is implemented only on those machines implementing the HFS file sys-
            tems.

DESCRIPTION

Tunefs is designed to change the dynamic parameters of a file system which affect the layout poli-
cies. The parameters which are to be changed are indicated by the flags given below:

-a maxcontig

> This specifies the maximum number of contiguous blocks that will be laid out before forc-
> ing a rotational delay (see -d below). The default value is one, since most device drivers
> require an interrupt per disk transfer. Device drivers that can chain several buffers
> together in a single transfer should set this to the maximum chain length.

-d rotdelay

> This specifies the expected time (in milliseconds) to service a transfer completion inter-
> rupt and initiate a new transfer on the same disk. It is used to decide how much rota-
> tional spacing to place between successive blocks in a file.

-e maxbpg

> This indicates the maximum number of blocks any single file can allocate out of a cylinder
> group before it is forced to begin allocating blocks from another cylinder group. Typically
> this value is set to about one quarter of the total blocks in a cylinder group. The intent is
> to prevent any single file from using up all the blocks in a single cylinder group, thus
> degrading access times for all files subsequently allocated in that cylinder group. The
> effect of this limit is to cause big files to do long seeks more frequently than if they were
> allowed to allocate all the blocks in a cylinder group before seeking elsewhere. For file
> systems with exclusively large files, this parameter should be set higher.

-m minfree

> This value specifies the percentage of space held back from normal users; the minimum
> free space threshold. The default value used is 10%. This value can be set to zero, how-
> ever up to a factor of three in throughput will be lost over the performance obtained at a
> 10% threshold. Note that if the value is raised above the current usage level, users will be
> unable to allocate files until enough files have been deleted to get under the higher thres-
> hold.

special|filesys

> This is the name of the file system which will be tuned. It is either a block or character
> special file for a mountable volume or volume section, or it is the pathname of a directory
> on which a file system is mounted.

SEE ALSO

fs(5), newfs(1M), mkfs(1M)

BUGS

This program should work on mounted and active file systems. Because the super–block is not
kept in the buffer cache, the program will only take effect if it is run on dismounted file systems.
(if run on the root file system, the system must be rebooted)

You can tune a file system, but you can't tune a fish.

NAME
　　　　uconfig - system reconfiguration

SYNOPSIS
　　　　/etc/uconfig [option　boot_device]

HP–UX COMPATIBILITY
　　　　Level:　　　HP–UX/NON–STANDARD

　　　　Origin:　　HP

　　　　Remarks:　*Uconfig* is implemented on the Series 500 only.

DESCRIPTION
　　　　Uconfig enables you to reconfigure certain system parameters. When invoked with no arguments, *uconfig* lists the current system configuration. The following *options* are recognized:

　　　　-f file　　reconfigures the system parameters in the boot area according to the specifications given in *file*. *File* may contain any combination of system parameters. Each line in *file* has the following format:

　　　　　　　　id value [#comment]

　　　　where *id* is a pre-defined system parameter name, *value* is one or more values associated with the parameter, and *comment* is a descriptive comment for that line. All characters between the comment delimiter (#) and a new–line are ignored. The *id*, *value*, and *comment* fields are delimited by one or more blanks and/or tabs.

　　　　The valid *ids* and *values* are:

　　　　vm_device driver_name addr1 addr2 addr3 addr4
　　　　　　where *driver_name* is an integer specifying the virtual device driver, and *addr1* - *addr4* are integers specifying the device select code, HP–IB address, unit, and volume, respectively.

　　　　cache_buf_size size
　　　　　　where *size* is an integer in the range 256 to (maximum memory) divided by (minimum number of cache buffers), specifying the number of bytes in each individual cache buffer. *Size* is rounded down to the closest multiple of 256.

　　　　cache_buf_num num
　　　　　　where *num* is an integer in the range 1 to (maximum memory) divided by (minimum size of cache buffers), specifying the number of individual cache buffers forming the cache.

　　　　read_ahead_level level
　　　　　　where *level* is an integer in the range 1 to the value of **cache_buf_num**, specifying the number of buffers that can be filled in one sequential read operation.

　　　　interactive_time time
　　　　　　where *time* is an integer in the range 1 to 32 767 ticks (a tick equals 10 msecs), specifying the amount of CPU time a process can consume after an interactive terminal read before it is no longer favored as interactive.

　　　　swap_time time
　　　　　　where *time* is an integer in the range of 1 to 32 767 ticks (a tick equals 10msec), specifying the time a virtual segment remains memory resident before being swapped to disc.

　　　　page_size size
　　　　　　where *size* is an integer in the range 512 to 8 192, specifying the size of paged data in bytes. If *size* is an odd number, it is rounded down to the next even number.

page_swap_time time
 where *time* is an integer in the range 1 to 32 767 ticks (a tick equals 10 msecs), specifying the time a page remains memory resident before being swapped to disc.

vm_pool_size size
 where *size* is an integer in the range 16 384 to maximum memory, specifying the maximum size in bytes of the virtual memory page pool.

scroll_pages num_pages
 where *num_pages* is an integer in the range 1 to 10, specifying the number of pages of display buffering (one page = 24 lines of display). The actual number of pages allocated depends on current available memory. This parameter applies to the Model 520 only.

stack_size size
 where *size* is an integer in the range 16 384 to maximum memory, specifying the maximum stack size in bytes for any partition.

work_set_ratio ratio
 where *ratio* is a floating-point number in the range 0 to 1, specifying the minimum virtual memory working set ratio.

max_proc_per_usr max_user_process
 where *max_user_process* is an integer specifying the maximum number of processes a single user can have.

-d reconfigures the system parameters in the boot area to their default values. The default values, as contained in the file **/etc/uconfigtab**, are:

vm_device
 0 0 0 0 0; root device as determined by the system at power-up;

cache_buf_size
 1 024 bytes;

cache_buf_num
 0; this value is dynamically computed;

read_ahead_level
 0; this value is dynamically computed;

interactive_time
 300 ticks; (one tick = 10 msecs);

swap_time
 0; this value is dynamically computed;

page_size
 1 024 bytes;

page_swap_time
 50 ticks; (one tick = 10 msecs);

vm_pool_size
 0; this value is dynamically computed;

scroll_pages
 2;

stack_size
 0; this value is dynamically computed;

work_set_ratio
 0.002;

        max_proc_per_usr
           500.

The **-f** and **-d** options are mutually exclusive.

Boot_device is the path name of a character special file containing a boot area. The new configuration is written out to the boot area on *boot_device*, and takes effect the next time the system is booted.

FILES

    /etc/uconfigtab   list of default system configuration parameters

WARNING

    Do not use *uconfig* to change the system parameters of an operating system in a boot area unless that operating system is identical to the operating system you are currently running. If the two operating systems differ, *uconfig* will execute successfully, but the new operating system will either fail to boot, or, if it boots successfully, exhibit strange behavior.

NAME

      umodem - XMODEM protocol file transfer program

SYNOPSIS

      /usr/contrib/bin/umodem [-options] files

      /usr/contrib/bin/umodem -c

HP–UX COMPATIBILITY

      Level: HP–UX/NON–STANDARD

      Origin: Public Domain

DESCRIPTION

      Umodem is a file transfer program that incorporates the well known XMODEM protocol used on CP/M systems, as well as on the HP110 portable computer.

      Options:

      -1   Employ TERM II FTP 1.

      -3   Enable TERM FTP 3 (CP/M UG).

      -7   Enable 7–bit transfer mask.

      -a   Turn on ARPA Net flag.

      -l   Turn on entry logging.

      -m   Allow overwriting of files.

      -d   Don't delete *umodem.log* before starting.

      -p   Print all messages.

      -r[t b] Receive file. Specify **t** for text, or **b** for binary.

      -s[t b] Send file. Specify **t** for text, or **b** for binary.

      -y   Display file status only.

      -c   Enter command mode.

      The usual way to invoke *umodem* is:

      umodem –rt7 file

           Receive a text file.

      umodem –rb file

           Receive a binary file.

      umodem –st7 file

           Send a text file.

      umodem –sb file

           Send a binary file.

SEE ALSO

      Kermit(1M), cu(1C), uucp(1C).

NAME
uucico - uucp copy in and copy out

SYNOPSIS
/usr/lib/uucp/uucico [**-r1**] [**-s**sys] [**-x**num]

HP–UX COMPATIBILITY
Level: HP–UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral PC.

DESCRIPTION
Uucico scans the */usr/spool/uucp* directory for work files. If such files exist, a connection to a remote system is attempted using the line protocol for the remote system specified in the **L.sys** file. *Uucico* then executes all requests for work and logs the results.

The options are as follows:

-r1   Start *uucico* in the MASTER mode; The default is SLAVE mode.

-ssys   Do work only for the system specified by *sys*. If there is no work for *sys* on the local spool directory, initiate a connection to *sys* to determine if *sys* has work for the local system.

-xnum   Use debugging option. *Num* is an integer in the range 1 - 9. More debugging information is given for larger values of *num*.

Uucico is usually started by a local program (*cron*(1M), *uucp*(1C), *uuc*(1C), *uuxqt*(1C), or *uucico*(1C)). It should *only* be directly initiated by a user when debugging.

When started by a local program, *uucico* is considered the MASTER and attempts a connection to a remote system. If *uucico* is started by a remote system, it is considered to be in SLAVE mode.

For the *uucico* connection to a remote system to be successful, there must be an entry in the */etc/passwd* file on the remote system of the form:

uucp::5:5::/usr/spool/uucppublic:/usr/lib/uucp/uucico

FILES
Refer to *Uucp File System* chapter in the *HP–UX Serial Networks Guide*, part number 97076–90001.

SEE ALSO
HP-UX Asynchronous Communications Guide, part number 97076-90001.

NAME
uuclean - uucp spool directory clean-up

SYNOPSIS
/usr/lib/uucp/uuclean [options]

HP–UX COMPATIBILITY
Level: HP–UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral PC.

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. If *directory* is not a valid spool direc-tory it cannot contain "work files" i.e., files whose names start with "C.". These files have special meaning to **uuclean** pertaining to **uucp** job statistics.

-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, how-ever, 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]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral PC.

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
    uuxqt - uucp command execution

SYNOPSIS
    /usr/lib/uucp/uuxqt [**-x**num]

HP-UX COMPATIBILITY
    Level: HP-UX/STANDARD

    Origin: System V

    Remarks: Not supported on the Integral PC.

DESCRIPTION
    The *uuxqt* daemon performs local command execution of execution files (**X.***) on the
    /usr/spool/uucp directory. *Uux* generates work files with an execution (X) grade which become
    execution files when transferred to the remote system. The command requested by the execution
    file is checked against the list of remotely executable commands in the L.cmds file. The USERFILE
    is then searched to find the first NULL system field for path access permission.

    The option **-x***num* is a parameter specifying debugging information. *Num* is an integer in the
    range 1 - 9. The amount of debugging information increases as the value of *num* increases.

FILES
    Refer to the *Uucp File System* chapter in the *HP-UX Serial Networks Guide*, part number
    97076-90001.

SEE ALSO
    HP-UX Asynchronous Communications Guide.

NAME

wall - write to all users

SYNOPSIS

/etc/wall

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Native Language Support:
8-bit data.

Remarks: Not supported on the Integral Personal Computer.

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)).

Wall has timing delays, and will take at least 30 seconds to complete.

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 - which users are doing what

SYNOPSIS

      /etc/whodo

HP–UX COMPATIBILITY

      Level: HP–UX/STANDARD

      Origin: System III

      Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION

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

FILES

      /etc/passwd

SEE ALSO

      ps(**1**), who(1).

NAME
> intro - introduction to system calls

HP–UX COMPATIBILITY
> Level: This entry describes where in the HP–UX compatibility model this capability appears.
>
> Origin: The system or systems from which this facility is derived.

DESCRIPTION
> This section describes all of the system calls. All of these calls return a function result. This result indicates the status of the call. Typically, a zero or positive result indicates that the call completed successfully, and -1 indicates an error. The individual descriptions specify the details. An error number is also made available in the external variable *errno* (see *errno*(2)). *Errno is not cleared on successful calls*, so it should be tested only after an error has been indicated.
>
> The descriptions of the facilities in this section depend on the definitions of several terms. See *glossary*(9) for precise definitions.

HARDWARE DEPENDENCIES
> Series 500:
>> A second error indicator, *errinfo*, is implemented in addition to *errno*. See *errinfo*(2).

SEE ALSO
> intro(3), glossary(9).

NAME

access - determine accessibility of a file

SYNOPSIS

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

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System V

DESCRIPTION

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

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

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

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

[ENOENT]   Read, write, or execute (search) permission is requested for a null path name.

[ENOENT]   The named file does not exist.

[EACCES]   Search permission is denied on a component of the path prefix.

[EROFS]   Write access is requested for a file on a read-only file system.

[ETXTBSY]   Write access is requested for a pure procedure (shared text) file that is being executed.

[EACCES]   Permission bits of the file mode do not permit the requested access.

[EFAULT]   *Path* points outside the allocated address space for the process.

The owner of a file has permission checked with respect to the "owner" read, write, and execute mode bits. Members of the file's group other than the owner have permissions checked with respect to the "group" mode bits, and all others have permissions checked with respect to the "other" mode bits. *Access* will always report accessibility when executed by the super-user.

Access will report that a file currently open for execution is not writable, regardless of the setting of its mode.

RETURN VALUE

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

HARDWARE DEPENDENCIES

Integral PC:

The Integral PC allows normal user processes all capabilities previously reserved for the super user.

A file currently open for execution is writable.

SEE ALSO

chmod(2), stat(2).

NAME

acct - enable or disable process accounting

SYNOPSIS

int acct (path)
char *path;

HP-UX COMPATIBILITY

Level: HP-UX/EXTENDED - Multi-User

Origin: System V

DESCRIPTION

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

Path points to a path name naming the accounting file. The accounting file format is given in *acct*(5).

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

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

[EPERM]   The effective user ID of the calling process is not super-user.

[EBUSY]   An attempt is being made to enable accounting when it is already enabled.

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

[ENOENT]   One or more components of the accounting file path name do not exist.

[EACCES]   The file named by *path* is not an ordinary file.

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

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

[ETXTBSY]   *Path* points to a text file which is currently open.

RETURN VALUE

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

HARDWARE DEPENDENCIES

Series 200/500:

The system's process accounting routine will ignore any locks placed on the process accounting file.

A child process which is created by *vfork*(2) but which does not call *exec*(2) before terminating will not generate a process accounting record.

If the size of the process accounting file reaches 5000 blocks, records for processes terminating after that point will be silently lost. However, in that case the *turnacct* command would still sense that process accounting is enabled. This loss of records can be prevented by the use of *ckpacct* (see *acctsh*(1M)).

Integral Personal Computer:

Process accounting is not supported on the Integral Personal Computer.

SEE ALSO

acct(1M), acctsh(1M), exit(2), lockf(2), signal(2), vfork(2), acct(5).

NAME

 alarm - set a process's alarm clock

SYNOPSIS

 unsigned long alarm (sec)
 unsigned long sec;

HP-UX COMPATIBILITY

 Level: HP-UX/RUN ONLY

 Origin: System V

DESCRIPTION

 Alarm instructs the alarm clock of the calling process to send the signal **SIGALRM** to the calling process after the number of real time seconds specified by *sec* have elapsed; see *signal*(2). *Sec* must be less than $2\hat{\ }32$. Specific implementations may place further limitations on the maximum alarm time supported. The constant *MAX_ALARM* defined in *<sys/param.h>* specifies the implementation–specific maximum. Whenever *sec* is greater than this maximum but less than $2\hat{\ }32$, it is silently rounded down to this maximum. On all implementations, *MAX_ALARM* is guaranteed to be at least 31 days (in seconds).

 The alarm will be signaled within a 0.5 second tolerance. For example, if you specify an alarm time of 1 second, the alarm will be signaled between 0.5 seconds and 1.5 seconds later. Due to variations in scheduling, the receipt of the signal may be delayed, particularly if the process is not running at the time the signal occurs.

 Alarm requests are not stacked; successive calls reset the alarm clock of the calling process.

 If *sec* is 0, any previously made alarm request is canceled.

 Alarms are not inherited by a child process across a *fork*, but are inherited across an *exec*.

 On systems which support the *getitimer(2)* and *setitimer(2)*, the timer mechanism used by *alarm* is the same as that used by *ITIMER_REAL*. Thus successive calls to *alarm*, *getitimer*, and *setitimer* will set and return the state of a single timer.

RETURN VALUE

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

SEE ALSO

 sleep(1), pause(2), getitimer(2), signal(2), sleep(3).

NAME

brk, sbrk - change data segment space allocation

SYNOPSIS

int brk (endds)
char *endds;

char *sbrk (incr)
int incr;

HP–UX COMPATIBILITY

Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION

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

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

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

Brk and *sbrk* will fail without making any change in the allocated space if one or more of the following are true:

[ENOMEM]   Such a change would result in more space being allocated than is allowed by a system–imposed maximum (see *ulimit*(2)).

[ENOMEM]   Such a change would cause a conflict between addresses in the data segment and any attached shared memory segment (see *shmop*(2)).

HARDWARE DEPENDENCIES

Series 500:

Brk and *sbrk* will fail without making any change in the allocated space if such a change would move the program break below the beginning of the process' indirect data area. Note that it is not possible to release the direct data area with this system call.

If the process' indirect data area is paged, then the size of that data area changes in increments of the page size, which is configurable. Consequently, increasing a paged process data area by one byte may cause it to increase by one page, and decreasing it by one byte may do nothing. If the process' data area is not paged, then the size of the process data area changes similarly in increments of 32 bytes.

The pointer returned by *sbrk* is not necessarily word–aligned. Loading or storing words through this pointer could cause word alignment problems.

RETURN VALUE

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

SEE ALSO

exec(2), end(3), malloc(3), shmop(2), ulimit(2).

NAME

chdir - change working directory

SYNOPSIS

int chdir (path)
char *path;

HP–UX COMPATIBILITY

Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION

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

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

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

[ENOENT]   The named directory does not exist.

[EACCES]   Search permission is denied for any component of the path name.

[EFAULT]   *Path* points outside the allocated address space of the process.

[ENOENT]   *Path* is null.

RETURN VALUE

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

SEE ALSO

cd(1), chroot(2).

NAME
chmod, fchmod - change access mode of file

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

fchmod (fd, mode)
int fd, mode;

HP–UX COMPATIBILITY
Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION
Path points to a path name naming a file. *Fd* is a descriptor for a file. *Chmod* sets the access permission portion of the file's mode according to the bit pattern contained in *mode*.

Access permission bits are interpreted as follows:

04000	Set user ID on execution.
02000	Set group ID on execution.
02000	Set file locking mode to enforced (shared with set group ID on execution bit)
01000	Save text image after execution
00400	Read by owner
00200	Write by owner
00100	Execute (or search if a directory) by owner
00070	Read, write, execute (search) by group
00007	Read, write, execute (search) by others

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

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

If the effective user ID of the process is not super–user and the effective group ID of the process does not match the group ID of the file and none of the group IDs in the access group list match the group ID of the file, mode bit 02000 (set group ID on execution and enforced file locking mode) is cleared.

The set group ID on execution bit is also used to cause file locking mode (see *lockf*(2)) to be enforced. Files with this bit set that are not group executable will have enforcement set.

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

Chmod will fail and the file mode will be unchanged if one or more of the following are true:

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

[ENOENT]   The named file does not exist.

[EACCES]   Search permission is denied on a component of the path prefix.

[EPERM]   The effective user ID does not match the owner of the file and the effective user ID is not super–user.

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

[EFAULT]   *Path* points outside the allocated address space of the process.

[ENOENT]   *Path* is null.

HARDWARE DEPENDENCIES
Series 500:

Chmod changes the mode of files created only in the HP–UX environment (that is, not those created by the HP 9000 BASIC Language System).

Fchmod is not implemented on Series 500.

Integral PC:

The Integral PC allows normal user processes all capabilities previously reserved for the super user.

The "save text image after execution" bit is not supported.

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

SEE ALSO
chmod(1), chown(2), mknod(2), lockf(2).

NAME

chown, fchown - change owner and group of a file

SYNOPSIS

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

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

HP–UX COMPATIBILITY

Level: chown: HP–UX/RUN ONLY
            fchown: HP–UX/STANDARD

Origin: System V and UCB

DESCRIPTION

Path points to a path name naming a file. *Fd* is a descriptor for a file. The owner ID and group ID of the file are set to the numeric values contained in *owner* and *group* respectively. Note that *owner* and *group* should be less than or equal to 65535, since only the least significant 16 bits are used.

Only processes with effective user ID equal to the file owner or super–user may change the ownership of a file. If privilege groups are supported, the owner of a file may change the ownership only if he is a member of a privilege group allowing *chown*, as set up by *setprivgrp*. The default gives the *chown* privilege to all users.

The group ownership of a file can be changed to any group in the current process's access list or to the real or effective group id of the current process. If privilege groups are supported and the user is permitted the *chown* privilege, the file can be given to any group.

Chown will fail and the owner and group of the named file will remain unchanged if one or more of the following are true:

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

[ENOENT]   The named file does not exist.

[EACCES]   Search permission is denied on a component of the path prefix.

[EPERM]   EPERM is set when the effective user ID is not super–user and one or more of the following conditions exist:

               The effective user ID does not match the owner of the file.

               When changing the owner of the file, if the owner of the file is not a member of a privilege group allowing *chown*.

               When changing the group of the file, if the owner of the file is not a member of a privilege group allowing *chown* and the group number is not in the current process's access list.

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

[FAULT]   *Path* points outside the allocated address space of the process.

HARDWARE DEPENDENCIES

Series 500:

Chown changes the owner and group of files created only in the HP–UX environment (that is, not those created by the HP 9000 BASIC Langauge System).

Fchown is not implemented on Series 500.

Integral PC:

32-bit device numbers and 24-bit minor device numbers are supported.

RETURN VALUE

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

SEE ALSO

chown(1), chmod(2).

NAME

 chroot - change root directory

SYNOPSIS

 int chroot (path)
 char *path;

HP-UX COMPATIBILITY

 Level: HP-UX/RUN ONLY

 Origin: System V

DESCRIPTION

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

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

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

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

 [ENOTDIR] Any component of the path name is not a directory.

 [ENOENT] The named directory does not exist.

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

 [EFAULT] *Path* points outside the allocated address space of the process.

 [ENOENT] *Path* is null.

RETURN VALUE

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

HARDWARE DEPENDENCIES

 Integral PC:

  Super-user capabilities are provided to the normal user.

SEE ALSO

 chroot(1), chdir(2).

NAME
close - close a file descriptor

SYNOPSIS
int close (fildes)
int fildes;

HP–UX COMPATIBILITY
Level: HP UX/RUN ONLY

Origin: System V

DESCRIPTION
Fildes is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call. *Close* closes
the file descriptor indicated by *fildes*. All associated file segments which have been locked by this
process with the *lockf* function are released (i.e., unlocked).

[EBADF]   *Close* will fail if *fildes* is not a valid open file descriptor.

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

SEE ALSO
creat(2), dup(2), exec(2), fcntl(2), open(2), pipe(2), lockf(2).

NAME
  creat - create a new file or rewrite an existing one

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

HP–UX COMPATIBILITY
  Level: HP–UX/RUN ONLY

  Origin: System V

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

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

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

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

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

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

[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	A component of the path prefix does not exist.
[EACCES]	Search permission is denied on a component of the path prefix.
[EACCES]	The named file has active locks on it (placed by *lockf*(2) facililty) that are owned by other processes.
[ENOENT]	The path name is null.
[EACCES]	The file does not exist and the directory in which the file is to be created does not permit writing.
[EROFS]	The named file resides or would reside on a read–only file system.
[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed.
[EACCES]	The file exists and write permission is denied.
[EISDIR]	The named file is an existing directory.
[EMFILE]	More than the maximum number of file descriptors are currently open.
[EFAULT]	*Path* points outside the allocated address space of the process.
[ENFILE]	The system file table is full.

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

HARDWARE DEPENDENCIES

Integral PC:
   __NFILE is equal to 20.

   A file does not exist on a disc for which it is intended until a close operation is performed
   on that file.

SEE ALSO
   chmod(2), close(2), dup(2), fcntl(2), lockf(2), lseek(2), open(2), read(2), truncate(2), umask(2),
   write(2).

NAME

dup - duplicate an open file descriptor

SYNOPSIS

int dup (fildes)
int fildes;

HP-UX COMPATIBILITY

Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION

Fildes is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call. *Dup* returns a new file descriptor having the following in common with the original:

Same open file (or pipe).

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

Same access mode (read, write or read/write).

Same file status flags (see *fcntl*(2), F_DUPFD).

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

The file descriptor returned is the lowest one available.

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

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

[EMFILE]   The maximum number of file descriptors (defined by _NFILE) are currently open.

RETURN VALUE

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

HARDWARE DEPENDENCIES

Integral PC:
        _NFILE is equal to 20.

SEE ALSO

close(2), creat(2), exec(2), fcntl(2), open(2), pipe(2).

NAME

dup2 - duplicate an open file descriptor to a specific slot

SYNOPSIS

int dup2(fildes, fildes2)
int fildes, fildes2;

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD - Version 7 compatibility and UCB compatibility

Origin: Version 7

Remarks: This facility is provided for backwards compatability with Version 7 and BSD systems. *Fcntl* should be used for all new code.

DESCRIPTION

Fildes is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call. *Fildes2* is a non–negative integer less than the maximum value allowed for file descriptors. *Dup2* causes *fildes2* to refer to the same file as *fildes*. If *fildes2* already referred to an open file, it is closed first. The file descriptor returned by *dup2* has the following in common with the original:

Same open file (or pipe).

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

Same access mode (read, write or read/write).

Same file status flags (see *fcntl*(2), F_DUPFD).

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

This call can be accessed by giving either (for Version 7) the –**lV7** or (for 4.1 or 4.2bsd) the –**lBSD** option to *ld*(1).

DIAGNOSTICS

Dup2 will fail if one or more of the following are true:

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

[EINVAL]   *Fildes2* is not in the range of legal file descriptors.

RETURN VALUE

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

HARDWARE DEPENDENCIES

Series 500:

Dup2 is not currently supported on the Series 500.

Integral PC:

_NFILE is equal to 20.

SEE ALSO

close(2), creat(2), dup(2), exec(2), fcntl(2), open(2), pipe(2).

NAME

ems - Extended Memory System

SYNOPSIS

#include <sys/ems.h>

HP-UX COMPATIBILITY

Level: Extended Memory – HP–UX/EXTENDED

Origin: HP

Remarks: The Extended Memory System is not available on the Series 200.

DESCRIPTION

Extended Memory System consists of intrinsics which allocate and deallocate address space, map files into address spaces, support shared memory, and change the protection of address spaces. There are separate manual pages for the intrinsics. This page describes features in common to all the intrinsics in EMS.

Definitions

memory space

This is the actual physical memory of a machine.

address space

This refers to the logical memory of a process. Memory space is shared by having processes' address space refer to the same memory space.

segment

A contiguous piece of address space.

Properties of a Segment

During the allocation of a segment, the following types of segments can be requested:

MEM_SHARED

The address space is to be sharable with other processes. The data is shared across *fork*(2) (i.e. not copied on a fork).

MEM_PRIVATE

The address space is process local, and is copied on a *fork*(2). All memory segments will be either MEM_SHARED or MEM_PRIVATE; the default is MEM_PRIVATE.

MEM_CODE

The address space may, at some time in its lifetime, be made executable.

MEM_DATA

The address space may, at some time in its lifetime, be read and/or written. A segment may be MEM_CODE, MEM_DATA, or both. The default type is derived from the initial access permissions:

$$
\begin{array}{lcl}
\text{MEM\_R | MEM\_W} & \rightarrow & \text{MEM\_DATA} \\
\text{MEM\_X} & \rightarrow & \text{MEM\_CODE} \\
\text{(MEM\_R | MEM\_W) \&\& MEM\_X} & \rightarrow & \text{MEM\_CODE | MEM\_DATA.}
\end{array}
$$

MEM_PAGED

Requests that a segment be created as a paged object. (This is ignored if not significant for a particular implementation).

File Mapping

EMS provides the facility for mapping a file into process address space. This is done via *memallc*(2). Files can be either private or shared.

For private file mapped segments, the address space will contain an image of the file as it existed at the time of the *memallc*(2) call. Subsequent alterations of the file will have no effect on the contents of the address space, and vice–versa.

For shared file mapped segments, the address space is identically the file (at least the mapped portion thereof). Changes to the address space represent changes to the file, and vice–versa. For example, a write or read to or from the address space is, in all ways, equivalent to a file system write or read. Similarly, re–creating (using *creat*(2)) the file will result in the address space containing all zeros.

The access permissions (e.g. read, write) applied to a shared mapped file are established by the first *memallc*(2) referencing that file. Subsequent mappings of the same file by other processes must request identical access permissions.

File mapping, as described above, is only guaranteed to apply to regular local files and block structured device files. File mapping is not applicable to remote files at this time. Attempting to map an unsupported file type will result in error EINVAL.

Note that file mapping, either MEM_PRIVATE or MEM_SHARED, *always* requires read permission on *fileid*. Access modes cannot exceed those on *fileid* for shared, mapped files.

Shared Memory

Shmget(2) is the preferred intrinsic for sharing memory space between processes. Avoid using when *shmget* is available.

By using *ems*, it is also possible to share a memory space between processes. Access to shared memory can occur in two ways. The first way is to associate a file name as the name of the shared memory space. Each related or unrelated process performs a *memallc*(2) to gain access to the shared memory through mapping the file.

Another method of sharing, without the file, is for related processes: a process can allocate a non–file–mapped shared segment; upon a *fork*(2), the child process will have access to the same memory space as the parent.

SEE ALSO

memadvise(2), memallc(2), memchmd(2), memlck(2), memvary(2), vsadv(2), vson(2) malloc(3C), shmget(2), shmop(2), shmctl(2).

NAME

   errinfo - error indicator

SYNOPSIS

   extern int errinfo;

HP–UX COMPATIBILITY

   Level: HP–UX/EXTENDED

   Origin: HP

   Remarks: *Errinfo* is implemented on the Series 500 only.

DESCRIPTION

   When an error occurs in a system call, the external variable *errno* is set to the standard HP–UX error number, and more detailed information is stored in the external variable *errinfo*. *Errinfo* obtains its value from the escape code returned by the underlying HP–UX kernel.

   Errinfo is not cleared on successful system calls, so it should only be checked after an error has been indicated.

   Software that is intended to be portable across HP–UX implementations should not reference *errinfo*.

   The *errinfo* values and their meanings are as follows:

   VALUE MEANING

*4	NVM address out of range;
5	buffer request is not within valid range;
6	buffer address space overflow;
*7	address specified does not reference a valid buffer;
*10	specified process priority level out of range;
*11	a non–existent code segment is specified;
*12	attempt to delete non–existent partition;
*13	system parameter not addressable;
*14	system parameter cannot be referenced with an EDS pointer;
*20	invalid message link;
*21	invalid message link;
*22	message limit exceeded;
*23	link limit exceeded;
*24	link being deleted contains processes waiting for messages;
*30	timer canceled;
*31	timer stopped;
*32	cancel already done for specified timer ID;
*33	stop already done for specified timer ID;
*34	timer ID not stopped before cleared;
*35	timer ID not canceled before cleared;
*36	attempt to set time and date to a value outside accessible range (midnight January 1, 1900 to midnight December 31, 25599);
*37	stack extension error;
40	memory overflow (private partition);
41	memory overflow;
42	no free partition available for allocation;
43	segment table overflow;
44	memory controller block overflow;
45	partition overflow;
46	pointer passed as an argument does not point to a valid segment;

47   segment size is out of range;
*48   free space chains are inconsistent, segment map corruption;
*49   free space chains are inconsistent, block map corruption;
50   pointer passed as an argument does not point to a valid segment;
*51   block address within a segment is invalid;
56   device or interface card timed out;
57   system call aborted by *signal*(2);
*59   improper resource management in operating system;
*60   improper resource management in operating system;
*63   routine called for wrong I/O device or at wrong time;
*64   routine called for wrong I/O device or at wrong time;
*65   used in BASIC only;
66   hardware or firmware error in interface card;
*67   I/O transaction aborted by device or interface card;
68   an HP-IO interface card failed its self test;
*69   used during power-up, produces "System halted – incompatible IOP's" message;
*70   no such object;
*73   out of timer ID's;
*74   timer ID out of range;
*75   start_partition parameters not consistent;
*76   parameter to start_partition not addressable;
*77   attempt to change to non-existent partition;
*78   must be a system process to change to partition;
80   device not ready for request, may be busy with some other operation, or power
        may be off;
81   media is write-protected and cannot be altered;
82   media has been mis-inserted;
83   format switch disables driver from doing a media format operation;
84   media error was detected, usually a CRC, parity, or checksum error; data may
        not be valid;
85   cannot find record on media; usually indicates trouble in reading the
        header/servo information on the media;
86   the read check of data written to a record has failed;
88   media may have been changed since last access; buffered data may have to be
        thrown away;
*89   used to implement internally generated re-tries;
*90   software failure was detected; perhaps data structures were corrupted, or an
        unexpected event occurred;
91   unknown error; indicates some type of device or interconnect malfunction;
*94   media_active (true) request must be made before first access;
95   a parameter for a particular request is not supported by this driver; usually indi-
        cates that the type of card does not support a special function;
97   termination mode is not supported by this device driver;
98   EOI must have a data byte associated with it before it can be written;
*99   driver must be opened for request;
100   record number out of allowed range; usually indicates corrupt directory struc-
        ture;
*101   the transfer length was negative, zero, or odd for a halfword read or write
        request; can also indicate a transfer past the end of the media volume;
102   halfword or byte mode transfers are not supported by this driver;
*103   cannot close a locked driver; this is a fault of the calling code;
105   the argument specified for this ioctl request is out of range or points to the
        wrong type of structure;

106   The ioctl command given is not recognized by this device;

107   an attempt was made to attach two different drivers to the same device; these
        drivers are incompatible and cannot co-exist; the new driver is not attached, but
        the old driver remains unchanged;

108   the size of the string is not correct for this string register access;

109   interleave factor not supported by disc; it is either zero, negative, or too big;

110   invalid address was detected by the driver, or the interface card occupies the
        same subaddress as the device;

*111   capacity of disc exceeds 32-bit record address range assumed by driver;

112   reference to an unsupported pseudo-register was made; if the request accessed
        multiple registers, the previous (if valid) register accesses were made;

113   HP-IB TCT byte must be at the end of the ATN sequence because you have
        passed control;

114   a request is not supported by this driver;

115   no driver with that name was found;

116   no driver is available for that card, or the device address value is out of range;

117   write verify is not supported for this mass storage device;

118   length of -1 specifying that a transfer should be used is invalid;

119   an invalid value was assigned to a pseudo-register;

120   data transfer was terminated due to the reception of a secondary address;

121   for buffered devices, a data transfer cannot be satisfied due to un-transferred
        data from the other direction; for example, a write may not be possible if there is
        still unread data present on the device;

122   device cannot satisfy this request because of a previous request or the current
        state of the device;

123   the beginning of the tape was encountered before the operation could be com-
        pleted;

124   the interface cannot be the HP-IB active controller when doing this operation;

125   synchronous data rate could not be met to complete the operation; system may
        be too heavily loaded, or the specified bandwidth parameters for this or another
        device may be wrong;

126   a hardware fault was detected; controller/status card should be examined for
        further information;

127   the device/interface was not found at the specified address; power may be off, or
        the address could be wrong;

128   the end of tape was encountered before the operation was complete;

129   the device failed its self test or a diagnostic; no further access to this device
        should be attempted;

130   the HP-IB interface is too slow for this synchronous device;

131   tape end of file was encountered before request could be completed;

132   the device was busy and could not handle the request;

133   the media is absent from the device;

134   the media is not formatted, and must be formatted before use;

135   too many media errors prevent formatting to complete; formatting operation
        may be only partially done;

136   the media has no more spares left but had to spare some data; the sparing was
        not done;

137   the HP-IB interface must be the active controller to execute this operation;

138   the HP-IB interface must be the system controller to execute this operation;

139   no data seen on media after a device specific length of media; this is a sequential
        tape error;

140   more data was found in the record than was requested for the read operation;
        the remaining data was lost, and cannot be read by the next read request;

141   the media physical format is incorrect for this disc;

142   media failure has occurred, or the media has deteriorated such that replacement is suggested; writing is no longer allowed; media may only last long enough for a back–up;

143   the HP–IB interface is not addressed to read or write as requested, and because it is active controller, it cannot become addressed;

144   the read or write request data transfer was aborted by an HP–IB IFC or an HP–IB device clear operation;

145   not all the data (or commands) were accepted by the device;

146   not all the data was sourced by the device;

147   controller or unit fault was reported by the device;

148   some failure occurred in receiving the device status result; usually means that not all the status was returned, or the controller reported a failure when the driver attempted to receive the status;

149   the operation cannot be completed because a user programmed hold off has occurred;

*150   system problem or failure;

*151   successful completion of task; should not be visible;

157   the volume label specified in the volume specifier does not match the volume label on the volume;

158   links may not be removed if the file has been opened with the "no purge link" option;

160   cannot open a directory with write access;

161   two or more volumes have the same volume label and the file system is unable to distinguish between them for this request;

162   an attempt was made to access an open file in a way forbidden by the file system;

163   the disc format does not support the requested operation;

164   the file cannot be opened for writing because it is currently being *exec*ed, or the file may not be opened with execute access because it is currently opened for writing;

165   the file/device could not be opened because the system open file table is full; this is caused by a memory overflooverflow

166   a file may not be opened in both "shared" and "exclusive" modes; your access mode conflicts with the current mode;

167   a signal was received while waiting to read or write to a pipe;

168   the request cannot be performed because the designated file is open or in use at the current time;

169   an attempt was made to purge a link to the file without obtaining the necessary access rights;

170   not enough disc space could be allocated to satisfy the request;

171   a file with the same name already exists in the directory;

172   the file ID passed to the system was bad;

173   an attempt was made to read beyond the physical end of the file;

174   tried to write to a pipe for which there are no readers;

*175   the request made is not supported by the file system;

176   same as error 162, except that the file may not be open;

177   a "position" (*lseek*) request was made on a pipe;

178   the device driver specified in the volume specifier does not match the current device driver being used for the volume;

179   the disc format specified in the volume specifier does not match the disc format on the volume;

181   some file in the file path could not be found;

182   the device specified is not a random access blocked device;

183   the disc format on the disc does not support volume labels;

184   the disc format on the disc does not support file passwords;

185   the disc does not contain a recognizable disc format; the disc format name given for an initialize request is not known to the system;

188   the region of the file that was accessed is currently locked;

189   a volume may not be initialized while there are open files on it;

193   a non–directory was specified where a directory was required;

198   the request cannot be satisfied because another file cannot be added to the directory; no i–nodes were available;

201   the request cannot be satisfied because the directory is not empty;

204   the file system was unable to extend a "contiguous" file without creating another extent;

*210   invalid file code;

216   the select code in the device address in the volume specifier is not within the acceptable range for this hardware configuration;

*217   an attempt was made to remove or change a password which does not exist;

*218   an attempt was made to put two identical passwords on a file with different capability sets;

*219   a simple deadlock was encountered when locking a file;

221   the file name is too long (LIF discs support 10 characters, HP 9845 format discs support 6 characters, and SDF discs support 16 characters);

222   invalid character in LIF or HP 9845 format disc file name;

*223   invalid character in LIF or HP 9845 format disc password;

*224   volume label is too long on a LIF or HP 9845 format disc;

*225   password too long on a LIF or HP 9845 format disc;

*226   invalid character in volume label on a LIF or HP 9845 format disc;

*227   invalid date on LIF or HP 9845 format disc;

*228   invalid record size on LIF or HP 9845 format disc;

229   invalid record mode on LIF or HP 9845 format disc;

230   a file name was expected and none was specified, or an attempt was made to purge the "." or ".." links from a directory;

231   a subdirectory was specified when the disc format does not support subdirectories;

232   links not supported on LIF or HP 9845 format discs;

233   non–UNIX systems are not allowed to establish duplicate links to a directory;

234   the device (file) specified for the *mount/umount* request is not a block special device;

235   the device (file) specified for the *umount* request is not currently mounted;

236   a volume could not be unmounted because it is currently being used (there are open files or working directories established on the mounted volume); a volume could not be mounted because it is already mounted; the directory being mounted on is open or is the root directory;

237   an attempt was made to establish a link from one volume to another;

238   raw discs must be *lseek*ed and read/write sizes must be multiples of the device's physical sector size (256 bytes for discs, 1024 bytes for cartridge tapes).

241   the byte address on a file access was outside the acceptable range for the file; the byte address must be non–negative;

242   the file system saw a directory, i–node, or bit map record which contains inconsistent data;

244   an attempt was made to read beyond the logical end of the file;

249   an attempt was made to unlock an unlocked file;

*252   time value out of range;

*253	hours, minutes, or seconds value out of range;
*254	day, month, or year value out of range;
*255	invalid date;
256	specified segment does not exist;
257	page table has not been initialized;
258	page has not been initialized;
259	lock count has overflowed;
260	lock count has underflowed;
261	entire working set cannot be locked;
262	lock length is invalid;
263	segment is not locked;
264	locked segment cannot be extended; ·
265	page is not locked;
266	segment is not paged;
267	segment is not shared;
268	requested segment lengths are inconsistent;
269	minimum working set request cannot be satisfied;
270	frame pool cannot be expanded;
271	virtual memory device table overflow;
272	virtual memory device index is invalid;
273	default virtual memory device cannot be removed;
274	virtual memory device index is inactive;
275	virtual memory device index is in use;
276	a locked page was encountered;
301	escape through user code for *exec*;
302	target process not found in *kill* call;
303	target process has the wrong real user ID in *kill* call;
304	no processes found in a broadcast signal attempt;
305	signal number out of range;
306	not super–user; requires super–user permission;
307	a bad argument was supplied to a system call;
308	an attempt was made to wait with no children;
309	an intrinsic was aborted by a signal;
310	process stack overflow;
311	unrecognized *ulimit* command;
312	your DB relative argument had an offset greater than 512 Kbytes;
313	fix–up offset exceeds segment size (see *a.out*(5));
314	stack pointer passed to *brk*;
315	invalid segment number in user pointer;
316	an attempt was made to *kill*(0,sig) with no current process group;
317	file number out of range;
318	specified file ID not open;
319	*ioctl* call not implemented;
320	inappropriate *ioctl* command for device;
321	ID not in the range 0 to 65535;
323	floating point divide–by–zero;
324	floating point overflow;
325	floating point underflow;
327	wrong number of system call parameters;
328	inconsistent executable file;
329	front panel timeout (series 500, models 30 and 40 only);
330	graphics to internal CRT timed out;
331	graphics hardware does not respond;

*332   unexpected error when performing an open;
*333   unexpected error when performing a close;
 334   illegal mode of driver was requested;
 335   a buffer was passed to an intrinsic that is too large;
 336   DMA terminated abnormally;
 337   received one more x coordinate than y coordinate;
 343   user program called missing kernel segment;
 345   attempt to execute a file which is too small;
 346   attempt to execute a file with a bad magic number;
 347   unimplemented configure function;
 348   maximum stack exceeded;
 349   fatal stack overflow;
 350   the requested heap size is too big;
 358   there is no tty device at this address;
 359   this request is not supported by this device;
 360   semid, msqid or shmid is not a valid IPC identifier;
 361   semnum in semctl(2) or mtype in msgsnd(2) out of range;
 362   invalid cmd to semctl(2), msgctl(2), or shmctl(2);
 363   nsems out of range in semget(2);
 364   ID for key exists but nsems or size inconsistent with existing ID;
 365   mtext is greater than msgsz and msg_noerror is false in msgrcv(2);
 366   IPC key exists but operation permission denied;
 367   IPC operation permission denied;
 368   operation requires caller to be super-user or owner or creator of specified IPC ID;
 369   ID does not exist and IPC_CREATE not specified;
 370   system-imposed limit on number of IDs exceeded; ID not created;
 371   ID exists for key, but IPC_CREATE and IPC_EXCL both specified;
 372   nsops is greater than the system-imposed maximum;
 373   sem_num is less than zero or greater than or equal to the number of semaphores in the set associated with semid;
 374   operation would result in suspension of the calling process but IPC_NOWAIT specified;
 375   operation would cause semval or semadj value overflow;
 376   specified semaphore or message queue ID has been removed from the system;
 377   insufficient memory for IPC structure;
 378   message queue does not contain message of desired type and IPC_NOWAIT specified;
 379   shared memory size or message size (msgsz) out of range;
 380   shmaddr is invalid (non-zero);
 381   number of shared memory segments per user exceeded;
 382   shmflg is invalid (SHM_RDONLY set);
 383   no line discipline of the requested value was found;
 384   the ioctl command given is not recognized by this device;
 385   the argument specified for this ioctl request is out of range or points to the wrong type of structure;
 386   an attempt was made to enable process accounting when it was already enabled.
 387   the file specified for process accounting is not an ordinary file;
 388   lockf deadlock detected;
 389   lockf no more free locks;
 390   plock permission invalid (not superuser);
 391   PROCLOCK is invalid (PROCLOCK, TXTLOCK, or DATLOCK exists);
 392   TXTLOCK is invalid (PROCLOCK or TXTLOCK exists);

393   DATLOCK is invalid (PROCLOCK or DATLOCK exists);
394   UNLOCK is invalid (no lock exists);
395   op is invalid (not PROCLOCK, TXTLOCK, DATLOCK, or UNLOCK);
396   *plock* invalid in [vfork,exec] window;
397   *get/setitimer* invalid in [vfork,exec] window;
398   timer specification is invalid;
399   timeval is invalid;
400   no interrupt packet for this file descriptor;
401   illegal mode mask used in hpib_io function call;
*440   internal error;
441   protection modes do not match with existing segment;
442   device is not a 'CS80' device;
443   attempt to add a device not specified with a device file;
444   attempt to pass an EMS intrinsic a parameter which is out of range;
445   attempt to *memchmd* segment codes which are shared by more than one process;
446   attempt to filemap a file which has already been filemapped by process;
447   insufficient memory available to complete *memallc* request;
448   the specified memory address is invalid;
449   attempt to use EMS intrinsic on memory not allocated by *memallc*;
450   super-user capability is required to create this kind of file;
451   specified file or directory does not exist;
452   an invalid RPM program descriptor was used;
453   an RPM child process was interrupted;
455   attempt to close file failed;
456   abortive file close occurred; data may have been lost;
457   attempt at an abortive file close failed;
458   incorrect select code; device or address does not exist;
459   too much data was given for an RPM request;
460   a string is too long;
461   a name used for RPM is too long;
462   an invalid file ID was used;
463   an open file could not be found;
464   attempt to create a process has failed;
465   connection limit set by the super-user was reached;
466   login not allowed;
467   RPM was not allowed to create a remote process;
470-483
        not enough memory could be found; check the network memory limit set with
        npowerup;
490   TCP security mismatch;
491   remote login failed;
493   an RPM login is invalid;
494   consumer login sequence is invalid;
496   login sequence is invalid;
497   connection attempt was not accepted by the remote system;
498   new inbound path rejected, possibly due to lack of local resources;
500   RPM cannot set up the login environment;
501   RPM service is denied;
502   service instance is denied;
503   login on the producer system is invalid;
505   illegal socket name length was used for IPC;
506   illegal node name length was used for IPC;
507   too many file name sets were given for RFA;

508   too many node names were given in an RFA path specifier;
510   attempt was made to copy a directory;
511   parameter contained an illegal value;
513–516
    register number or value is unacceptable;
517   internal error; contact qualified HP support personnel;
518   incorrect file type; cannot create RFA remote file;
519   flag specified for RPM is invalid;
520   an option specified for RPM is invalid;
521   unacceptable format for an RPM option;
522   address given could not be used;
*523–524
    internal error; contact qualified HP support personnel;
525   illegal characters in an IPC name;
526   incorrect IPC socket descriptor used;
529   illegal IPC flag value was used;
530   illegal IPC data length was used;
532   illegal IPC control request was used;
533   illegal IPC option structure was used;
535   illegal IPC request value was used;
536   illegal IPC timeout value was specified;
537   IPC receive size too big;
540   IPC send size too big;
541   data unit is too large;
543   IPC socket specified is not a virtual circuit socket;
544   illegal address format;
545   nested remote path names are not allowed;
546   IPC socket specified is not a destination socket;
547   IPC socket specified is not a source socket;
548   error in field endpoint;
559   no local IPC socket descriptors are available;
*560–685
    internal error; contact qualified HP support personnel;
690   network is already up;
691–692
    network is down;
*694   internal error; contact qualified HP support personnel;
695   network is going down;
700   incorrectly formatted network directory was specified;
701   2285A LAN Unit download file is bad;
705   a LAN Interface hardware problem has been detected;
706   LAN Interface failed its selftest;
707   LAN Interface failed during a transmit attempt;
708   LAN Interface failed during a receive attempt;
709–710
    2285A LAN Unit failed during a download;
711   HP–IB Interface failed;
720–722
    network transport timeout occurred;
723   remote system did not respond to retransmission attempts;
724–725
    no activity on a connection; the connection has been aborted;
726   attempt to establish a connection has failed;

730–732
    remote system has violated network protocol;
733   a message is too long;
734   request was made that is unacceptable to the transport or to a remote service;
735   unrecognized RFA request;
736   request is unserviceable at this time;
737   unrecognized RFA request;
738   invalid response from the remote system;
739   remote RPM process has violated network protocol;
740   remote RPM process has reported an unrecognized error;
*741   an unrecoverable network protocol error has occurred;
745   requested service cannot be supplied;
747   system cannot support an interchange operation;
748   system cannot support a restart operation;
749   checkpointing not supported;
750   system cannot support a transient operation;
751   unknown system type;
752   buffer too small;
753   invalid remote file request;
754   an error response was received;
755   RPM does not support the requested feature;
756   remote node's version of IPC is incompatible;
*757   internal error; contact qualified HP support personnel;
761   incorrect or unknown path name;
762–763
    destination is unreachable;
764   file specified is not a network special file;
*765   internal error; contact qualified HP support personnel;
*767   internal error; contact qualified HP support personnel;
768   system name used is unknown to the local node;
770–774
    connection has been lost;
777   IPC connection request failed;
778   connection to producer is down;
780   name specified for the producer system could not be found;
782   name specified for the consumer system could not be found;
784   insufficient resources on the producer system;
785   insufficient resources on the consumer system;
786–787
    not enough memory could be obtained on the remote system. The remote system could be out of physical memory or the network memory limit on the remote node could be too small;
788   IPC socket already exists;
790   IPC socket name could not be found;
792   IPC virtual circuit connection was killed;
794   IPC virtual circuit socket cannot be named;
796   IPC connection is pending;
798   IPC process does not own the socket;
800   IPC operation would block;
804   the program for RPM is invalid;
806   the program for RPM could not be loaded;
808   LAN Interface failed. If resetting the Interface does not eliminate the problem, contact qualified HP personnel.

All *errinfo* values marked with an asterisk (*) indicate a serious system problem which should be checked by qualified HP support personnel.

For errinfo values 360 382, IPC refers to the interprocess communications facilities provided by message queues, shared memory, and semaphores. For errinfo values 450–999, IPC refers to the interprocess communications facilities provided by local area networking.

SEE ALSO

err(1), errnet(2), errno(2), perror(3C).

WARNING

Errinfo is intended for diagnostic purposes only. Values and meanings may change in future releases of HP–UX.

Errno.h

NAME
errno - error indicator for system calls

SYNOPSIS
#include <errno.h>
extern int errno;

HP–UX COMPATIBILITY
Level: HP–UX/RUN ONLY

Origin: System V, HP

DESCRIPTION
Errno is an external variable whose value is set whenever an error occurs in a system call. This value can be used to obtain a more detailed description of the error. An error condition is indicated by an otherwise impossible returned value. This is almost always -1; the individual descriptions specify the details. *Errno* is not cleared on successful system calls, so its value should be checked only when an error has been indicated.

Each system call description attempts to list all possible error numbers. The following is a complete list of the error names. The numeric values can be found in **<sys/errno.h>** but should not normally be used.

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

ENOENT No such file or directory
> This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist. It also occurs with *msgget, semget, shmget*
> when *key* does not refer to any object and the *IPC_CREAT* flag is not set.

ESRCH No such process
> No process can be found corresponding to that specified by *pid* in *kill, ptrace,* or *rtprio,* or the process is not accessible.

EINTR Interrupted system call
> An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition unless the system call is restarted (see *sigvector*(2)).

EIO I/O error
> Some physical I/O error. This error may in some cases occur on a call following the one to which it actually applies.

ENXIO No such device or address
> I/O on a special file refers to a subdevice which does not exist, or is beyond the limits of the device. It may also occur when, for example, a tape drive is not on–line or no disk pack is loaded on a drive, or when a read or write is attempted beyond the physical limit of a device.

E2BIG Arg list too long
> An argument and or environment list longer than maximum supported size is presented to a member of the *exec* family. Other possibilities include: message size or number of semaphores exceeds system limit (*msgop, semop*), or too many privileged groups have been set up (*setprivgrp*).

ENOEXEC Exec format error
> A request is made to execute a file which, although it has the appropriate permissions, does

not start with a valid magic number (see *a.out*(5)), or the file is too small to have a valid executable file header.

EBADF Bad file number

Either a file descriptor refers to no open file, a read (respectively write) request is made to a file which is open only for writing (respectively reading), or the file descriptor is not in the legal range of file descriptors.

ECHILD No child processes

A *wait* was executed by a process that had no existing or unwaited-for child processes.

EAGAIN No more processes

A *fork* failed because the system's process table is full, the user is not allowed to create any more processes, or a *semop* or *msgop* call would have to block.

ENOMEM Not enough space

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

EACCES Permission denied

An attempt was made to access a file or IPC object in a way forbidden by the protection system.

EFAULT Bad address

The system encountered a hardware fault in attempting to use an argument of a system call; can also result from passing the wrong number of parameters to a system call.

ENOTBLK Block device required

A non-block file was mentioned where a block device was required, e.g., in *mount*.

EBUSY Device or resource busy

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

EEXIST File exists

An existing file was mentioned in an inappropriate context, e.g., *link*.

EXDEV Cross-device link

A link to a file on another device was attempted.

ENODEV No such device

An attempt was made to apply an inappropriate system call to a device; e.g., read a write-only device.

ENOTDIR Not a directory

A non-directory was specified where a directory is required, for example in a path prefix or as an argument to *chdir*(2).

EISDIR Is a directory

An attempt to open a directory for writing.

EINVAL Invalid argument

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

ENFILE File table overflow

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

EMFILE Too many open files

No process may have more than a system defined number of file descriptors open at a time. For systems below HP-UX STANDARD the minimum number is 20. For systems at or above HP-UX STANDARD the minimum number is 60.

ENOTTY Not a typewriter

The (*ioctl*(2)) command is inappropriate to the selected device type.

ETXTBSY Text file busy

An attempt to execute an executable file which is currently open for writing (or reading). Also, an attempt to open for writing an otherwise writable file which is currently open for execution.

EFBIG File too large

The size of a file exceeded the maximum file size allowed by the file system, ULIMIT; was exceeded (see *ulimit*(2)), or bad semaphore number in *semop*(2) call.

ENOSPC No space left on device

During a *write* to an ordinary file, there is no free space left on the device; or, no space in system table during *msgget*(2), *semget*(2), *shmget*(2), or *semop*(2) while **SEM_UNDO** flag is set.

ESPIPE Illegal seek

An *lseek* was issued to a pipe.

EROFS Read only file system

An attempt to modify a file or directory was made on a device mounted read–only.

EMLINK Too many links

An attempt to make more than the maximum number of links (1000) to a file.

EPIPE Broken pipe

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

EDOM Math argument

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

ERANGE Result too large

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

ENOMSG No message of desired type

An attempt was made to receive a message of a type that does not exist on the specified message queue; see *msgop*(2).

EIDRM Identifier Removed

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

ENAMETOOLONG File name too long

A component of a path name exceeded the maximum number of characters for a file name, or an entire path name exceeded 1023 characters. Not all file systems always detect this error.

ENOTEMPTY Directory not empty

An attempt was made to remove a non–empty directory.

EDEADLK Resource deadlock would occur

A process which has locked a system resource would have been put to sleep while

attempting to access another process' locked resource.

ENET Local area network error

An error occured in the software or hardware associated with your local area network.

HARDWARE DEPENDENCIES

Series 500:

In the definition of ENOMEM, the maximum space size is not a system parameter. Also, the terms "text, data, and stack segments", "segmentation registers", and "swap space" are invalid.

In the definition of EMLINK, the maximum number of links is 32767.

One additional *errno* values is implemented:

EUNEXPECT Unexpected error

An unexpected error was returned from the system, indicating some type of system problem. This error should never occur; if it does, it indicates a system bug.

A second error indicator, *errinfo*, is implemented in addition to *errno*. See *errinfo*(2).

SEE ALSO

On the Series 500: err(1), errinfo(2).

NAME
exterior execl, execv, execle, execve, execlp, execvp - execute a file

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

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

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

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

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

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

HP-UX COMPATABILITY
Level: HP-UX/RUN ONLY

Origin: System V

DESCRIPTION
Exec, in all its forms, loads a program from an ordinary, executable file onto the current process, replacing the current program. This file is either an executable object file, or a file of data for an interpreter, called a script file.

An executable object file consists of a header (see *a.out*(5)), a text segment, and a data segment. The data segment contains an initialized portion and an uninitialized portion (bss). For *execlp* and *execvp* the shell (*/bin/sh*) may be loaded to interpret a script instead. There can be no return from a successful *exec* because the calling program is overlaid by the new program.

When a C program is executed, it is called as follows:

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

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. As indicated, *argc* is conventionally at least one and the first member of the array points to a string containing the name of the file. (The exit conditions from *main* are discussed in *exit*(2)).

Path points to a path name that identifies the executable file containing the new program.

File (in *execlp* or *execvp*) points to a file name identifying the executable file containing the new program. The path prefix for this file is obtained by a search of the directories passed as the *environment* line "PATH =" (see *environ*(7)). The environment is supplied by the shell (see *sh*(1)). If *file* does not have an executable magic number (*magic*(5)), then it is passed to */bin/sh* under the assumption that *file* is a shell script.

Arg0, arg1, ..., argn are pointers to null-terminated character strings. These strings constitute the argument list available to the new program. By convention, at least *arg0* must be present and point to a string that is the same as *path* (or its last component).

Argv is an array of character pointers to null-terminated strings. These strings constitute the argument list available to the new program. By convention, *argv* must have at least one member, and it must point to a string that is the same as *path* (or its last component). *Argv* is terminated by a null pointer.

Envp is an array of character pointers to null–terminated strings. These strings constitute the environment in which the new program will run. *Envp* is terminated by a null pointer. For *execl* and *execv*, the C run–time start–off routine places a pointer to the environment of the calling program in the global cell:

extern char **environ;

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

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

Signals set to terminate the calling program will be set to terminate the new program. Signals set to be ignored by the calling program will be set to be ignored by the new program. Signals set to be caught by the calling program will be set to their default value in the new program; see *signal*(2).

If the set–user–ID mode bit of the executable file pointed to by *path* or *file* is set (see *chmod*(2)), *exec* sets the effective user ID of the new program to the owner ID of the executable file. Similarly, if the set–group–ID mode bit of the executable file is set, the effective group ID of the new program is set to the group ID of the executable file. The real user ID and real group ID of the new program remain the same as those of the calling program. Note that the set–user(group)–id function does not apply to scripts, and thus if *execlp* or *execvp* executes a script, even if it has the set–user(group)–id bits set, they will be ignored.

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

Profiling is disabled for the new program; see *profil*(2).

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

> nice value (see *nice*(2))
> process ID
> parent process ID
> process group ID
> real–time priority (see *rtprio*(2))
> interval timers (see *getitimer*(2))
> semadj values (see *semop*(2))
> tty group ID (see *exit*(2) and *signal*(2))
> trace flag (see *ptrace*(2) request 0)
> time left until an alarm clock signal (see *alarm*(2))
> current working directory
> root directory
> file mode creation mask (see *umask*(2))
> file size limit (see *ulimit*(2))
> *utime*, *stime*, *cutime*, and *cstime* (see *times*(2))

A script file begins with a line of the form "#! interpreter" or "#! interpreter argument", where #! must be the first two bytes of the file. The interpreter name begins with the first character other than space or tab following the #!. When such a file is *exec*'d, the system *exec*'s the specified interpreter, as an executable object file, in its place. Even in the case of *execlp* or *execvp*, no path searching is done on the interpreter name.

The argument is anything after any tabs or spaces following the interpreter name on the #! line, including any imbedded tabs or spaces. If there is an argument, it is passed to the interpreter as *argv*[1] and the name of the script file is passed as *argv*[2]. Otherwise, the name of the script file is passed as *argv*[1]. *argv*[0] is passed as specified in the exec call. All other arguments specified in the exec call are passed following the name of the script file (that is, beginning at *argv*[3] if there is an argument; otherwise at *argv*[2]).

If the #! line exceeds some system defined maximum number of characters, an error will be posted and *exec* will not succeed; the line is terminated by either a new line or null character. The minimum value for this limit is 32.

Set–user id and set–group–id bits are honored for the script and not for the interpreter.

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

[ENOENT]	One or more components of the executable file's path name or the interpreter's path name do not exist.
[ENOTDIR]	A component of the executable file's path prefix or the interpreter's path prefix is not a directory.
[EACCES]	Search permission is denied for a directory listed in the executable file's or the interpreter's path prefix.
[EACCES]	The executable file or the interpreter is not an ordinary file.
[EACCES]	The file pointed to by *path* or *file* is not executable. The super–user cannot *exec* a file unless at least one of the three execute bits is set in the file's mode.
[EACCESS]	Read permission is denied for the executable file or the interpreter, and the process's trace flag (see *ptrace*(2) request 0) is set.
[ENOEXEC]	The exec is not an *execlp* or *execvp*, and the executable file has the appropriate access permission but there is neither a valid magic number nor a #! in its header.
[ETXTBSY]	The executable file is currently open for writing. Note: normal executable files are only open for a short time when they start execution. Other executable file types may be kept open for a long time, or indefinitely under some circumstances.
[ENOMEM]	The new program requires more memory than is available, or than is allowed by the system–imposed maximum MAXMEM.
[E2BIG]	The number of bytes in the new program's argument list is greater than the system–imposed limit. This limit will be at least 5120 bytes on HP UX systems.
[EFAULT]	The executable file is not as long as indicated by the size values in its header, or is otherwise inconsistent.
[EFAULT]	*Path, argv,* or *envp* point to an illegal address.
[ENOENT]	*Path* is null.
[ENOEXEC]	The number of bytes in the #! line of a script file exceeds the system's maximum.

HARDWARE DEPENDENCIES

Series 500:

References to memory, such as "text segment", "data segment", "initialized portion", "uninitialized portion", and "bss", are invalid. See *a.out*(5) for the Series 500.

Script files are not supported on Series 500.

Integral PC:

The super–user capabilities are provided to the normal user.

RETURN VALUE

If *exec* returns to the calling program, an error has occurred; the return value will be -1 and *errno* will be set to indicate the error.

SEE ALSO

sh(1), alarm(2), exit(2), fork(2), nice(2), ptrace(2), semop(2), signal(2), times(2), ulimit(2),

umask(2), a.out(4), environ(5).

NAME

exit, _exit - terminate process

SYNOPSIS

void exit (status)
int status;

void _exit (status)
int status;

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System V

DESCRIPTION

Exit terminates the calling process and passes *exit*'s argument to the system for inspection; see *wait*. Returning from *main* in a C program has the same effect as *exit*; the *exit* value is the function value RETURNed by *main*. (This value will be undefined if *main* does not take care to return a value or explicitly call *exit*.)

Exit is equivalent to *_exit*, except that *exit* flushes stdio buffers, while *_exit* does not. Both *exit* and *_exit terminate the calling process with* the following consequences:

All open file descriptors found during the calling process are closed.

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

If the parent process of the calling process is not executing a *wait* and does not have **SIGCLD** set to **SIG.IGN**, the calling process is transformed into a zombie process. A *zombie process* is a process that only occupies a slot in the process table. It has no other space allocated either in user or kernel space. Time accounting information is recorded for use by *times*(2).

The parent process ID of all of the calling process's existing child processes and zombie processes is set to 1. This means the initialization process (proc1, see *glossary*(9)) inherits each of these processes.

Each attached shared memory segment is detached and the value of **shm_nattach** in the data structure associated with its shared memory identifier is decremented by 1 (see *shmop*(2)).

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

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

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

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

SEE ALSO

Exit conditions (**$?**) in sh(1), acct(2), plock(2), semop(2), shmop(2), signal(2), times(2), vfork(2), wait(2).

NAME
　　fcntl - file control

SYNOPSIS
　　#include <fcntl.h>

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

HP–UX COMPATIBILITY
　　Level:　　　Basic calls: HP–UX/RUN ONLY

　　　　　　　Real time extensions: HP–UX/STANDARD - Real Time

　　Origin:　　System V, UCB, and HP

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

　　The *cmd*s available are:

　　F_DUPFD　　　Return a new file descriptor that has the following characteristics:

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

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

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

　　　　　　　　Same access mode (read, write or read/write).

　　　　　　　　Same file status flags (i.e., both file descriptors share the same file status flags).

　　　　　　　　The close–on–exec flag associated with the new file descriptor is set to remain open across *exec*(2) system calls.

　　F_GETFD　　　Get the close-on-exec flag associated with the file descriptor *fildes*. If the low–order bit is **0** the file will remain open across *exec*(2), otherwise the file will be closed upon execution of *exec*(2).

　　F_SETFD　　　Set the close-on-exec flag associated with *fildes* to the low–order bit of *arg* (see F_GETFD).

　　F_GETFL　　　Get *file* status flags; see *fcntl*(7).

　　F_SETFL　　　Set *file* status flags to *arg*. Only certain flags can be set; see *fcntl*(7).

ERRORS
　　Fcntl fails if one or more of the following conditions are true. *errno* is set accordingly:

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

　　[EMFILE]　　　*Cmd* is F_DUPFD and the maximum number of file descriptors is currently open.

　　[EINVAL]　　　*Cmd* is F_DUPFD and *arg* is negative or greater than the maximum number of file descriptors.

RETURN VALUE
　　Upon successful completion, the value returned depends on *cmd* as follows:
　　　　　F_DUPFD　　A new file descriptor.
　　　　　F_GETFD　　Value of close–on–exec flag (only the low–order bit is defined).
　　　　　F_SETFD　　Value other than -1.
　　　　　F_GETFL　　Value of file status flags.
　　　　　F_SETFL　　Value other than -1.
　　Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

SEE ALSO
    close(2), exec(2), open(2), fcntl(7).

NAME

fork - create a new process

SYNOPSIS

int fork ( )

HP-UX COMPATIBILITY

Level: HP -UX/RUN ONLY

Origin: System V, HP

DESCRIPTION

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

environment
close-on exec flag (see *exec*(2))
signal handling settings (i.e., **SIG_DFL**, **SIG_IGN**, function address)
set-user-ID mode bit
set-group-ID mode bit
profiling on/off status (see *profil*(2))
real-time priority (see *rtprio*(2))
nice value (see *nice*(2))
all attached shared memory segments (see *shmop*(2))
process group ID
tty group ID (see *exit*(2) and *signal*(2))
trace flag (see *ptrace*(2) request 0)
current working directory
root directory
file mode creation mask (see *umask*(2))
file size limit (see *ulimit*(2))

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

The child process has a unique process ID.

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

The child process has its own copy of the parent's file descriptors. Each of the child's file descriptors shares a common file pointer with the corresponding file descriptor of the parent.

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

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

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

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

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

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

The parent and child processes resume execution immediately after the *fork* call; they are identified by the value returned by *fork* (see below).

Note that standard I/O buffers are duplicated in the child. Thus, if you fork after a buffered I/O operation that was not flushed, you may get duplicate output.

Vfork is provided as a higher performance, limited version of *fork* on some systems. See *vfork*(2) for details.

HARDWARE DEPENDENCIES

    Series 200:

        [ENOSPC]   *Fork* will fail if there is not enough swapping memory to create the new process.

        [ENOMEM]   *Fork* will fail if there is not enough physical memory to create the new process.

    Series 500:

        [ENOMEM]   *Fork* will fail if there is not enough physical memory to create the new process.

      profil(2) is not supported on Series 500 Computers.

RETURN VALUE

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

SEE ALSO

    exec(2), nice(2), plock(2), ptrace(2), semop(2), shmop(2), signal(2), times(2), ulimit(2), umask(2), vfork(2), wait(2).

NAME

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

SYNOPSIS

fsync(fildes)
int fildes;

HP-UX COMPATABILITY

Level: Basic Calls: HP–UX/RUN ONLY
           Real Time extensions: HP–UX/STANDARD – Real Time

Origin: UCB and HP

DESCRIPTION

Fsync causes all modified data and attributes of *fildes* to be moved to a permanent storage device. This normally results in all in–core modified copies of buffers for the associated file to be written to a disk. *Fsync* applies to ordinary files, and applies to block special devices on systems which permit I/O to block special devices.

Fsync should be used by programs which require a file to be in a known state; for example in building a simple transaction facility.

ERRORS

Fsync will fail if one of the following conditions is true and *errno* will be set accordingly:

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

[EINVAL]   *Fildes* refers to a file type to which *fsync* does not apply.

RETURN VALUE

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

WARNING

If the process has multiple file descriptors open on a file and the file descritpors have asynchronous writes pending, then the process cannot be guaranteed that the buffers associated with these writes will be flushed by the *fsync*(2). The process must wait until all of the writes have completed before performing an *fsync*(2) call.

BUGS

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

SEE ALSO

fcntl(2), fcntl(7), open(2), select(2), sync(2), sync(8).

NAME

ftime - get date and time more precisely

SYNOPSIS

#include <sys/types.h>
#include <sys/timeb.h>
ftime(tp)
struct timeb *tp;

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD - Version 7 compatibility

Origin: Version 7

Remarks: This facility is provided for backwards compatibility with Version 7 systems. Either
            time or *gettimeofday* should be used for all new code.

DESCRIPTION

Ftime entry fills in a structure pointed to by its argument, as defined by *<sys/timeb.h>*:

```
/*
 * Structure returned by ftime system call
 */
struct timeb {
        time_t   time;
        unsigned short millitm;
        short    timezone;
        short    dstflag;
};
```

The structure contains the time in seconds since 00:00:00 GMT, January 1, 1970, up to 1000 mil-
liseconds of more–precise interval, the local timezone (measured in minutes of time westward from
Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time applies locally during
the appropriate part of the year. *Gettimeofday* should be consulted for more details on the mean-
ing of the timezone field.

This call can be accessed by giving the –**lV7** option to *ld*(1).

Ftime can fail for exactly the same reasons as *gettimeofday*(2).

HARDWARE DEPENDENCIES

Series 500:

        Ftime is not supported on the Series 500.

SEE ALSO

date(1), gettimeofday(2), time(2), stime(2), ctime(3)

BUGS

The millisecoud value usually has a granularity greater than one due to the resolution of the sys-
tem clock. Depending on any granularity (particularly of one) will render code non–portable.

NAME

getgroups - get group access list

SYNOPSIS

#include <sys/param.h>

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

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: UCB

Remarks: Not available on Series 500.

DESCRIPTION

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

RETURN VALUE

A non–negative value indicates that the call succeeded, and is the number of elements in *gidset* . A value of -1 indicates that an error occurred, and the error code is stored in the global variable *errno* .

The possible errors for *getgroups* are:

[EFAULT]   *gidset* specifies an invalid address.

[EINVAL]   *ngroups* is less than the number of groups in the current group access list of the process.

HARDWARE DEPENDENCIES

Not available on Series 500.

SEE ALSO

setgroups(2), initgroups(3C)

NAME

gethostname - get name of current host

SYNOPSIS

char hostname[];

gethostname(hostname, sizeof (hostname));

HP–UX COMPATIBILITY

Level: HP–UX/RUN ONLY

Origin: UCB

DESCRIPTION

Gethostname returns the standard host name for the current processor, as set by *sethostname*(2). The name is truncated to sizeof(hostname)–1 and is null-terminated.

Gethostname can fail if:

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

SEE ALSO

hostname(1), uname(1), sethostname(2), uname(2).

NAME

    getitimer, setitimer - get/set value of interval timer

SYNOPSIS

    #include <sys/time.h>

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

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

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

HP–UX COMPATIBILITY

    Level: HP–UX/STANDARD

    Origin: UCB

DESCRIPTION

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

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

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

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

    Time values smaller than the resolution of the system clock are rounded up to this resolution. The machine–dependent clock resolution is *1/HZ* seconds, where the constant *HZ* is defined in *<sys/param.h>*. Time values larger than an implementation–specific maximum value are rounded down to this maximum. The maximum values for the three interval timers are specified by the constants *MAX_ALARM*, *MAX_VTALARM*, and *MAX_PROF* defined in *<sys/param.h>*. On all implementations, these values are guaranteed to be at least 31 days (in seconds).

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

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

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

NOTES

    Three macros for manipulating time values are defined in *<sys/time.h>*. *Timerclear* sets a time

value to zero, *timerisset* tests if a time value is non–zero, and *timercmp* compares two time values (beware that $>=$ and $<=$ do not work with this macro).

The timer used with *ITIMER_REAL.* is the same as that used by *alarm(2).* Thus successive calls to *alarm, getitimer,* and *setitimer* will set and return the state of a single timer.

RETURN VALUE

If the calls succeed, a value of 0 is returned. If an error occurs, the value -1 is returned, and a more precise error code is placed in the global variable *errno*. *Getitimer* or *setitimer* can fail if:

[EFAULT]   The *value* structure specified a bad address.

[EINVAL]   A *value* structure specified an invalid time.

[EINVAL]   *Which* does not specify one of the three possible timers.

HARDWARE DEPENDENCIES
Series 500

An error is generated if a call is made to *getitimer* or *setitimer* in the [vfork,exec] window.

[EINVAL]   Call not allowed in [vfork,exec] window

SEE ALSO
alarm(2), signal(2), gettimeofday(2)

NAME

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

SYNOPSIS

    int getpid ( )

    int getpgrp ( )

    int getppid ( )

HP–UX COMPATIBILITY

    Level: HP–UX/RUN ONLY

    Origin: System V

DESCRIPTION

    Getpid returns the process ID of the calling process.

    Getpgrp returns the process group ID of the calling process.

    Getppid returns the parent process ID of the calling process.

SEE ALSO

    exec(2), fork(2), setpgrp(2), signal(2).

NAME

　　getprivgrp, setprivgrp - get and set special attributes for group

SYNOPSIS

　　#include <sys/privgrp.h>

　　int setprivgrp(grpid, mask)
　　int grpid, mask[PRIV_MASKSIZE];

　　int getprivgrp(grplist)
　　struct privgrp_map grplist[PRIV_MAXGRPS];

HP-UX COMPATIBILITY

　　Level:　　HP–UX/STANDARD

　　Origin:　　HP

DESCRIPTION

　　Setprivgrp associates a kernel capability with a group id. This allows subsetting of super-user like privileges for members of a particular group or groups. *Setprivgrp* takes two arguments: the integer group id and a mask of permissions. The mask is created by treating the access types defined in <sys/privgrp.h> as bit numbers (using 1 for the least significant bit). Thus, privilege number 5 would be represented by the bit $1<<(5-1)$ or 16. More generally, privilege **p** is represented by:

$$mask[((\mathbf{p}{-}1) \; / \; BITS\_PER\_INT)] \; \& \; (1 << ((\mathbf{p}{-}1) \; \% \; BITS\_PER\_INT)).$$

　　As it is possible to have more than **word size** distinct privileges, mask is a pointer to an integer array of size **PRIV_MASKSIZ**.

　　Setprivgrp privileges include those specified in the file **sys/privgrp.h**. A process may access the system call protected by a specific privileged group if it belongs to or has an effective group id of a group having access to the system call.

　　Specifying a *grpid* of **PRIV_NONE** causes privileges to be revoked on all privileged groups having any of the privileges specified in *mask*. Specifying a *grpid* of **PRIV_GLOBAL** causes privileges to be granted to all processes.

　　The constant **PRIV_MAXGRPS** in <sys/privgrp.h> defines the system limit on the number of groups which can be assigned privileges. One of these is always the psuedo–group **PRIV_GLOBAL**, allowing for **PRIV_MAXGRPS**-1 actual groups.

　　Getprivgrp returns a table of the privileged group assignments into a user supplied structure. *Grplist* points to an array of structures of type **privgrp_map** associating a groupid with a privilege mask. Privilege masks are formed by *or*ing together elements from the access types specified in <sys/privgrp.h>. The array may have gaps in it distinguished as having a **priv_groupno** field of **PRIV_NONE**. The group number **PRIV_GLOBAL** gives the global privilege mask. Only information about groups which are in the user's group access list, or about his real or effective group id, is returned to an ordinary user. The complete set is returned to the super–user.

NOTES

　　Only the super user may use *setprivgrp*.

ERRORS

　　Setprivgrp returns –1 and an error code in *errno* if:

　　[EPERM]　　　　The caller is not super user.

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

[EINVAL]   *Mask* has bits set for one or more unknown privileges.

[E2BIG]   The   request   would   require   assigning   privileges   to   more   than
                PRIV_MAXGRPS groups.

Getprivgrp returns −1 and an error code in *errno* if:

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

Both calls return 0 on success.

HARDWARE DEPENDENCIES
Not implemented on Series 500 or Integral PC.

SEE ALSO
getprivgrp(1), setprivgrp(1M), setgroups(2), privgrp(5)

NAME
gettimeofday, settimeofday - get/set date and time

SYNOPSIS
#include <time.h>

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

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

HP-UX COMPATIBILITY
Level: HP-UX/STANDARD

Origin: UCB

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

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

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

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

The *timezone* structure indicates the local time zone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time applies locally during the appropriate part of the year (the value of the flag identifies the algorithm to be used for Daylight Saving time). Programs should use this timezone information only in the absence of the TZ environment variable.

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

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

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

[EFAULT]   An argument address referenced invalid memory.

[EPERM]   A user other than the super-user attempted to set the time.

SEE ALSO
date(1), stime(2), time(2), ctime(3)

BUGS
The microsecond value usually has a granularity much greater than one due to the resolution of the system clock. Depending on any granularity (particularly of one) will render code non-portable.

NAME

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

SYNOPSIS

      unsigned short getuid ( )

      unsigned short geteuid ( )

      unsigned short getgid ( )

      unsigned short getegid ( )

HP–UX COMPATIBILITY

      Level: HP–UX/RUN ONLY

      Origin: System V

DESCRIPTION

      Getuid returns the real user ID of the calling process.

      Geteuid returns the effective user ID of the calling process.

      Getgid returns the real group ID of the calling process.

      Getegid returns the effective group ID of the calling process.

SEE ALSO

      setuid(2).

NAME

ioctl - control device

SYNOPSIS

#include <sys/ioctl.h>

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

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System V

DESCRIPTION

Ioctl performs a variety of functions on character special files (devices). The write–ups of various devices in Section 4 discuss how *ioctl* applies to them. The type of *Arg* is dependent on the specific *ioctl* call, as described in Section 4.

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

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

[ENOTTY]   The request is not appropriate to the selected device.

[EINVAL]   *Request* or *arg* is not valid.

[EINTR]   A signal was caught during the *ioctl* system call.

Request is made up of several fields. They encode the size and direction of the argument (refer enced by *arg*), as well as the desired command. An enumeration of the request fields are:

IOC_IN (Bit 31)
Argument is read by the driver. (That is, the argument is copied from the application to the driver.)

IOC_OUT (Bit 30)
Argument is written by the driver. (That is, the argument is copied from the driver to the application.)

IOCSIZE_MASK
Number of bytes in the passed argument. A nonzero size indicates that *arg* is a pointer to the passed argument. A zero size indicates that *arg* is the passed argument (if the driver wants to use it), and is not treated as a pointer.

IOCCMD_MASK (Bits 15–0)
The request command itself.

When both IOC_IN and IOC_OUT are zero, it can be assumed that *request* is not encoded for size and direction, for compatibility purposes. Requests which do not require any data to be passed and requests which use *arg* as a value (as opposed to a pointer), have the IOC_IN bit set to one and the IOCSIZE_MASK field set to zero.

Note: any data structure referenced by *arg* may *not* contain any pointers.

RETURN VALUE

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

SEE ALSO

Section 4.

NAME
     kill - send a signal to a process or a group of processes

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

HP-UX COMPATIBILITY
     Level: HP–UX/RUN ONLY

     Origin: System V

DESCRIPTION
     Kill sends a signal to a process or a group of processes. The process or group of processes to
     which the signal is to be sent is specified by *pid*. The signal that is to be sent is specified by *sig*
     and is either one from the list given in *signal*(2), or 0. If *sig* is 0 (the null signal), error checking
     is performed but no signal is actually sent. This can be used to check the validity of *pid*.

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

     The processes with a process ID of 0 and a process ID of 1 are special processes (see *glossary*(9))
     and will be referred to below as *proc0* and *proc1*, respectively. The value **KILL_ALL_OTHERS**
     is defined in the file *<sys/signal.h>* and is guaranteed not to be the ID of any process in the sys-
     tem or the negation of the ID of any process in the system.

     If *pid* is greater than zero and not equal to **KILL_ALL_OTHERS**, *sig* will be sent to the process
     whose process ID is equal to *pid*. *Pid* may equal 1.

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

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

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

     If *pid* is **KILL_ALL_OTHERS** the behavior is the same as for *pid* equal to -1 except that *sig* is
     not sent to the calling process.

     If *pid* is negative but not -1 or **KILL_ALL_OTHERS**, *sig* will be sent to all processes (exclud-
     ing *proc0* and *proc1*) whose process group ID is equal to the absolute value of *pid*, and whose real
     and/or effective *uid* meet the constraints described above for matching *uid*s.

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

     [EINVAL]   *Sig* is not a valid signal number or zero.

     [EINVAL]   *Sig* is SIGKILL and *pid* is 1 (proc1).

     [ESRCH]   No process can be found corresponding to that specified by *pid*.

     [EPERM]   The user ID of the sending process is not super–user, and its real or effective user
                      ID does not match the real or saved effective user ID of the receiving process.

HARDWARE DEPENDENCIES
     Series 500:
                      References to *proc0* above are invalid because *proc0* does not exist on Series 500.

     Series 200:
                      A special process known as the pagedaemon has process ID 2. All references to *proc0* and
                      proc1 apply to it as well.

RETURN VALUE

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

SEE ALSO

kill(1), getpid(2), setpgrp(2), signal(2), and sigvector(2).

NAME

link - link to a file

SYNOPSIS

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

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System V

DESCRIPTION

Path1 points to a path name naming an existing file. *Path2* points to a path name naming the new directory entry to be created. *Link* creates a new link (directory entry) for the existing file.

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

[ENOTDIR]	A component of either path prefix is not a directory.
[ENOENT]	A component of either path prefix does not exist.
[EACCES]	A component of either path prefix denies search permission.
[ENOENT]	The file named by *path1* does not exist.
[EEXIST]	The link named by *path2* exists.
[EPERM]	The file named by *path1* is a directory and the effective user ID is not super-user.
[EXDEV]	The link named by *path2* and the file named by *path1* are on different logical devices (file systems).
[ENOENT]	*Path2* points to a null path name.
[EACCES]	The requested link requires writing in a directory with a mode that denies write permission.
[EROFS]	The requested link requires writing in a directory on a read-only file system.
[EFAULT]	*Path* points outside the allocated address space of the process.
[ENOENT]	*Path1* or *path2* is null.
[EMLINK]	The maximum number of links to a file would be exceeded.

HARDWARE DEPENDENCIES

Series 500:

For Structured Directory Format (SDF) discs, if *path2* is "..", then that directory's i-node will be altered such that its ".." entry points to the directory specified by *path1*. In this way, the super-user can establish the parent directory of an existing directory.

RETURN VALUE

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

SEE ALSO

cp(1), link(1), unlink(2).

NAME

lockf - provide semaphores and record locking on files

SYNOPSIS

#include <unistd.h>

lockf(fildes, function, size)
long size;
int fildes, function;

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: /usr/group

DESCRIPTION

Lockf will allow regions of a file to be used as semaphores (advisory locks) or accessable only by the locking process (enforcement mode record locks). Other processes which attempt to access the locked resource will either return an error or sleep until the resource becomes unlocked. All the locks for a process are removed when the process closes the file or terminates.

Fildes is an open file descriptor.

Function is a control value which specifies the action to be taken. The permissible values for *function* are defined in <**unistd.h**> as follows:

```
#define F_ULOCK      0   /* Unlock a region */
#define F_LOCK       1   /* Lock a region */
#define F_TLOCK      2   /* Test and Lock a region */
#define F_TEST       3   /* Test region for lock */
```

All other values of *function* are reserved for future extensions and will result in an error return if not implemented.

F_TEST is used to detect if a lock by another process is present on the specified region. **F_TEST** returns zero if the region is accessable and minus one (-1) if it is not; in this case *errno* will be set to EACCES. **F_LOCK** and **F_TLOCK** both lock a region of a file if the region is available. **F_ULOCK** removes locks from a region of the file.

Size is the number of contiguous bytes to be locked or unlocked. The resource to be locked starts at the current offset in the file, and extends forward for a positive *size*, and backward for a negative *size* (the preceding byte(s), up to but not including the current offset). If *size* is zero the region from the current offset through the end of the largest possible file is locked (i.e., from the current offset through the present or any future end-of-file). An area need not be allocated to the file in order to be locked, as such locks may exist past the end of the file.

The regions locked with **F_LOCK** or **F_TLOCK** may, in whole or part, contain or be contained by a previously locked region for the same process. When this occurs or if adjacent regions occur, the regions are combined into a single region. If the request requires that a new element be added to the table of active locks and this table is already full, an error is returned, and the new region is not locked.

F_LOCK and **F_TLOCK** requests differ only by the action taken if the resource is not available: **F_LOCK** will cause the calling process to sleep until the resource is available, and the **F_TLOCK** will return an [EACCES] error if the region is already locked by another process.

F_ULOCK requests may, in whole or part, release one or more locked regions controlled by the process. When regions are not fully released, the remaining regions are still locked by the process. Releasing the center section of a locked region requires an additional element in the table of active locks. If this table is full, an [EDEADLK] error is returned, and the requested region is not released.

Regular files with the file mode of **S_ENFMT** not having the group execute bit set will have an enforcement policy enabled. With enforcement enabled, reads and writes which would access a locked region will sleep until the entire region is available. File access by other system functions like *exec* are not subject to the enforcement policy. Locks on directories, pipes, and special files are advisory only; no enforcement policy will be used.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by access-ing another process's locked resource. Thus calls to *lockf*, *read*, or *write* scan for a deadlock prior to sleeping on a locked resource. Deadlock is not checked for the *wait* and *pause* system calls, so potential for deadlock is not eliminated. A *creat* call or an *open* call with the O_CREATE and O_TRUNC flags set on a regular file will return [EACCES] error if another process has locked part of the file and the file is currently in enforcement mode.

RETURN VALUE

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

ERRORS

Lockf will fail if any of the following occur:

[EACCES]   Will be returned for requests in which the region is already locked by another process.

[ENOENT]   The named file does not exist.

[EBADF]   If is not a valid, open file descriptor.

[EDEADLK]   Will be returned by *lockf* if a deadlock would occur; or if the number of entries in the lock table would exceed a system dependent maximum. HP-UX guarantees this value to be at least 50.

[EINVAL]   Will be returned if:

              function is not one of the functions specified above.

              The resulting upper bound of the region to be locked will be greater than 2^30.

              The current *offset* is greater than 2^30.

              The resulting lower bound of the region to be locked will be negative.

WARNINGS

Deadlock conditions may arise when either the *wait* or *pause* system calls are used in conjunction with enforced locking; see *wait*(2) and *pause*(2) for details.

File and record locking using file descriptors obtained through *dup*(2) or *link*(2) may not work as expected, e.g. unlocking regions which were locked using either file descriptor may also unlock regions which were locked using the other file descriptor.

The shell will wait for locked files with enforcement mode set to become unlocked before executing them.

As a side effect of the definition of enforcement mode, it is possible to have files which are execut-able for the owner and others, but which result in an error if executed by a groupmember.

BUGS

Unexpected results may occur in a process that does buffering in the user address space. The process may later read/write data which is/was locked. The standard I/O package, *stdio*(3), is the most common source of unexpected buffering.

In a hostile environment locking may be misused by holding key public resources locked. This is

particularly true with public read files that have enforcement mode enabled.

HARDWARE DEPENDENCIES

Series 200 and 500:

The system's process accounting routine will ignore any locks put on the process accounting file.

SEE ALSO

open(2), creat(2), read(2), write(2), close(2), chmod(2), stat(2), wait(2), pause(2), acct(2) link(2), dup(2).

NAME

       lseek - move read/write file pointer; seek

SYNOPSIS

       #include <unistd.h>
       long lseek (fildes, offset, whence)
       int fildes;
       long offset;
       int whence;

HP–UX COMPATIBILITY

       Level: HP–UX/RUN ONLY

       Origin: System V

DESCRIPTION

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

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

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

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

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

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

       [EBADF]   *Fildes* is not an open file descriptor.

       [ESPIPE]   *Fildes* is associated with a pipe or fifo.

       [EINVAL and SIGSYS signal]
              Whence is not 0, 1 or 2.

       [EINVAL]   The resulting file pointer would be negative.

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

RETURN VALUE

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

SEE ALSO

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

NAME

memadvise - advise OS about segment reference patterns

SYNOPSIS

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

memadvise(addr, len, behav, adrtype)
caddr_t   addr;
int   len, behav;
enum   memtype {mem_code, mem_data} adrtype;

HP-UX COMPATIBILITY

Level: Backing Store Control – HP–UX/EXTENDED

Origin: HP

Remarks: *Memadvise* is not currently implemented on the Series 200.

DESCRIPTION

The purpose of this call is to allow an application program to notify the system of its known patterns of reference in specific areas of process memory. The intent is to allow the system to then adapt its memory management algorithms and/or policies based on this knowledge to maximize the performance of the program. For example, a program that uses a very large hash table might inform the system of its random patterns of reference to this area. The system might, then, elect not to do any pre-fetching or clustered reads in this area.

Addr is the starting address of the area in question and *len* is the length in bytes. *Addr* may be any legal address in the process's address space. Since some implementations use different (and indistinguishable) addressing formats for code and data space, *adrtype* is used to indicate whether *addr* is a code or data address. On systems with a uniform addressing format for code and data, *adrtype* will have no effect.

The boundaries of the address space for which the advice is applied may be rounded up and/or down to appropriate system dependent values (e.g. pages, segments, blocks, etc).

Variable *behav* describes the reference pattern in the specified area:

MEM_NORMAL

No known extraordinary patterns of reference.

MEM_SEQ

References are highly sequential in nature.

MEM_RANDOM

References are totally random and unpredictable.

MEM_NEEDED

Area is expected to be highly referenced in near future.

MEM_NOTNEEDED

Area is not expected to be referenced in the near future.

Memadvise may be reduced to a no–op, or some of the behavior types may be ignored (treated as no–ops).

HARDWARE DEPENDENCIES

This system call is supported on Series 500 only.

SEE ALSO

ems(2), memallc(2)

NAME

memallc, memfree - allocate and free address space

SYNOPSIS

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

caddr_t	**memallc(fileid, offset, len, maxlen, type, mode);**
int	**fileid, offset, len;**
int	**maxlen, type, mode;**
int	**memfree(addr);**
caddr_t	**addr;**

HP-UX COMPATIBILITY

Level: Extended Memory – HP–UX/EXTENDED

Origin: HP

Remarks: *Memallc* and *memfree* are not currently implemented on the Series 200.

DESCRIPTION

Memallc allocates a memory segment (i.e. a contiguous piece of process address space) and returns a pointer to it. The memory segment may be shared (i.e. accessible by other processes) or private. Private segments are copied on *fork*(2), giving separate, per -process images of the segment. Shared segments are not copied across *fork*(2) but, instead, both processes have access to the same memory space. The segment may optionally be initialized to the contents of a specific open file (private mapped file) or can be made equivalent to a specific file (shared mapped file).

Fileid is the HP–UX file id of an open file which will be mapped into the process's address space. *Fileid* must refer to a file on a CS–80 disc. If *fileid* is -1, the allocated address space will be initialized to zeros. A mapping of a file (either shared or private) generates an implicit reference to the file (similar to the result of *dup*(2)). Subsequent to the mapping, *fileid* may safely be closed.

Offset specifies the starting point in *fileid* (i.e. byte offset) where mapping is to begin. The value returned by *memallc* is a pointer to the byte in the new address space that corresponds to byte *offset*. If *fileid* is not specified (i.e. set to –1), *offset* is ignored.

Len specifies the size (in bytes) of the address space. The guaranteed range of accessibility is from *ptr* thru *ptr+len–1* (where *ptr* is the value returned by the *memallc* call). Depending on the value of *offset*, *len*, and the specific implementation, additional data space MAY be accessible at addresses less than *ptr* and/or greater than *ptr+len–1* but the effects of reading and/or writing these areas are undefined.

If *len+offset* is greater than the size of the file, the additional address space is initialized to zeros. If the segment is shared, the file is extended to the required size (if fileid is not writable, the call fails). A *creat*(2) call on a file that has a shared mapping applied to it will zero the file but will not alter the file size.

Maxlen specifies the maximum length to which a segment may grow using *memvary*(2).

Type specifies the attributes assigned to the segment, which is constructed by taking the union of the desired attributes: MEM_SHARED, MEM_PRIVATE, MEM_PAGED, MEM_DATA, or MEM_CODE (see *ems*(2)).

Mode specifies the access permissions assigned to the segment for the requesting process.

MEM_R, MEM_W, MEM_X:

Initial access modes to be assigned to segment (see *memchmd*(2)).

Note that all MEM_SHARED mappings of a specific file must use identical access modes. An attempt to map a file with access modes different than those already in effect will return an error [EACCES].

Memfree deallocates a memory segment created by *memallc*. It takes, as an argument, a pointer returned by *memallc*. When the segment is shared, the memory will not be deallocated until the last reference to the memory is removed.

The number of segments allocated to a given process at any one time may be limited to a system dependent maximum of **MAXSEGS** found in **ems.h**.

RETURN VALUE
Upon successful completion, *memallc* returns the byte pointer to the address space. Otherwise, a value of −1 is returned and *errno* is set to indicate error.

HARDWARE DEPENDENCIES
This system call is supported by Series 500 only.

SEE ALSO
ems(2), memvary(2), memchmd(2).

BUGS
Non–paged segments can be extended past **maxlen** using *memvary(2)*.

NAME

memchmd - change memory segment access modes

SYNOPSIS

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

int   memchmd (addr, mode);
caddr_t   addr;
int   mode;

HP–UX COMPATIBILITY

Level: Extended Memory   HP UX/EXTENDED

Origin: HP

Remarks: *Memchmd* is not currently implemented on the Series 200.

DESCRIPTION

This procedure may be used to change the access mode of a memory segment created by *memallc*(2). The procedure returns the previous access mode (or –1 if there is an error).

Addr is the segment pointer returned by *memallc*(2).

The access modes for a shared segment is an attribute of the segment and is the same for all processes sharing the segment or any portion thereof. The access mode of a segment may not be changed if it is being shared with any other process (e.g. more than one *memallc* of a peculiar file, or a *memallc* followed by a *fork*(2)). An attempt to *memchmd* such a shared segment will return an error [EACCES].

The access mode of a MEM_PRIVATE segment may be changed without restrictions.

The definition of the access modes are:

MEM_X   Execute capability

MEM_W   Write capability

MEM_R   Read capability

An error is returned if *addr* is not a valid segment pointer.

Access modes granted to a MEM_SHARED file mapped segment may not exceed the access modes granted to the user of the file when it was opened.

RETURN VALUE

Upon successful completion, *memchmd*(2) returns the old set of access modes. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

HARDWARE DEPENDENCIES

This system call is supported on Series 500 only.

SEE ALSO

ems(2), memallc(2), memvary(2).

NAME

memlck, memulck - lock/unlock process address space or segment

SYNOPSIS

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

int        memlck (addr, len, adrtype);
caddr_t    addr;
int        len;
enum       memtype {mem_code, mem_data} adrtype;

int        memulck (addr, len, adrtype);
caddr_t    addr;
int        len;
enum       memtype {mem_code, mem_data} adrtype;
```

HP-UX COMPATIBILITY

Level: Backing Store Control – HP–UX/RUN ONLY

Origin: HP

Remarks: *Memlck* and *memulck* are not currently implemented on the Series 200.

DESCRIPTION

Memlck is used to lock a section of process address space into physical memory. This call may take a substantial amount of time to complete, but the address space in question is guaranteed to be in memory and locked upon successful completion of the call. The locked address space will not migrate to backing store regardless of process state and will, furthermore, remain at the same physical address space for the duration of the lock. Locks are not inherited across *fork*(2). Multiple locks on any address range can occur (unlocking requires that as many unlocks as locks occur). The locks will be segment local, and unlocking may be done by a process unrelated to the one which did the locking. A locked segment will be released when there are no processes with references to the locked segment. (This may occur either via *memfree*(2) or process death.)

Addr is the starting address of the area in question and *len* is the length in bytes. *Addr* may be any legal address in the process's address space. Since some implementations use different (and indistinguishable) addressing formats for code and data space, *adrtype* is used to indicate whether addr is a code or data address. On systems with a uniform addressing format for code and data, *adrtype* will have no effect.

The boundaries of the locked address space may be rounded up (on the upper end of the address range) and down (on the lower end of the address range) to appropriate system dependent values (e.g. pages, segments, blocks, etc). Locking will not cross segment boundaries. For example, one *memlck* call cannot lock part of a text segment and part of a data segment.

Memulck undoes the effects of a *memlck*.

The use of this call is restricted to the super–user.

This call may be reduced to a no–op.

RETURN VALUE

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

HARDWARE DEPENDENCIES

This system call is supported on Series 500 only.

SEE ALSO

ems(2), memallc(2).

NAME

memvary - modify segment length

SYNOPSIS

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

int   memvary(addr, len);
caddr_t   addr;
int   len;

HP–UX COMPATIBILITY

Level: Extended Memory – HP–UX/EXTENDED

Origin: HP

Remarks: *Memvary* is not currently implemented on the Series 200.

DESCRIPTION

Memvary allows the modification of the size of the memory space allocated by *memallc*(2).

Addr is the pointer to the address space which can be either shared or private. If the address space has been file mapped and is extended beyond the the end of the file, then the file will also reflect the change in length. When the file mapped address space is reduced, the actual file length will be unchanged and the file space after the end of the mapped file space will also remain unchanged. A change in length for a private file mapped address space will have no effect on the source file.

Len specifies the new length of the address space. In the case of an error, the address space and file space will be the same as before the intrinsic call.

When private file mapped address space is extended, the additional address space is initialized to zeroes. When shared file mapped address space is extended, the additional space is initialized to the contents of the file, or zeros if the file is extended.

The address space cannot be extended beyond the 'maxlen' specified during the *memallc*(2) intrinsic call.

RETURN VALUE

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

HARDWARE DEPENDENCIES

This system call is supported by Series 500 only.

SEE ALSO

ems(2), memallc(2), memchmd(2).

NAME

    mkdir - make a directory file

SYNOPSIS

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

HP–UX COMPATIBILITY

    Level: HP–UX/STANDARD

    Origin: UCB

    Remarks: Not all systems implement this as a system call; some use a library call to the
              mkdir(1) command to achieve the same effect. The errors documented below will
              appear in any case, and no error messages will ever be printed.

DESCRIPTION

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

    The directory's owner ID is set to the process's effective user ID. The directory's group ID is set
    to the process's effective group ID.

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

RETURN VALUE

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

ERRORS

    Mkdir will fail and no directory will be created if:

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

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

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

    [EEXIST]   The named file exists.

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

    [EIO]   An I/O error occured while writing to the file system.

SEE ALSO

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

NAME

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

SYNOPSIS

    #include <mknod.h>
    int mknod (path, mode, dev)
    char *path;
    int mode;
    dev_t dev;

HP-UX COMPATIBILITY

    Level: HP-UX/RUN ONLY

    Origin: System V

DESCRIPTION

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

        0170000 file type; one of the following:
                0010000   fifo special
                0020000   character special
                0040000   directory
                0060000   block special
                0100000 or 0000000 ordinary file
                0110000   network special
        0004000 set user ID on execution
        0002000 set group ID on execution
        0001000 save text image after execution
        0000777 access permissions; constructed from the following:
                0000400   read by owner
                0000200   write by owner
                0000100   execute (search on directory) by owner
                0000070   read, write, execute (search) by group
                0000007   read, write, execute (search) by others

Values of *mode* other than those above are undefined and should not be used.

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

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

Dev is meaningful only if *mode* indicates a block or character special file, and is ignored otherwise. It is an implementation and configuration dependent specification of a character or block I/O device. A device name is created by using the **makedev** macro defined in *mknod.h*. The arguments to **makedev** are the major and minor device numbers, the value and interpretation of which are implementation dependent. The result of **makedev** is an object of type **dev_t**.

Mknod may be invoked only by the super-user for file types other than FIFO special.

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

[EPERM]   The effective user ID of the process is not super-user.

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

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

[EROFS]   The directory in which the file is to be created is located on a read-only file system.

[EEXIST]   The named file exists.

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

[ENOENT]   *Path* is null.

[EACCES]   *Path* is in a directory that denies write permission, *mode* is for fifo special file,
                and the caller is not super–user.

HARDWARE DEPENDENCIES
Series 200 and 500:
> An additional value is available for network special files under file type. Its value is
> 0110000.

Integral PC:
> The super–user capabilities are provided to the normal user.

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

SEE ALSO
mkdir(1), chmod(2), mkdir(2), exec(2), umask(2), fs(5), mknod(5), mknod(8).

NAME
  mount - mount a file system

SYNOPSIS
  int mount (spec, dir, rwflag)
  char *spec, *dir;
  int rwflag;

HP-UX COMPATIBILITY
  Level: HP–UX/RUN ONLY

  Origin: System V

DESCRIPTION
  Mount requests that a removable file system contained on the block special device file identified by *spec* be mounted on the directory identified by *dir*. *Spec* and *dir* are pointers to path names.

  Upon successful completion, references to the file *dir* will refer to the root directory on the mounted file system.

  The low-order bit of *rwflag* is used to control write permission on the mounted file system; if **1**, writing is forbidden, otherwise writing is permitted according to individual file accessibility.

  Mount may be invoked only by the super–user.

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

  [EPERM]   The effective user ID is not super–user.

  [ENOENT]   Any of the named files does not exist.

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

  [ENOTBLK]   *Spec* is not a block special device.

  [ENXIO]   The device associated with *spec* does not exist.

  [ENOTDIR]   *Dir* is not a directory.

  [EFAULT]   *Spec* or *dir* points outside the allocated address space of the process.

  [EBUSY]   *Dir* is currently mounted on, is someone's current working directory, or is other-wise busy.

  [EBUSY]   The device associated with *spec* is currently mounted.

  [EBUSY]   There are no more mount table entries.

  [ENOENT]   *Spec* or *dir* is null.

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

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

BUGS
  If *mount* is called from the program level (i.e. not called from *mount*(1)), the table of mounted devices contained in /etc/mnttab is not updated.

HARDWARE DEPENDENCIES
  Integral PC:
        The super–user capabilities are provided to the normal user.

        If a file system is mounted via this kernel call on the Integral PC, it can be unmounted only using the *umount* kernel call. The *unmount*(1) and *mount*(1) commands do not operate on a file system mounted via the *mount* kernel call.

The *unmount* command is unable to unmount any file system mounted with the *mount* kernel call. The Integral PC file system utilities cannot properly deal with file systems mounted with the *mount* kernel call.

SEE ALSO

mount(1), umount(2).

NAME

     msgctl - message control operations

SYNOPSIS

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

     int msgctl (msqid, cmd, buf)
     int msqid, cmd;
     struct msqid_ds *buf;

HP-UX COMPATABILITY

     Level: HP-UX/STANDARD

     Origin: System V

DESCRIPTION

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

     IPC_STAT   Place the current value of each member of the data structure associated with *msqid*
                  into the structure pointed to by *buf*. The contents of this structure are defined in
                  glossary(9). {READ}

     IPC_SET   Set the value of the following members of the data structure associated with *msqid*
                  to the corresponding value found in the structure pointed to by *buf*:
                            msg_perm.uid
                            msg_perm.gid
                            msg_perm.mode /* only low 9 bits */
                            msg_qbytes

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

     IPC_RMID   Remove the message queue identifier specified by *msqid* from the system and des-
                  troy the message queue and data structure associated with it. This *cmd* can only
                  be executed by a process that has an effective user ID equal to either that of
                  super-user or to the value of either **msg_perm.uid** or **msg_perm.cuid** in the
                  data structure associated with *msqid*.

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

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

     [EINVAL]   *Cmd* is not a valid command.

     [EACCES]   *Cmd* is equal to **IPC_STAT** and {READ} operation permission is denied to the
                  calling process (see *glossary*(9)).

     [EPERM]   *Cmd* is equal to **IPC_RMID** or **IPC_SET** and the effective user ID of the calling
                  process is not equal to that of super user and it is not equal to the value of
                  msg_perm.uid in the data structure associated with *msqid*.

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

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

RETURN VALUE

     Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and

errno is set to indicate the error.

SEE ALSO

msgget(2), msgop(2), stdipc(3).

NAME
msgget - get message queue

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

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

HP-UX COMPATIBILITY
Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION
Msgget returns the message queue identifier associated with *key*.

A message queue identifier and associated message queue and data structure (see *glossary*(9)) are created for *key* if one of the following are true:

Key is equal to **IPC_PRIVATE**.

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

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

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

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

Msg_qnum, **msg_lspid**, **msg_lrpid**, **msg_stime**, and **msg_rtime** are set equal to 0.

Msg_ctime is set equal to the current time.

Msg_qbytes is set equal to the system limit.

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

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

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

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

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

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

SEE ALSO
msgctl(2), msgop(2), stdipc(3).

NAME

msgop - message operations

SYNOPSIS

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

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

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

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

DESCRIPTION

Msgsnd is used to send a message to the queue associated with the message queue identifier specified by *msqid*. {WRITE} *Msgp* points to a structure containing the message. This structure is composed of the following members:

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

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

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

The number of bytes already on the queue is equal to **msg_qbytes** (see *glossary*(9)).

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

These actions are as follows:

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

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

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

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

The calling process receives a signal that is to be caught and the signal handler does not specify that the call is to be restarted (see *sigvector*(2)). In this case the message is not sent and the calling process resumes execution in the manner prescribed in *signal*(2).

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

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

[EACCES]   Operation permission is denied to the calling process (see *glossary*(9)).

[EINVAL]   *Mtype* is less than 1.

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

[EINVAL]   *Msgsz* is less than zero or greater than the system–imposed limit.

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

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see glossary (9)).

> **Msg_qnum** is incremented by 1.

> **Msg_lspid** is set equal to the process ID of the calling process.

> **Msg_stime** is set equal to the current time.

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

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

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

Msgtyp specifies the type of message requested as follows:

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

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

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

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

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

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

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

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

>> The calling process receives a signal that is to be caught and the signal handler does not specify that the call is to be restarted (see *sigvector*(2)). In this case a message is not received and the calling process resumes execution in the manner prescribed in *signal*(2).

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

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

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

[EINVAL]   *Msgsz* is less than 0.

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

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

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

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see glossary (9)).

Msg_qnum is decremented by 1.

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

Msg_rtime is set equal to the current time.

RETURN VALUES

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

Upon successful completion, the return value is as follows:

Msgsnd returns a value of 0.

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

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

SEE ALSO

msgctl(2), msgget(2), signal(2), stdipc(3).

NAME
nice - change priority of a process

SYNOPSIS
int nice (incr)
int incr;

HP–UX COMPATIBILITY
Level: HP–UX/RUN ONLY

Origin: System V

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

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

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

RETURN VALUE
Upon successful completion, *nice* returns the new nice value minus 20. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

Note that *nice* assumes a user process priority value of 20. If the super–user of your system has changed the user process priority value to something less than 20, certain increments can cause *nice* to return -1, which is indistinguishable from an error return.

HARDWARE DEPENDENCIES
Integral PC:
        The super–user capabilities are provided to the normal user.

SEE ALSO
nice(1), exec(2), rtprio(2).

NAME
open - open file for reading or writing

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

HP-UX COMPATIBILITY
Level: Basic calls: HP–UX/RUN ONLY

Real time extensions: HP–UX/STANDARD - Real Time

Origin: System V, UCB, and HP

DESCRIPTION
Path points to a path name naming a file; it may not exceed 1024 bytes in length. *Open* opens a file descriptor for the named file and sets the file status flags according to the value of *oflag*. *Oflag* values are constructed by OR–ing flags from the list below.

Note that exactly one of the first three flags below **must** be used. Several of the other flags can be changed during the time the file is open using *fcntl*. See *fcntl*(2) and *fcntl*(7) for details.

mode specifies the low–order 12 bits of the files mode when the file did not previously exist and is being created by this call.

O_RDONLY
> Open for reading only.

O_WRONLY
> Open for writing only.

O_RDWR   Open for reading and writing.

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

> When opening a FIFO with O_RDONLY or O_WRONLY set:

> If O_NDELAY is set:

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

> If O_NDELAY is clear:

>> An *open* for reading–only will block until a process opens the file for writing. An *open* for writing–only will block until a process opens the file for reading.

> When opening a file associated with a communication line:

> If O_NDELAY is set:

>> The open will return without waiting for carrier.

> If O_NDELAY is clear:

>> The open will block until carrier is present.

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

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

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

The "save text image after execution", set-user id and set group–id bits of the mode is cleared. See *chmod*(2).

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

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

O_SYNCIO If a file is opened with O_SYNCIO or is set with the F_SETFL option of *fcntl*, file system writes for that file will be done through the cache to the disc as soon as possible, and the process will block until this is completed. This flag is ignored by all I/O calls except *write*, and is ignored for files other than ordinary files and block special devices on those systems which permit I/O to block special devices.

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

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

ERRORS

Open will fail and the file will not be opened if one of the following conditions is true. **Errno** will be set accordingly:

[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	O_CREAT is not set and the named file does not exist.
[EACCES]	A component of the path prefix denies search permission.
[EAGAIN]	One or more segments of a pre-existing file have been locked with *lockf* by some other process, and **O_TRUNC** is set.
[EACCES]	*Oflag* permission is denied for the named file.
[EISDIR]	The named file is a directory and *oflag* is write or read/write.
[EROFS]	The named file resides on a read-only file system and *oflag* is write or read/write.
[EMFILE]	More than the maximum number file descriptors are currently open.
[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
[ETXTBSY]	The file is open for execution and *oflag* is write or read/write. Normal executable files are only open for a short time when they start execution. Other executable file types may be kept open for a long time, or indefinitely under some circumstances. The conditions are described in HARDWARE DEPENDENCIES below.
[EFAULT]	*Path* points outside the allocated address space of the process.
[EEXIST]	O_CREAT and O_EXCL are set, and the named file exists.
[ENXIO]	O_NDELAY is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading.
[EINTR]	A signal was caught during the *open* system call.
[ENFILE]	The system file table is full.
[ENOENT]	*Path* is null.
[EINVAL]	*Oflag* specifies both O_WRONLY and O_RDWR.

RETURN VALUE

Upon successful completion, the file descriptor is returned. Otherwise, a value of -1 is returned

and *errno* is set to indicate the error.

HARDWARE DEPENDENCIES

Series 500:

Execute and write access are mutually exclusive.

Shared program files remain open for execution as long as there exists a process executing the program.

Once a shared program file with its sticky bit set has been loaded, it appears to be open indefinitely, even if the actual number of processes executing the program drops to zero. Refer to the System Administrator Manual for a discussion of the sticky bit.

Demand loaded program files that are not shared remain open until all of the code and data have been loaded. Then they are closed.

Integral PC:

_NFILE is equal to 20.

SEE ALSO

chmod(2), close(2), creat(2), dup(2), fcntl(2), lseek(2), read(2), select(2), umask(2), write(2), lockf(2).

NAME

pause - suspend process until signal

SYNOPSIS

pause ( )

HP–UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System V

DESCRIPTION

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

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

If the signal is *caught* by the calling process and control is returned from the signal–catching function (see *signal*(2)), the calling process resumes execution from the point of suspension if the signal–catching function does not specify restart of the pause (see *sigvector*(2)). The return value is -1 from *pause*, and *errno* is set to EINTR.

SEE ALSO

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

NAME

pipe - create an interprocess channel

SYNOPSIS

int pipe (fildes)
int fildes[2];

HP–UX COMPATIBILITY

Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION

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

Writes up to a system dependent maximum number bytes of data are buffered by the pipe before the writing process is blocked. HP–UX guarantees a minimum value of 4096 for this number. A read only file descriptor *fildes*[0] accesses the data written to *fildes*[1] on a first in first–out (FIFO) basis.

[EMFILE]   *Pipe* will fail if NFILE − 1 or more file descriptors are currently open.

[ENFILE]   The system file table is full.

RETURN VALUE

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

HARDWARE DEPENDENCIES

Integral PC:

Writes of up to 10240 bytes of data are buffred by the pipe before the writing process is blocked.

_NFILE is equal to 20.

SEE ALSO

sh(1), read(2), write(2), popen(3S).

NAME

plock - lock process, text, or data in memory

SYNOPSIS

#include <sys/lock.h>

int plock (op)
int op;

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Plock allows the calling process to lock the text portion of the process (text lock), its data portion (data lock), or both its text and data portion (process lock) into memory. Locked segments are immune to all routine swapping. *Plock* also allows these segments to be unlocked. The effective user ID of the calling process must be super–user, or the user must have PRIV_MLOCK (see *set-privgrp*(2)) to use this call. *Op* specifies the following:

PROCLOCK - lock text and data segments into memory (process lock)

TXTLOCK -   lock text segment into memory (text lock)

DATLOCK -   lock data segment into memory (data lock)

UNLOCK -   remove locks

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

[EPERM]   The effective user ID of the calling process is not super user and the user does not have PRIV_MLOCK.

[EINVAL]   *Op* is equal to **PROCLOCK** and a process lock, a text lock, or a data lock already exists on the calling process.

[EINVAL]   *Op* is equal to **TXTLOCK** and a text lock, or a process lock already exists on the calling process.

[EINVAL]   *Op* is equal to **DATLOCK** and a data lock, or a process lock already exists on the calling process.

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

[EINVAL]   *Op* is not equal to either **PROCLOCK, TXTLOCK, DATLOCK,** or **UNLOCK.**

All locks are released by *fork* and *exec*.

RETURN VALUE

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

HARDWARE DEPENDENCIES

Series 200 and 500:

Plock is not allowed in the [vfork,exec] window.

[EINVAL]   *Plock* not allowed in [vfork,exec] window

SEE ALSO

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

NAME

prealloc - preallocate fast disk storage

SYNOPSIS

#include <realtime.h>

int prealloc (fildes, size)
int fildes;
unsigned size;

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD – Real Time

Origin: HP

Remarks:

Not supported on the Integral Personal Computer.

DESCRIPTION

Fildes is a file descriptor obtained from a *creat, open, dup* or *fcntl* system call for an ordinary file
of zero length. *Size* is the size in bytes to be preallocated for the file specified by *fildes*, at least
size bytes will be allocated. The space will be allocated in an implementation dependent fashion
for fast sequential reads and writes. The EOF in an extended file will be left at the end of the
preallocated area. The current file pointer is left at zero. The file is zero–filled.

Using *prealloc* on a file does **not** give the file an attribute which is inherited when copying or res-
toring the file using a program like *cp*(1) or *tar*(1). It simply guarantees that the disk space has
been preallocated for *size* bytes in a manner suited for sequential access. The file can be extended
beyond these limits by *write* operations past the original end of file, however this space will be not
necessarily be allocated using any special strategy.

HARDWARE DEPENDENCIES

As the exact effect, and performance benefits, to be obtained by using this call vary with the
implementation of the file system, the performance–related details are described in the *System
Administrator Manual* for each specific machine.

ERRORS

Prealloc will fail and no disk space will be allocated if one or more of the following are true:

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

[ENOTEMPTY]   *Fildes* not associated with an ordinary file of zero length.

[ENOSPC]   Not enough space left on device to allocate the requested amount; no space
                  was allocated.

[EFBIG]   *Size* exceeds the maximum file size or the process's file size limit. See
                  ulimit(2).

RETURN VALUE

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

SEE ALSO

creat(2), dup(2), fcntl(2), open(2), read(2), ulimit(2), write(2), prealloc(1).

BUGS

The allocation of the file space is highly dependent on the current disk usage. A successful return
does not tell you how fragmented the file actually might be if the disk is reaching its capacity.

NAME

profil - execution time profile

SYNOPSIS

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

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

Remarks: Not available on Series 500 or on Integral Personal Computer.

DESCRIPTION

Buff points to an area of core whose length (in bytes) is given by *bufsiz*. After this call, the user's program counter (pc) is examined each clock tick, *offset* is subtracted from it, and the result is multiplied by *scale*. If the resulting number corresponds to a byte offset inside *buff*, that unsigned short, 16–bit counter is incremented. The number of samples per second for a given implementation is given by HZ as found in **<sys/param.h>**.

The scale is interpreted as a 16–bit unsigned, fixed–point fraction with binary point at the left: 0177777 (octal) gives a 1–1 mapping of pc's to counters in *buff*; 077777 (octal) maps each pair of instruction words together.

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

HARDWARE DEPENDENCIES

Series 500:
   Profil is not currently available.

Integral PC:
   Profil is not currently available.

RETURN VALUE

Not defined.

SEE ALSO

prof(1), monitor(3C).

NAME

ptrace - process trace

SYNOPSIS

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

HP–UX COMPATIBILITY

Level: HP-UX/DEVELOPMENT

Origin: System V

Remarks: Series 200 only. Much of the functionality of this capability is highly dependent on the underlying hardware. An application which uses this intrinsic should not be expected to be portable across architectures or implementations.

DESCRIPTION

Ptrace provides a means by which a parent process may control the execution of a child process. Its primary use is for the implementation of breakpoint debugging; see *adb*(1). The child process behaves normally until it encounters a signal (see *signal*(2) for the list), at which time it enters a stopped state and its parent is notified via *wait*(2). When the child is in the stopped state, its parent can examine and modify its "core image" using *ptrace*. Also, the parent can cause the child either to terminate or continue, with the possibility of ignoring the signal that caused it to stop.

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

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

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

1, 2 With these requests, the word at location *addr* in the address space of the child is returned to the parent process. If instruction (I) and data (D) space are separated, request **1** returns a word from I space, and request **2** returns a word from D space. If I and D space are not separated, either request **1** or request **2** may be used with equal results. The *data* argument is ignored. These two requests will fail if *addr* is not the start address of a word, in which case a value of -1 is returned to the parent process and the parent's *errno* is set to EIO.

3   With this request, the word at location *addr* in the child's USER area in the system's address space (see <**sys/user.h**>) is returned to the parent process. Addresses in this area are system dependent, but start at zero. The limit can be derived from <**sys/user.h**>. The *data* argument is ignored. This request will fail if *addr* is not the start address of a word or is outside the USER area, in which case a value of -1 is returned to the parent process and the parent's *errno* is set to EIO.

4, 5 With these requests, the value given by the *data* argument is written into the address space of the child at location *addr*. Request **4** writes a word into I space, and request **5** writes a word into D space. Upon successful completion, the value written into the address space of the child is returned to the parent. These two requests will fail if *addr* is not the starting address of a word, or if *addr* is a location in a pure procedure space and either another process is executing in that space or the parent process does not have write access for the executable file corresponding

to that space. Upon failure a value of -1 is returned to the parent process and the parent's *errno* is set to EIO.

6 With this request, a few entries in the child's USER area can be written. *Data* gives the value that is to be written and *addr* is the location of the entry. The few entries that can be written are dependent on the architecture of the system, but include the user data registers, auxiliary data registers, and status register (the set of registers, or bits in registers, which the user's program could modify).

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

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

9 This request causes a flag to be set so that an interrupt will occur upon the completion of one machine instruction, and then executes the same steps as listed above for request **7**. If the processor does not provide a trace bit, this request returns an error. This effectively allows single stepping of the child.

Whether or not the trace bit remains set after this interrupt is a function of the hardware.

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

GENERAL ERRORS
Ptrace will in general fail if one or more of the following are true:

[EIO]   *Request* is an illegal number.

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

HARDWARE DEPENDENCIES
Series 200 only; not supported on Series 500.

SEE ALSO
adb(1), exec(2), signal(2), wait(2).

NAME
read, readv - read input

SYNOPSIS
int read (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;

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

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

HP-UX COMPATIBILITY
Level: read: HP-UX/RUN ONLY

             readv: HP-UX/STANDARD

Origin: System V, UCB, and HP

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

Read attempts to read *nbyte* bytes from the file associated with *fildes* into the buffer pointed to by *buf*. *Readv* performs the same action but scatters the input data into the *iovcnt* buffers specified by the elements of the *iovec* array: iov[0], iov[1], ..., iov[*iovcnt* − 1].

For *readv* the iovec structure is defined as

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

Each *iovec* entry specifies the base address and length of an area in memory where data should be placed. *Readv* will always fill one area completely before proceeding to the next area. The *iovec* array may be at most *MAXIOV* long.

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

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

Upon successful completion, *read* returns the number of bytes actually read and placed in the buffer; this number may be less than *nbyte* if 1) the file is associated with a communication line (see *ioctl*(2) and *termio*(4)), or 2) if the number of bytes left in the file is less than *nbyte* bytes. A value of 0 is returned when an end-of-file has been reached.

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

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

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

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

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

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

ERRORS

Read will fail if one of the following conditions is true and errno will be set accordingly:

[EBADF]   *Fildes* is not a valid file descriptor open for reading.

[EFAULT]   *Buf* points outside the allocated address space.

[EINTR]   A signal was caught during the *read* system call.

[EDEADLK]   A resource deadlock would occur as a result of this operation (see *lockf*(2)).

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

[EINVAL]   *Iovcnt* was less than or equal to 0, or greater then *MAXIOV*.

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

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

RETURN VALUE

Upon successful completion a non–negative integer is returned indicating the number of bytes actually read. Otherwise, a -1 is returned and *errno* is set to indicate the error.

CAVEATS

Record locking may or may not be enforced by the system depending on the setting of the file's mode bits (see *lockf*(2)).

BUGS

The character special devices, and raw discs in particular, apply constraints on how *read* can be used. See the specific Section 4 entries for details on particular devices.

HARDWARE DEPENDENCIES

Series 500:

> *Readv* is not currently supported on the Series 500.

Integral PC:

> Information read from a disc by the operating system is cached in memory to speed up access to information in files. Consequently, not every read operation causes the system to access the physical medium. If it becomes necessary to access the physical medium, you should execute *sync* before executing *read*.

SEE ALSO

creat(2), dup(2), fcntl(2), ioctl(2), open(2), pipe(2), select(2), ustat(2), tty(4), lockf(2).

NAME

    reboot - boot the system

SYNOPSIS

    int reboot (howto, device_file, filename)

    int howto; char *device_file; char *filename;

HP-UX COMPATIBILITY

    Level: HP-UX/NON STANDARD

    Origin: UCB

    Remarks: *Reboot* is implemented on the Series 200 only.

DESCRIPTION

    Reboot causes the system to be rebooted. *Howto* is a mask of reboot options (see **<sys/reboot.h>** Only RB_HALT, RB_AUTOBOOT, RB_NOSYNC, RB_NEWDEVICE, and RB_NEWFILE are recognized options.

    The *howto* options are:

        RB_AUTOBOOT

            a filesystem sync is performed (unless RB_NOSYNC is set) and the processor is rebooted from the device and file from which it was previously booted.

        RB_HALT

            the processor is simply halted. A sync of the filesystem will be done unless the RB_NOSYNC flag is set. RB_HALT should be used with caution.

        RB_NOSYNC

            a sync of the filesystem is not to be performed.

        RB_NEWDEVICE

            the *device_file* argument to the system call is to be used as the filename of the device from which to reboot.

        RB_NEWFILE

            the *filename* argument to the system call is to be used as the name of the file to be rebooted.

    Device_file specifies the device from which the reboot is to take place. *Device_file* must be a block or character special file name and is used only if the RB_NEWDEVICE option is set.

    Filename specifies the name of the file to be rebooted (only used if the RB_NEWFILE option is set). This file will be loaded into memory by the bootstrap and control passed to it. *Filename* must be one of the files listed by the boot rom at power up.

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

RETURN VALUES

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

NAME

rmdir - remove a directory file

SYNOPSIS

rmdir(path)
char *path;

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

Remarks: Not all systems implement this as a system call; some use a library call to the *rmdir*(1) command to achieve the same effect. The errors documented below will appear in any case, and no error messages will ever be printed.

DESCRIPTION

Rmdir removes a directory file whose name is given by *path*. The directory must be empty before it can be removed.

RETURN VALUE

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

ERRORS

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

[ENOTEMPTY]
              The named directory is not empty (it contains files named other than "." and "..").

[ENOENT]   The pathname was too long.

[ENOTDIR]   A component of the path prefix is not a directory, or the named file is not a directory.

[ENOENT]   The named file does not exist.

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

[EACCES]   Write permission is denied on the directory containing the link to be removed.

[EBUSY]   The directory to be removed is the mount point for a mounted file system.

[EROFS]   The directory entry to be removed resides on a read-only file system.

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

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

[EINVAL]   "." and ".." are not allowed as directory names.

HARDWARE DEPENDENCIES

Series 500:
              The directory identifiers "." and ".." are recognized by Series 500 HP-UX, but files of the same names do not appear in the directory structure.

Series 200:
              An empty directory contains two files named "." and "..", respectively (*ls* -a option is used to display them).

SEE ALSO

mkdir(2), unlink(2)

NAME

rtprio - change or read realtime priority

SYNOPSIS

#include <sys/rtprio.h>

int rtprio (pid, prio)
int pid, prio;

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD – Real Time

Origin: HP

DESCRIPTION

Rtprio is used to set or read the realtime priority of a process. If *pid* is zero, it names the calling process; otherwise it gives the pid of the process. When setting the realtime priority of another process, the real or effective user ID of the calling process must match the real or effective user ID of the process to be modified, or the effective user ID of the calling process must be that of super–user. The calling process must also be a member of or have an effective group id of a group having PRIV_RTPRIO access to be a realtime process (see **getprivgrp(2)**) or be super user. Simply reading realtime priorities requires no special privilege.

Real time scheduling policies differ from the normal timesharing policies in that the realtime priority is used to absolutely order all realtime processes; this priority is not degraded over time. All realtime processes are of higher priority than normal user processes, although some system processes may run at realtime priorities themselves. If there are several eligible processes at the same priority level, they will be run in a round robin fashion as long as no process with higher priority intercedes. A realtime process will receive cpu service until it either voluntarily gives up the cpu or is preempted by a process of equal or higher priority. Interrupts may also preempt a realtime process.

Valid realtime priorities run from zero to 127. Zero is the highest (most important) priority. This realtime priority is inherited across *forks* and *execs*.

Prio specifies the following:

0-127	Set process to this realtime priority.
RTPRIO_NOCHG	
	Do not change realtime priority. This is used for reading the process realtime priority.
RTPRIO_RTOFF	
	Set this process to no longer have a realtime priority. It will resume a normal timesharing priority. Any process, regardless of privilege, is allowed to turn off its own realtime priority using a *pid* of zero.

RETURN VALUE

If no error occurs, *rtprio* will return the *pid*'s former (before the call) realtime priority. If the process was not a realtime process, the value RTPRIO_RTOFF will be returned. If an error does occur, –1 is returned and *errno* is set to one of the values described in the ERRORS section.

ERRORS

[EINVAL]	*Prio* is not RTPRIO_NOCHG, RTPRIO_RTOFF or in the range 0 to 127.
[ESRCH]	No process can be found corresponding to that specified by *pid*.
[EPERM]	The calling process is not the super-user and neither its real or effective user-id match the real or effective user-id of the process indicated by *pid*.
[EPERM]	The calling process is not a member of a group having PRIV_RTPRIO capability and *prio* is not RTPRIO_NOCHG, or RTPRIO_RTOFF with a *pid* of zero.

HARDWARE DEPENDENCIES

Series 500:

Some of the work done by the system on behalf of users is done with daemon processes which have varioius priorities. Some functions such as copying user space on a fork, virtual memory swapping, and LAN activity are done at a priority lower than any of the *rtprio*(2) priorities.

Other functions, such as terminal I/O, disc I/O, DIL interrupts, signals, *select*(2) wake-ups, and system timers, are done at a priority level equivalent to an *rtprio*(2) priority of 64.

If there is a real-time process that is consuming all available CPU time, the system will not be able to accomplish any other system activities that have a lower priority, even if the function is on behalf of the real-time process. In the case of multi-CPU systems, it will take multiple real-time processes to lock out the system.

The user of *rtprio*(2) must decide whether the task requiring real-time priorities needs to have an effective priority greater than or less than the system functions provided.

SEE ALSO

rtprio(1), getprivgroup(2), plock(2),nice(2).

NOTES

Normally, compute bound programs should not be run at realtime priorities, because all time sharing work on the cpu would come to a complete halt.

NAME
select - synchronous I/O multiplexing

SYNOPSIS
#include <time.h>

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

HP–UX COMPATIBILITY
Level: HP–UX/STANDARD – Real Time

Origin: HP and UCB

DESCRIPTION
Select examines the file descriptors specified by the bit masks *readfds*, *writefds* and *exceptfds*. The bits from 0 through *nfds*–1 are examined. File descriptor *f* is represented by the bit 1<<f in the masks. More formally, a file descriptor is represented by:

fds[(f / BITS_PER_INT)] & (1 << (f % BITS_PER_INT))

When *select* completes successfully it returns the three bit masks modified as follows: For each file descriptor less than *nfds*, the corresponding bit in each mask is set if the bit was set upon entry and certain conditions prevail. The bit is set if the file descriptor is ready for reading, writing or has an exceptional condition pending.

If *timeout* is a non–zero pointer, it specifies a maximum interval to wait for the selection to complete. If *timeout* is a zero pointer, the select waits until an event causes one of the masks to be returned with a valid (non–zero) value. To poll, the *timeout* argument should be non–zero, pointing to a zero valued timeval structure. Specific implementations may place limitations on the maximum timeout interval supported. The constant *MAX_ALARM* defined in *<sys/param.h>* specifies the implementation–specific maximum (in seconds). Whenever *timeout* specifies a value greater than this maximum, it is silently rounded down to this maximum. On all implementations, *MAX_ALARM* is guaranteed to be at least 31 days (in seconds). Note that the use of a timeout does not affect any pending timers set up by *alarm(2)* or *setitimer(2)*.

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

ERRORS
An error return from *select* indicates:

[EBADF]   One or more of the bit masks specified an invalid descriptor.

[EINTR]   A signal was delivered before any of the selected for events occurred or before the time limit expired.

[EFAULT]   One or more of the pointers was invalid.

[EINVAL]   The *timeout* value specified a time outside of the acceptable range.

RETURN VALUE
Select returns the number of descriptors contained in the bit masks, or -1 if an error occurred. If the time limit expires then *select* returns 0 and all the masks are cleared.

HARDWARE DEPENDENCIES
Series 500:

Select(2) supports the following devices and file types:

pipes
fifo special files (named pipes)
Model 520 Internal Terminal Emulator (ITE)

HP 98700H ITE and HP–HIL input devices
            (such as HP 46020A Keyboard and HP 46086A Button Box)
HP 27128A ASI tty driver
ι HP 27140A Modem MUX tty driver
pty(4) special files
HP 27125A LAN interface card driver (LLA)
ν HP 27130A/B port MUX (with appropriate firmware revision)

Ordinary files always select true whenever selecting on reads, writes, and/or exceptions.

The convention for device files that do not support *select*(2) is to always return true for those conditions the user is selecting on.

Series 200:
            Select(2) supports the following devices and file types:

            pipes
            fifo special files (named pipes)
            Model 520 Internal Terminal Emulator (ITE)
            HP 98643 LAN interface card driver
            All serial interfaces
            pty(4) special files
            All ITEs and HP–HIL input devices

Ordinary files always select true whenever selecting on reads, writes, and/or exceptions.

File types not supporting *select*(2) always return true.

SEE ALSO
            fcntl(2), read(2), write(2).

NAME
semctl - semaphore control operations

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

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

HP-UX COMPATIBILITY
Level: HP-UX/STANDARD

Origin: System V

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

The following *cmds* are executed with respect to the semaphore specified by *semid* and *semnum:*

GETVAL	Return the value of semval (see *glossary(9)*). {READ}
SETVAL	Set the value of semval to *arg.val.* {ALTER} When this cmd is success-fully executed, the semadj value corresponding to the specified semaphore in all processes is cleared.
GETPID	Return the value of sempid. {READ}
GETNCNT	Return the value of semncnt. {READ}
GETZCNT	Return the value of semzcnt. {READ}

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

GETALL	Place semvals into array pointed to by *arg.array*. {READ}
SETALL	Set semvals according to the array pointed to by *arg.array*. {ALTER} When this cmd is successfully executed the semadj values corresponding to each specified semaphore in all processes are cleared.

The following *cmds* are also available:

IPC_STAT	Place the current value of each member of the data structure associated with *semid* into the structure pointed to by *arg.buf*. The contents of this structure are defined in *glossary(9)*. {READ}
IPC_SET	Set the value of the following members of the data structure associated with *semid* to the corresponding value found in the structure pointed to by *arg.buf:*

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

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

IPC_RMID   Remove the semaphore identifier specified by *semid* from the system and destroy the set of semaphores and data structure associated with it. This cmd can only be executed by a process that has an effective user ID equal to either that of super–user or to the value of either **sem_perm.uid** or **sem_perm.cuid** in the data structure associated with *semid*.

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

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

[EINVAL]   *Semnum* is less than zero or greater than **sem_nsems**.

[EINVAL]   *Cmd* is not a valid command.

[EACCES]   Operation permission is denied to the calling process (see *glossary(9)*).

[ERANGE]   *Cmd* is **SETVAL** or **SETALL** and the value to which semval is to be set is greater than the system imposed maximum.

[EPERM]   *Cmd* is equal to **IPC_RMID** or **IPC_SET** and the effective user ID of the calling process is not equal to that of super–user and it is not equal to the value of either **sem_perm.uid** or **sem_perm.cuid** in the data structure associated with *semid*.

[EFAULT]   *Arg.buf* or *arg.array* points to an illegal address.

RETURN VALUE

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

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

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

SEE ALSO

semget(2), semop(2), stdipc(3).

NAME
semget - get set of semaphores

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

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

HP-UX COMPATIBILITY
Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION
Semget returns the semaphore identifier associated with *key*.

A semaphore identifier and associated data structure and set containing *nsems* semaphores (see *glossary*(9)) are created for *key* if one of the following are true:

    Key is equal to **IPC_PRIVATE**.

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

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

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

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

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

    Sem_otime is set equal to 0 and **sem_ctime** is set equal to the current time.

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

[EINVAL]	*Nsems* is either less than or equal to zero or greater than the system-imposed limit.
[EACCES]	A semaphore identifier exists for *key*, but operation permission (see *glossary*(9)) as specified by the low-order 9 bits of *semflg* would not be granted.
[EINVAL]	A semaphore identifier exists for *key*, but the number of semaphores in the set associated with it is less than *nsems* and *nsems* is not equal to zero.
[ENOENT]	A semaphore identifier does not exist for *key* and (*semflg* & **IPC_CREAT**) is "false".
[ENOSPC]	A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphore identifiers system wide would be exceeded.
[ENOSPC]	A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores system wide would be exceeded.
[EEXIST]	A semaphore identifier exists for *key* but ((*semflg* & IPC_CREAT) **&** (*semflg* & **IPC_EXCL**)) is "true".

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

SEE ALSO
    semctl(2), semop(2), stdipc(3).

NAME
semop - semaphore operations

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

    int semop (semid, sops, nsops)
    int semid;
    struct sembuf *sops;
    int nsops;

HP–UX COMPATIBILITY
Level: HP–UX/STANDARD

Origin: System V

DESCRIPTION
Semop is used to atomically perform an array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by *semid*. *Sops* is a pointer to the array of semaphore–operation structures. *Nsops* is the number of such structures in the array. The contents of each structure includes the following members:

    ushort   sem_num;   /* semaphore number */
    short   sem_op;   /* semaphore operation */
    short   sem_flg;   /* operation flags */

Each semaphore operation specified by *sem_op* is performed on the corresponding semaphore specified by *semid* and *sem_num*. Semaphore array operations are atomic, in that none of the semaphore operations will be performed until blocking conditions on all of the semaphores in the array have been removed.

Sem_op specifies one of three semaphore operations as follows:

If *sem_op* is a negative integer, one of the following will occur: {ALTER}

If semval (see *glossary(9)*) is greater than or equal to the absolute value of *sem_op*, the absolute value of *sem_op* is subtracted from semval. Also, if (*sem_flg* & **SEM_UNDO**) is "true", the absolute value of *sem_op* is added to the calling process's semadj value (see *exit(2)*) for the specified semaphore.

If semval is less than the absolute value of *sem_op* and (*sem_flg* & **IPC_NOWAIT**) is "true", *semop* will return immediately.

If semval is less than the absolute value of *sem_op* and (*sem_flg* & **IPC_NOWAIT**) is "false", *semop* will increment the semncnt associated with the specified semaphore and suspend execution of the calling process until one of the following conditions occur.

Semval becomes greater than or equal to the absolute value of *sem_op*. When this occurs, the value of semncnt associated with the specified semaphore is decremented, the absolute value of *sem_op* is subtracted from semval and, if (*sem_flg* & **SEM_UNDO**) is "true", the absolute value of *sem_op* is added to the calling process's semadj value for the specified semaphore.

The semid for which the calling process is awaiting action is removed from the system (see *semctl(2)*). When this occurs, *errno* is set equal to EIDRM, and a value of -1 is returned.

The calling process receives a signal that is to be caught and the signal handler does not specify that the call be restarted (see *sigvector(2)*). When

this occurs, the value of semncnt associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in *signal*(2).

If *sem_op* is a positive integer, the value of *sem_op* is added to semval and, if (*sem_flg* & **SEM_UNDO**) is "true", the value of *sem_op* is subtracted from the calling process's semadj value for the specified semaphore. {ALTER}

If *sem_op* is zero, one of the following will occur: {READ}

If semval is zero, *semop* will proceed to the next semaphore operation specified by *sops*, or return immediately if this is the last operation.

If semval is not equal to zero and (*sem_flg* & **IPC_NOWAIT**) is "true", *semop* will return immediately.

If semval is not equal to zero and (*sem_flg* & **IPC_NOWAIT**) is "false", *semop* will increment the semzcnt associated with the specified semaphore and suspend execution of the calling process until one of the following occurs:

Semval becomes zero, at which time the value of semzcnt associated with the specified semaphore is decremented.

The semid for which the calling process is awaiting action is removed from the system. When this occurs, *errno* is set equal to EIDRM, and a value of -1 is returned.

The calling process receives a signal that is to be caught and the signal handler does not specify that the call be restarted (see *sigvector*(2)). When this occurs, the value of semzcnt associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in *signal*(2).

Semop will fail if one or more of the following are true for any of the semaphore operations specified by *sops*:

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

[EFBIG]   *Sem_num* is less than zero or greater than or equal to the number of semaphores in the set associated with *semid*.

[E2BIG]   *Nsops* is greater than the system-imposed maximum.

[EACCES]   Operation permission is denied to the calling process (see *glossary(9)*).

[EAGAIN]   The operation would result in suspension of the calling process but (*sem_flg* & **IPC_NOWAIT**) is "true".

[ENOSPC]   The limit on the number of individual processes requesting an **SEM_UNDO** would be exceeded.

[EINVAL]   The number of individual semaphores for which the calling process requests a **SEM_UNDO** would exceed the limit.

[ERANGE]   An operation would cause a semval to overflow the system-imposed limit.

[ERANGE]   An operation would cause a semadj value to overflow the system-imposed limit.

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

Upon successful completion, the value of sempid for each semaphore specified in the array pointed to by *sops* is set equal to the process ID of the calling process. The value of Sem_otime in the data structure associated with the semaphore identifier will be set to the current time.

RETURN VALUE

If *semop* returns due to the receipt of a signal, a value of -1 is returned to the calling process and *errno* is set to EINTR. If it returns due to the removal of a *semid* from the system, a value of -1 is returned and *errno* is set to EIDRM.

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

SEE ALSO

exec(2), exit(2), fork(2), semctl(2), semget(2), stdipc(3).

NAME

setgroups - set group access list

SYNOPSIS

#include <sys/param.h>

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

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: UCB

Remarks: Not available on Series 500.

DESCRIPTION

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

Only the super–user may set new groups by adding to the group access list of the current user process; any user may delete groups from it.

RETURN VALUE

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

ERRORS

The *setgroups* call will fail if:

[EPERM]   The caller is not the super–user and has attempted to set new groups.

[EFAULT]   The address specified for *gidset* is outside the process address space.

[EINVAL]   *ngroups* is greater than NGROUPS or not positive.

[EINVAL]   An entry in *gidset* is not a valid group ID.

SEE ALSO

getgroups(2), initgroups(3C)

NAME
sethostname - set name of host cpu

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

HP–UX COMPATIBILITY
Level: HP–UX/RUN ONLY

Origin: UCB

DESCRIPTION
This call sets the name of the host processor to be *name*, which has a length of *namelen* charac-
ters. The maximum value of *namelen* is determined by the *uname* structure. This is normally
executed when the system is bootstrapped, executed out of the file **/etc/rc.** This intrinsic sets the
nodename field in the *utsname* structure returned by *uname*(2).

Sethostname will fail and return an error if:

[EPERM]   It is not executed by the super–user.

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

HARDWARE DEPENDENCIES
Integral PC:
         The super–user capabilities are provided to the normal user.

SEE ALSO
hostname(1), uname(1), gethostname(2), uname(2).

NAME
    setpgrp - set process group ID

SYNOPSIS
    int setpgrp ( )

HP–UX COMPATIBILITY
    Level: HP–UX/RUN ONLY

    Origin: System V

DESCRIPTION
    Setpgrp sets the process group ID of the calling process to the process ID of the calling process and returns the new process group ID.

    Setpgrp breaks the calling process's terminal affiliation unless it is already the process group leader. See *termio*(4).

RETURN VALUE
    Setpgrp returns the value of the new process group ID.

SEE ALSO
    exec(2), fork(2), getpid(2), kill(2), signal(2).

NAME

setuid, setgid - set user and group IDs

SYNOPSIS

int setuid (uid)
int uid;

int setgid (gid)
int gid;

HP–UX COMPATIBILITY

Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION

Setuid sets the real, effective, and/or saved user ID of the calling process (*"ruid"*, *"euid"*, and *"suid"*, respectively).

If *uid* is not equal to the super–user's ID, and is equal to *ruid*, then *setuid* sets *euid* to *uid*.

Otherwise, if *uid* is not equal to the super–user's ID, and is equal to *euid*, then *setuid* sets *ruid* to *uid*.

Otherwise, if *uid* is not equal to the super–user's ID, and is equal to *suid*, then *setuid* sets *euid* to *uid*.

Otherwise, if *euid* is equal to the super–user's user ID then *setuid* sets *ruid*, *euid*, and *suid* to *uid*.

Setgid sets the real, effective, and/or saved group ID of the calling process (*"rgid"*, *"egid"*, and *"sgid"*, respectively).

If *gid* is equal to *rgid* then *setgid* sets *egid* to *gid*.

Otherwise, if *gid* is equal to *egid* then *setgid* sets *rgid* to *gid*.

Otherwise, if *gid* is equal to *sgid* then *setgid* sets *egid* to *gid*.

Otherwise, if *euid* is equal to the super–user's user ID then *setgid* sets *rgid*, *egid*, and *sgid* to *gid*.

Setuid and *setgid* will fail and return -1 if:

[EPERM]   None of the conditions above are met.

[EINVAL]   *Uid* (*gid*) is not a valid user (group) ID.

RETURN VALUE

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

HARDWARE DEPENDENCIES

The Integral PC does not currently support saved user IDs or saved group IDs.

SEE ALSO

exec(2), getuid(2), setgroups(2).

NAME

shmctl - shared memory control operations

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl (shmid, cmd, buf)
int shmid, cmd;
struct shmid_ds *buf;

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Shmctl provides a variety of shared memory control operations as specified by *cmd*. The following *cmd*s are available:

IPC_STAT   Place the current value of each member of the data structure associated with *shmid* into the structure pointed to by *buf*. The contents of this structure are defined in *glossary*(9). {READ}

IPC_SET   Set the value of the following members of the data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:
shm_perm.uid
shm_perm.gid
shm_perm.mode /* only low 9 bits */

This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user or to the value of either **shm_perm.uid** or **shm_perm.cuid** in the data structure associated with *shmid*.

IPC_RMID   Remove the shared memory identifier specified by *shmid* from the system and destroy the shared memory segment and data structure associated with it. This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user or to the value of either **shm_perm.uid** or **shm_perm.cuid** in the data structure associated with *shmid*.

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

[EINVAL]   *Shmid* is not a valid shared memory identifier.

[EINVAL]   *Cmd* is not a valid command.

[EACCES]   *Cmd* is equal to **IPC_STAT** and {READ} operation permission is denied to the calling process (see *glossary*(9)).

[EPERM]   *Cmd* is equal to **IPC_RMID** or **IPC_SET** and the effective user ID of the calling process is not equal to that of super-user, nor is it equal to the value of **shm_perm.uid** or **shm_perm.cuid** in the data structure associated with *shmid*.

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

RETURN VALUE

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

HARDWARE DEPENDENCIES

Series 200 and 500:

Two additional shared memory control operations as specified by *cmd* are available:

SHM_LOCK   Lock the shared memory segment specified by *shmid* in memory. This *cmd* can only be executed by a process that either has an effective user ID equal to super–user or has PRIV_MLOCK privilege (see *setprivgrp*(2)).

SHM_UNLOCK
             Unlock the shared memory segment specified by *shmid*. This *cmd* can only be executed by a process that either has an effective user ID equal to super–user or has PRIV_MLOCK privilege (see *setprivgrp*(2)).

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

[EPERM]   *Cmd* is equal to **SHM_LOCK** or **SHM_UNLOCK** and the effective user ID of the calling process is not equal to that of super–user and the calling process does not have PRIV_MLOCK privilege (see *setprivgrp*(2)).

[EINVAL]   *Cmd* is equal to **SHM_UNLOCK** and the shared–memory segment specified by *shmid* is not locked in memory.

SEE ALSO
        shmget(2), shmop(2), stdipc(3).

NAME

shmget - get shared memory segment

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget (key, size, shmflg)
key__t key;
int size, shmflg;

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

DESCRIPTION

Shmget returns the shared memory identifier associated with *key*.

A shared memory identifier and associated data structure and shared memory segment of size *size* bytes (see *glossary*(9)) are created for *key* if one of the following are true:

> *Key* is equal to **IPC__PRIVATE**.

> *Key* does not already have a shared memory identifier associated with it, and (*shmflg* & **IPC__CREAT**) is "true".

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

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

> The low–order 9 bits of **shm__perm.mode** are set equal to the low–order 9 bits of *shmflg*. **Shm__segsz** is set equal to the value of *size*.

> **Shm__lpid**, **shm__nattch**, **shm__atime**, and **shm__dtime** are set equal to 0.

> **Shm__ctime** is set equal to the current time.

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

[EINVAL]	*Size* is less than the system–imposed minimum or greater than the system–imposed maximum.
[EACCES]	A shared memory identifier exists for *key* but operation permission (see *glossary*(9)) as specified by the low–order 9 bits of *shmflg* would not be granted.
[EINVAL]	A shared memory identifier exists for *key* but the size of the segment associated with it is less than *size* and *size* is not equal to zero.
[ENOENT]	A shared memory identifier does not exist for *key* and (*shmflg* & **IPC__CREAT**) is "false".
[ENOSPC]	A shared memory identifier is to be created but the system–imposed limit on the maximum number of allowed shared memory identifiers system wide would be exceeded.
[ENOMEM]	A shared memory identifier and associated shared memory segment are to be created but the amount of available physical memory is not sufficient to fill the request.
[EEXIST]	A shared memory identifier exists for *key* but ((*shmflg* & IPC__CREAT) && (*shmflg* & **IPC__EXCL**)) is "true".

HARDWARE DEPENDENCIES
Series 500:

Shared memory segments larger than 16384 bytes are virtual paged segments; otherwise they are virtual non-paged segments.

RETURN VALUE
Upon successful completion, a non-negative integer, namely a shared memory identifier is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

SEE ALSO
shmctl(2), shmop(2), stdipc(3).

NAME

      shmop - shared memory operations

SYNOPSIS

      #include <sys/types.h>
      #include <sys/ipc.h>
      #include <sys/shm.h>

      char *shmat (shmid, shmaddr, shmflg)
      int shmid;
      char *shmaddr
      int shmflg;

      int shmdt (shmaddr)
      char *shmaddr

HP–UX COMPATABILITY

      Level: HP–UX/STANDARD

      Origin: System V Release 2

DESCRIPTION

      Shmat attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the data segment of the calling process. If the shared memory segment has not already been attached *shmaddr* must be specified as zero, and the segment will be attached at a location selected by the operating system. That location will be the same in all processes accessing that shared memory object. If the shared memory segment has already been attached a non-zero value of *shmaddr* will be accepted as long as the specified address is the same as the current attach address of the segment. Some implementations may permit the specification of a non–zero value as a machine dependent extension, as discussed in HARDWARE DEPENDENCIES below. Systems which do this do not necessarily guarantee that a given shared memory object will appear at the same address in all processes which access it, unless the user specifies an address.

      The segment is attached for reading if (*shmflg* & **SHM_RDONLY**) is "true" {READ}, otherwise it is attached for reading and writing {READ/WRITE}. It is not possible to attach a segment for write only.

      Shmat will fail and not attach the shared memory segment if one or more of the following are true:

      [EINVAL]   *Shmid* is not a valid shared memory identifier.

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

      [ENOMEM]   The available data space is not large enough to accommodate the shared memory segment.

      [EINVAL]   *Shmaddr* is not zero and the machine does not permit non zero values or *shmaddr* is not equal to the current attach location for the shared memory segment.

      [EMFILE]   The number of shared memory segments attached to the calling process would exceed the system–imposed limit.

      Shmdt detaches from the calling process's data segment the shared memory segment located at the address specified by *shmaddr*.

      Shmdt fails if the following is true.

      [EINVAL]   *Shmdt* will fail and not detach the shared memory segment if *shmaddr* is not the data segment start address of a shared memory segment.

HARDWARE DEPENDENCIES

      Series 500:

Shmaddr must be zero in all cases for *shmat*. Otherwise, an error is generated. In addition, **SHM_RDONLY** is not supported, and if it is set in *shmflg*, an error is generated.

[EINVAL]   *Shmflg* has **SHM_RDONLY** set.

Series 200:

Shmaddr may be non–zero. If it is, the segment is attached at the address specified by one of the following criteria:

> If *shmaddr* is equal to zero, the segment is attached at the first available address as selected by the system. The selected value will vary for each process accessing that shared memory object.

> If *shmaddr* is not equal to zero and (*shmflg* & **SHM_RND**) is "true", the segment is attached at the address given by (*shmaddr* – (*shmaddr* modulus **SHMLBA**)).

> If *shmaddr* is not equal to zero and (*shmflg* & **SHM_RND**) is "false", the segment is attached at the address given by *shmaddr*.

Shmat will fail and not attach the shared memory segment if one or more of the following are true:

[EINVAL]   *Shmaddr* is not equal to zero, and the value of (*shmaddr* – (*shmaddr* modulus **SHMLBA**)) is an illegal address.

[EINVAL]   *Shmaddr* is not equal to zero, (*shmflg* & **SHM_RND**) is "false", and the value of *shmaddr* is an illegal address.

RETURN VALUES

Upon successful completion, the return value is as follows:

> *Shmat* returns the data segment start address of the attached shared memory segment.

> *Shmdt* returns a value of 0.

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

SEE ALSO

exec(2), exit(2), fork(2), shmctl(2), shmget(2), stdipc(3).

NAME

sigblock - block signals

SYNOPSIS

long sigblock(mask);
long mask;

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: UCB

DESCRIPTION

Sigblock causes the signals specified in *mask* to be added to the set of signals currently being blocked from delivery. Signal i is blocked if the i-th bit in *mask* is a 1 (that is, if (mask & (1L $<<$ (i–1))) != 0).

It is not possible to block those signals which cannot be ignored, as documented in *signal*(2); this restriction is silently imposed by the system.

Sigsetmask(2) can be used to set the mask absolutely.

RETURN VALUE

The previous set of masked signals is returned.

SEE ALSO

kill(2), sigvector(2), sigsetmask(2), sigpause(2)

NAME
signal - specify what to do upon receipt of a signal

SYNOPSIS
#include <signal.h>

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

func(sig [, code, scp])
int sig, code;
struct sigcontext *scp;

HP-UX COMPATIBILITY
Level: HP-UX/RUN ONLY

Origin: System V, HP

DESCRIPTION
Signal allows the calling process to choose one of three ways in which it is possible to handle the receipt of a specific signal. *Sig* specifies the signal and *func* specifies the choice.

Sig can be assigned any one of the following except **SIGKILL**:

SIGHUP	01	hangup
SIGINT	02	interrupt
SIGQUIT	03*	quit
SIGILL	04*•	illegal instruction
SIGTRAP	05*•	trace trap
SIGIOT	06*	software generated (sent by *abort*(3C))
SIGEMT	07*	software generated
SIGFPE	08*	floating point exception
SIGKILL	09‡+	kill
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGUSR1	16	user defined signal 1
SIGUSR2	17	user defined signal 2
SIGCLD	18†	death of a child (see details below)
SIGPWR	19•†	power fail (see details below and hardware dependencies)
SIGVTALRM	20	virtual timer alarm; see *getitimer*
SIGPROF	21	profiling timer alarm; see *getitimer*
SIGIO	22	Reserved for future use
SIGWINDOW	23†	A window change or mouse signal; see the windowing package

*   Indicates that a core dump may be generated.

†   Indicates that the action on **SIG_DFL** is to ignore the signal, rather than terminate the process.

•   Indicates that the signal is not reset when it is caught by *signal*.

‡   Indicates that the signal cannot be ignored.

+   Indicates that the signal cannot be caught.

Details follow.

Func is assigned one of three values: **SIG_DFL**, **SIG_IGN**, or a *function address*. The actions prescribed by these values are as follows:

SIG_DFL - (usually) terminate process upon receipt of a signal.

    For those signals not flagged with a dagger (†) above, upon receipt of the signal *sig*, the receiving process is to be terminated with all of the consequences outlined in *exit*(2). In addition a "core image" will be made in the current working directory of the receiving process if *sig* is one for which an asterisk appears in the above list *and* the following conditions are met:

        The effective user ID and the real user ID of the receiving process are equal.

        An ordinary file named **core** exists and is writable or can be created. If the file must be created, it will have the following properties:

            a mode of 0666 modified by the file creation mask (see *umask*(2))

            a file owner ID that is the same as the effective user ID of the receiving process

            a file group ID that is the same as the effective group ID of the receiving process

The semantics of those signals that are flagged with a dagger are discussed below.

SIG_IGN - ignore signal

    The signal *sig* is to be ignored.

    When *signal(2) is called with func* = **SIG_IGN**, and a *sig* signal is pending, the pending signal is cleared (except for a pending **SIGKILL** signal).

    Note: the signal **SIGKILL** cannot be ignored.

function address - catch signal

    Upon receipt of the signal *sig*, the receiving process is to execute the signal-catching function pointed to by *func*. The signal number *sig* will be passed as the first parameter to the signal-catching function. The HP-UX kernel will also pass two additional (optional) parameters to signal handler routines. The complete parameter list for *func* is:

    sig   signal number

    code   a word of information usually provided by the hardware.

    scp   a pointer to the machine dependent structure *sigcontext* defined in the include file signal.h.

    Depending on the value of *sig*, *code* may be zero and/or *scp* may be NULL. The meanings of *code* and *scp* and the conditions upon which they are other than zero or NULL are implementation dependent. It is permissible for *code* to always be zero, and *scp* to always be NULL.

    The pointer *scp* will only be valid during the context of the signal handler.

    The optional parameters can be omitted from the handler parameter list, in which case the handler is exactly compatible with System V UNIX. Truly portable software should not use the optional parameters in signal-catching routines.

    Before entering the signal-catching function, the value of *func* for the caught signal will be set to **SIG_DFL** unless the signal is one of those flagged with a bullet (●) above.

    Upon return from the signal-catching function, the receiving process will resume execution at the point it was interrupted.

    When a signal that is to be caught occurs during calls such as (but not limited to) *read*, *write*, *readv*, *writev*, *open*, or an *ioctl* system call on a slow device such as a terminal

(but not a file); during a *select*, *semop*, *msgsnd*, *msgrcv*, *pause*, or *sigpause* system call; or during a *wait* system call that does not return immediately due to the existence of a previously stopped or zombie process, the signal catching function will be executed. The interrupted system call may return a -1 to the calling process with *errno* set to EINTR.

Note: The signal **SIGKILL** cannot be caught.

SIGKILL may be sent by the system in the event of an unsuccessful *exec*, if the original program has already been deleted.

SPECIAL SIGNALS

SIGCLD, SIGPWR, and B SIGWINDOW signals have special characteristics described below and in hardware dependencies.

For these signals, *func* is assigned one of three values: **SIG_DFL, SIG_IGN,** or a *function address*. The actions prescribed by these values of are as follows:

SIG_DFL – ignore signal (applies to **SIGWINDOW** as well)
  The signal is to be ignored.

SIG_IGN – ignore signal
  The signal is to be ignored. Also, if *sig* is **SIGCLD**, the calling process's child processes will not create zombie processes when they terminate; see *exit*(2).

function address – catch signal
  If the signal is **SIGPWR**, the action to be taken is the same as that described above for *func* equal to *function address*. The same is true if the signal is **SIGCLD**, except that when *signal*(2) is called to install a signal–catching function, and the calling process currently has terminated (zombie) children, the **SIGCLD** signal will be immediately sent to the calling process.

  NOTE: If a signal–catching routine is used to catch **SIGCLD**, *wait*(2) must be called in the catching routine to clear out the pending zombie before *signal*(2) is called to reinstate the signal–catching routine. Otherwise, the process will enter the signal–catching routine recursively until the process runs out of stack space.

The **SIGCLD** signal affects two other system calls (*wait*(2), and *exit*(2)) in the following ways:

wait
  If the *func* value of **SIGCLD** is set to **SIG_IGN** and a *wait* is executed, the *wait* will block until all of the calling process's child processes terminate; it will then return a value of -1 with *errno* set to ECHILD.

exit
  If in the exiting process's parent process the *func* value of **SIGCLD** is set to **SIG_IGN**, the exiting process will not create a zombie process.

When processing a pipeline, the shell makes the last process in the pipeline the parent of the preceding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should take care not to set **SIGCLD** to be caught.

FAIL CONDITIONS

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

[EINVAL]
  Sig is an illegal signal number, or is equal to **SIGKILL**.

[EFAULT]
  Func points to an illegal address.

HARDWARE DEPENDENCIES

Series 200:
  The signal **SIGPWR** is not currently generated.

  The *code* word is always zero for all signals except signal 4 (**SIGILL**) and signal 8 (**SIGFPE**). For **SIGILL**, *code* has the following values:

0	illegal instruction;
6	check instruction;
7	TRAPV;
8	privilege violation.

For **SIGFPE**, *code* has the following values:

0	floating point exception;
5	divide–by–zero.

Refer to the MC68000 processor documentation provided with your system for more detailed information about the meaning of these errors.

Series 500:

Core image files are currently files with zero length.

The **SIGEMT** signal means "out of memory", and is generated by the HP–UX Operating System. When sent by the system, this signal is always fatal to the process, and cannot be caught or ignored.

SIGIOT can be sent if an invalid string operation is attempted, or if a bounds range check trap is encountered.

The signal **SIGBUS** is not currently generated by the operating system.

The signal handler parameter *code* contains the trap number provided by the hardware in the event a trap occurs in the user's program; see *trapno*(2) for a list of these trap numbers. The structure pointer *scp* is defined when a trap occurs in the user's program, and points to the structure *sigcontext* defined in **signal.h**.

A zero value is returned on floating point underflow. Floating point overflow, divide–by–zero, integer divide–by–zero, and illegal floating point operation exceptions result in the signal **SIGFPE** being sent to the process. An undefined value is returned as the result of the operation if the signal **SIGFPE** is ignored or caught.

SIGFPE is not sent on integer overflow. Instead, a wrapped integer result is returned.

The signal **SIGPWR** is generated by the operating system for the HP 9000 Model 550 only, provided that the system is equipped with a standby power supply. Series 500 models 520, 530, and 540 do not currently generate **SIGPWR**.

When the computer standby power supply is activated due to a loss of line voltage, the operating system determines whether a power glitch or power outage has occurred. The determination is made by waiting about five seconds then re–examining the state of the standby power supply. If the normal line voltage has been restored, *code* is set to **PWR_GLITCH**; otherwise, it is set to **PWR_OUTAGE**. The **SIGPWR** signal is then sent to all user processes (excluding the INIT process) with the appropriate *code* value. When normal line power is restored following a power outage, **SIGPWR** is sent to all user processes (again excluding the INIT process) with *code* set to **PWR_NORMAL**.

Integral PC:

The Integral PC implements the signal SIGMOUSE with a value of 20.

The Integral PC does not create core files.

RETURN VALUE

Upon successful completion, *signal* returns the previous value of *func* for the specified signal *sig*. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

SEE ALSO

kill(1), kill(2), lseek(2), pause(2), sigvector(2), trapno(2), wait(2), abort(3C), setjmp(3C).

NAME

    sigpause - atomically release blocked signals and wait for interrupt

SYNOPSIS

    long sigpause(sigmask)
    long sigmask;

HP–UX COMPATIBILITY

    Level: HP–UX/STANDARD

    Origin: UCB

DESCRIPTION

    Sigpause assigns *sigmask* to the set of masked signals and then waits for a signal to arrive; on
    return the set of masked signals is restored to the *sigmask* value which existed before the *sigpause*
    call. *Sigmask* is usually 0L to indicate that no signals are now to be blocked.

    Normally, at the beginning of a critical code section, a specified signal(s) is blocked using *sig-*
    block(2). When the process has completed the critical section and needs to wait for the previously
    blocked signal(s), it pauses by calling *sigpause* with the mask that was returned by the *sigblock*
    call.

RETURN VALUE

    Sigpause always terminates by being interrupted, returning –1 and setting *errno* to EINTR.

SEE ALSO

    sigblock(2), sigvector(2)

NAME
sigsetmask - set current signal mask

SYNOPSIS
long sigsetmask(mask);
long mask;

HP–UX COMPATIBILITY
Level: HP–UX/STANDARD

Origin: UCB

DESCRIPTION
Sigsetmask sets the current signal mask (those signals which are blocked from delivery). Signal i is blocked if the i-th bit in *mask* is a 1 (that is, if (mask & (1L << (i-1))) != 0).

It is not possible to mask those signals which cannot be ignored, as documented in *signal*(2); this restriction is silently imposed by the system.

Sigblock(2) can be used to add elements to the set of blocked signals.

RETURN VALUE
The previous set of masked signals is returned.

SEE ALSO
kill(2), sigvector(2), sigblock(2), sigpause(2)

NAME
     sigspace - assure sufficient signal stack space.

SYNOPSIS
     #include <signal.h>

     long sigspace(ss);
     long ss;

HP–UX COMPATIBILITY
     Level: HP–UX/STANDARD

     Origin: HP

DESCRIPTION
     Sigspace allows users to define additional space for stack use which is guaranteed to be available if signals are to be processed. If *ss* is positive, it specifies a space, in bytes, which the system guarantees will be available when processing a signal. A zero value removes any guarantee of space and any negative value leaves the guarantee unchanged, and may be used to interrogate the current guaranteed value. When a signal's action indicates its handler should use the guaranteed space (specified with a *sigvector*(2) call), the system checks to see if the process is currently using that space. If the process is not currently using that space the system arranges for that space to be available for the duration of the signal handler's execution. If that space has already been made available (due to a previous signal) no change is made. The normal stack discipline is resumed when the signal causing the use of the guaranteed space is exited.

     The guaranteed space is inherited by child processes after a *fork* but the guarantee of space is removed after an *exec*.

NOTES
     The guaranteed space may not be increased in size automatically, as is done for the normal stack. If the stack overflows the guaranteed space unpredictable results may occur.

     Guaranteeing space for a stack may cause interference with other memory allocation routines, in an implementation dependent manner.

     During normal execution of the program the system checks for possible overflow of the stack. Guaranteeing space may cause the space available for normal execution to be reduced.

     Leaving the context of a service routine in an abnormal way, such as by *longjmp*(3), may remove the guarantee that the ordinary execution of the program will not extend into the guaranteed space. It may also cause the program to forever loose its ability to automatically increase the stack size, and the program may then be limited to the guaranteed space.

RETURN VALUE
     Upon successful completion, the size of the old guaranteed space is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

ERRORS
     Sigspace will fail and the guaranteed amount of space will remain unchanged if one of the following occurs.

     [ENOMEM]   Enough space cannot be guaranteed because of either hardware limitations or because some software imposed limit would be exceeded.

HARDWARE DEPENDENCIES
     Series 500:
          Sigspace is ignored (as a no–op) by Series 500. The return value is always zero.

     Series 200 and 300:
          The guaranteed space is allocated with malloc(3). This call may thus interfere with other heap management mechanisms.

The kernel overhead taken in the reserved space is 148 bytes on Series 200 computers and 440 bytes on Series 300. This overhead must be included in the requested amount. These values are subject to change in future releases.

BUGS

Methods for calculating the required size are not yet well developed.

SEE ALSO

sigvector(2), setjmp(3)

NAME

sigvector - software signal facilities

SYNOPSIS

#include <signal.h>

sigvector(sig, vec, ovec)
int sig;
struct sigvec *vec, *ovec;

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB and HP

DESCRIPTION

The system defines a set of signals that may be delivered to a process. The set of signals is defined in *signal*(2), along with the meaning and side effects of each signal. This manual page, along with those for *sigblock*(2), *sigsetmask*(2), *sigpause*(2), and *sigspace*(2) define an alternate mechanism for handling these signals that assures the delivery of signals and integrity of signal handling procedures. The facilities described here should not be used in the same program as *signal*(2) without a thorough understanding of the interactions between the two mechanisms.

With this interface, signal delivery resembles the occurrence of a hardware interrupt: the signal is blocked from further occurrence, the current process context is saved, and a new one is built. A process may specify a *handler* to which a signal is delivered, or specify that a signal is to be *blocked* or *ignored*. A process may also specify that a default action is to be taken by the system when a signal occurs. It is possible to assure a minimum amount of stack space for processing signals using the *sigspace*(2) call.

All signals have the same priority. Signal routines execute with the signal that caused their invocation *blocked*, but other signals may yet occur. A global *signal mask* defines the set of signals currently blocked from delivery to a process. The signal mask for a process is initialized from that of its parent (normally 0). It may be changed with a *sigblock*(2), *sigsetmask*(2), or *sigpause*(2) call, or when a signal is delivered to the process.

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

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

Sigvector assigns a handler for a specific signal. *Vec* and *ovec* are pointers to *sigvec* structures which include the following elements:

```
int     (*sv_handler)();
int     sv_mask;
int     sv_onstack;
```

If *vec* is non-zero, it specifies a handler routine and mask to be used when delivering the specified signal. Further, if *sv_onstack* is 1, the system will use, or permit the use of, the space reserved for signal processing in the *sigspace*(2) call. If *ovec* is non-zero, the previous handling information

for the signal is returned to the user. If *vec* is zero, signal handling is unchanged; thus, the call can be used to enquire about the current handling of a given signal.

Once a signal handler is installed, it remains installed until another *sigvector* call is made, or an *exec*(2) is performed. The default action for a signal may be reinstated by setting *sv_handler* to SIG_DFL; this default is usually termination. If *sv_handler* is SIG_IGN the signal is usually subsequently ignored, and pending instances of the signal are discarded. The exact meaning of SIG_DFL and SIG_IGN for each signal is discussed in *signal*(2). Unlike *signal*(2) there is no category of "reset when caught" signals.

Certain system calls can be interrupted by a signal, the remainder will complete before the signal is serviced. The *scp* pointer described in *signal*(2) is always non–null if *sigvector* is supported. *Scp* points to a machine–dependent *sigcontext* structure. All implementations of this structure include the field

```
int   sc_syscall;
char  sc_syscall_action;
```

The value **SYS_NOTSYSCALL** for the sc_syscall field indicates that the signal is not interrupting a system call; any other value indicates which system call it is interrupting.

If a signal which is being caught occurs during one of the interruptable calls, the signal handler is immediately invoked. If the signal handler is exited in a normal way, the value of the sc_syscall_action field is inspected; if the value is **SIG_RETURN,** the system call is aborted and the interrupted program continues past the call with the result of the interrupted call being –1 and *errno* set to EINTR. If the value of the sc_syscall_action field is **SIG_RESTART,** the call is restarted. Other values are undefined and reserved for future use.

Exiting the handler abnormally (such as with *longjmp*(3)) will abort the call, and the user is responsible for the context of further execution. The value of *scp–>sc_syscall* is ignored when the value of *scp–>sc_syscall* is **SYS_NOTSYSCALL.**

When a signal interrupts a *read, write, readv,* or *writev* call that has transferred a non–zero portion of the requested data, the call is considered to have completed with a partial transfer. In this case, the signal handler is invoked with *scp–>sc_syscall* set to **SYS_NOT_SYSCALL** and, following return from the handler, the system call returns the number of bytes actually transferred.

When an interruptable call is interrupted by multiple signals, if any signal handler returns a value of **SIG_RETURN** in *scp–>sc_syscall_action,* all subsequent signal handlers are passed a value of **SYS_NOTSYSCALL** in *scp–>sc_syscall. scp–>sc_syscall_action* is always initialized to **SIG_RETURN** before invocation of a signal handler.

The interruptable system calls, and corresponding values for *scp–>sc_syscall* are listed below.

Call	sc_syscall value
read (slow devices)	**SYS_READ**
readv (slow devices)	**SYS_READV**
write (slow devices)	**SYS_WRITE**
writev (slow devices)	**SYS_WRITEV**
open (slow devices)	**SYS_OPEN**
ioctl (slow requests)	**SYS_IOCTL**
wait	**SYS_WAIT**

select	SYS_SELECT
pause	SYS_PAUSE
sigpause	SYS_SIGPAUSE
semop	SYS_SEMOP
msgsnd	SYS_MSGSND
msgrcv	SYS_MSGRCV

Note that *read, readv, write, writev, open,* or *ioctl* on fast devices (discs) is not interruptable, but I/O to a slow device (terminal) is. Additional system calls such as those used for networking may also be interruptable on some implementations. In these cases, additional values may be specified for *scp->sc_syscall*. Programs that look at the values of *scp->sc_syscall* should always compare them to these symbolic constants; the numerical values represented by these constants vary among implementations.

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

Exec(2) resets all caught signals to default action; ignored signals remain ignored; the signal mask remains the same; the reserved signal stack space is released.

NOTES

The mask specified in *vec* is not allowed to block those signals which cannot be ignored, as defined in *signal*(2). This is enforced silently by the system.

If *sigvector* is called to catch **SIGCLD** in a process which has currently terminated (zombie) children, a **SIGCLD** signal is delivered to the calling process immediately, or as soon as **SIGCLD** is unblocked if it is currently blocked. Thus, in a process which spawns multiple children and catches **SIGCLD**, it is somtimes advisable to re-install the handler for **SIGCLD** after each invocation in case there are multiple zombies present. This is true even though the handling of the signal is not reset by the system as with *signal*(2); otherwise, the presence of the zombie which caused the first signal will always cause another signal to be sent.

RETURN VALUE

A 0 value indicated that the call succeeded. A -1 return value indicates an error occurred and *errno* is set to indicate the reason.

ERRORS

Sigvector will fail and no new signal handler will be installed if one of the following occurs:

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

[EINVAL]   *Sig* is not a valid signal number.

[EINVAL]   An attempt is made to ignore or supply a handler for a signal which cannot be caught or ignored. See *signal (2)*.

SEE ALSO

kill(1), ptrace(2), kill(2), sigblock(2), sigsetmask(2), sigpause(2), sigspace(2), setjmp(3), signal(2), termio(4)

WARNING

Restarting a *select*(2) call can sometimes cause unexpected results. If the select call has a timeout specified, the timeout is restarted with the call, ignoring any portion which had elapsed prior to interruption by the signal. Normally this simply extends the timeout and is not a problem. However, if a handler repeatedly catches signals and the timeout specified to select is longer than the time between those signals, restarting the select call effectively renders the timeout infinite.

NAME

stat, lstat, fstat - get file status

SYNOPSIS

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

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

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

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

Level: Stat and fstat: HP–UX/RUN ONLY
         Lstat: HP–UX/EXTENDED

Origin: System V Release 2

DESCRIPTION

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

Similarly, *fstat* obtains information about an open file known by the file descriptor *fildes*, obtained from a successful *open*, *creat*, *dup*, *fcntl*, or *pipe* system call.

Lstat is like *stat* except in the case where the named file is a symbolic link, in which case *lstat* returns information about the link, while *stat* returns information about the file the link references. Not all HP-UX systems provide symbolic links.

Buf is a pointer to a *stat* structure into which information is placed concerning the file.

The contents of the structure pointed to by *buf* include the following members:

```
dev_t    st_dev;      /* ID of device containing a */
                      /* directory entry for this file */
ino_t    st_ino;      /* Inode number */
ushort   st_mode;     /* File mode; see mknod(2) */
short    st_nlink;    /* Number of links */
ushort   st_uid;      /* User ID of file owner */
ushort   st_gid;      /* Group ID of file group */
dev_t    st_rdev;     /* Device ID; this entry defined */
                      /* only for char or blk spec files */
off_t    st_size;     /* File size (bytes) */
time_t   st_atime;    /* Time of last access */
time_t   st_mtime;    /* Last modification time */
time_t   st_ctime;    /* Last file status change time */
                      /* Measured in secs since */
                      /* 00:00:00 GMT, Jan 1, 1970 */
```

st_atime Time when file data was last accessed. Changed by the following system calls: *creat*(2), *mknod*(2), *pipe*(2), *utime*(2), and *read*(2).

st_mtime

Time when data was last modified. Changed by the following system calls: *creat*(2), *mknod*(2), *pipe*(2), *utime*(2), and *write*(2).

st_ctime Time when file status was last changed. Changed by the following system calls: *chmod*(2), *chown*(2), *creat*(2), *link*(2), *mknod*(2), *pipe*(2), *unlink*(2), *utime*(2), and *write*(2).

The *touch*(1) command can be used to explicitly control the times of a file.

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

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

[ENOENT]   The named file does not exist.

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

[EFAULT]   *Buf* or *path* points to an invalid address.

[ENOENT]   *Path* is null.

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

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

[EFAULT]   *Buf* points to an invalid address.

RETURN VALUE

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

HARDWARE DEPENDENCIES

Integral PC:

The *bstat* and *bfstat* calls are identical to *stat* and *fstat* except that the *st_dev* and *st_rdev* fields in the *bstat* structure are larger than the *st_dev* and *st_rdev* fields in the *stat* structure. *Bstat* and *bfstat* are used in place of *stat* and *fstat* on HP–UX implementations with long device numbers (such as the Integral PC). This difference is relevant only to applications which examine the *st_dev* or *st_rdev* fields of the *stat* (*bstat*) structure.

```
int     bstat (path, buf)
char    *path;
struct  bstat *buf;


int     bstat (fildes, buf)
int     fildes;
struct  bstat *buf;
```

Series 200 and 500:

Symbolic links are not supported on Series 200 and Series 500 computers.

SEE ALSO

touch(1), chmod(2), chown(2), creat(2), link(2), mknod(2), pipe(2), read(2), time(2), unlink(2), utime(2), write(2), stat(7).

NAME
stime - set time and date

SYNOPSIS
int stime (tp)
long *tp;

HP–UX COMPATIBILITY
Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION
Stime sets the system's idea of the time and date. *Tp* points to the value of time as measured in seconds from 00:00:00 GMT January 1, 1970.

[EPERM]   *Stime* will fail if the effective user ID of the calling process is not super–user.

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

HARDWARE DEPENDENCIES
Integral PC:
The super–user capabilities are provided to the normal user.

SEE ALSO
date(1), gettimeofday(2), time(2).

NAME
stty, gtty - control device

SYNOPSIS
#include <sgtty.h>

stty(fildes,argp)
int fildes;
struct sgttyb *argp;

gtty(fildes,argp)
int fildes;
struct sgttyb *argp;

HP–UX COMPATIBILITY

Level: Version 6 and PWB Compatibility – HP–UX/STANDARD

Origin: Version 6

Remarks: This system call is preserved for backward compatibility with Bell Version 6. It pro-
            vides as close an approximation as possible to the old Version 6 function. All new
            code should use the TCSETA/TCGETA *ioctl* calls described in *termio*(4). Note that
            these calls are incompatible with the Version 7 calls of the same names.

            Not supported on the Integral Personal Computer.

DESCRIPTION
For certain status setting and status inquiries about terminal devices, the functions *stty* and *gtty*
are equivalent to

            ioctl(fildes, TIOCSETP, argp)
            ioctl(fildes, TIOCGETP, argp)

respectively; see *sttyV6*(4).

SEE ALSO
stty(1), exec(2), sttyV6(4), termio(4).

DIAGNOSTICS
Zero is returned if the call was successful; –1 if the file descriptor does not refer to the kind of file
for which it was intended.

NAME
swapon - add a swap device for interleaved paging/swapping

HP–UX COMPATIBILITY
Level: Large Machine/HP Extension/HFS

Origin: UCB

SYNOPSIS
swapon(special)
char *special;

DESCRIPTION
Swapon makes the block device *special* available to the system so it can be allocated for paging and swapping. The names of potentially available devices are known to the system and defined at system configuration time. See the System Administrator's Manual for information on how the size of the swap area is calculated.

HARDWARE DEPENDENCIES
Not implemented on Series 500 or Integral PC.

SEE ALSO
swapon(1M)

BUGS
There is no way to stop swapping on a disc so that the pack can be dismounted.

NAME

sync - update super-block

SYNOPSIS

void sync ( )

HP–UX COMPATIBILITY

Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION

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

sync should be used by programs which examine a file system, for example *fsck*, *df*, etc. Sync is mandatory before stopping the system (such as when rebooting) in order to ensure the preservation of file system integrity.

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

In some HP–UX systems, *sync* may be reduced to a no–op. This is permissible on a system which does not cache buffers, or in a system that in some way ensures that the discs are always in a consistent state.

SEE ALSO

sync(8), reboot(2), reboot(1M).

NAME

time - get time

SYNOPSIS

long time ((long *) 0)

long time (tloc)
long *tloc;

HP–UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System V

DESCRIPTION

Time returns the value of time in seconds since 00:00:00 GMT, January 1, 1970.

If *tloc* (taken as an integer) is non–zero, the return value is also stored in the location to which *tloc* points.

[EFAULT]   *Time* will fail if *tloc* points to an illegal address.

RETURN VALUE

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

SEE ALSO

date(1), gettimeofday(2), stime(2).

NAME
times - get process and child process times

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

long times (buffer)
struct tms *buffer;

HP–UX COMPATIBILITY
Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION
Times fills the structure pointed to by *buffer* with time–accounting information. The structure defined in **sys/times.h** looks as follows:

```
struct tms {
    time_t  tms_utime;   /* user time */
    time_t  tms_stime;   /* system time */
    time_t  tms_cutime;  /* user time, children */
    time_t  tms_cstime;  /* system time, children */
};
```

This information comes from the calling process and each of its terminated child processes for which it has executed a *wait*. The times are in units of 1/HZ seconds, where HZ is processor dependent (see **<sys/param.h>**).

Tms_utime is the CPU time used while executing instructions in the user space of the calling process.

Tms_stime is the CPU time used by the system on behalf of the calling process.

Tms_cutime is the sum of the *tms_utime*s and *tms_cutime*s of the child processes.

Tms_cstime is the sum of the *tms_stime*s and *tms_cstime*s of the child processes.

[EFAULT]   *Times* will fail if *buffer* points to an illegal address.

HARDWARE DEPENDENCIES
Series 500:
For computers with multiple CPU's, the child CPU times listed can be greater than the actual elapsed real time, since the CPU time is counted on a per–CPU basis. Thus, if all three CPUs are executing, the CPU time is the sum of the three execution times of the CPUs.

RETURN VALUE
Upon successful completion, *times* returns the elapsed real time, in units of 1/HZ of a second, since an arbitrary point in the past (e.g., system start–up time). This point does not change from one invocation of *times* to another. If *times* fails, a -1 is returned and *errno* is set to indicate the error.

SEE ALSO
exec(2), fork(2), time(2), wait(2).

BUGS
Not all CPU time expended by system processes on behalf of a user process is counted in the system CPU time for that process.

NAME

trapno - hardware trap numbers

HP–UX COMPATIBILITY

Level: HP–UX/NON STANDARD

Origin: HP

Remarks: The following description of hardware trap numbers is valid for the Series 500 only.

DESCRIPTION

The following trap numbers refer to hardware traps occurring on the HP 9000 Series 500 computers. *Trapno* values are reported by the *err*(1) command, and are passed to signal handlers (see *signal*(2)) when hardware traps cause signals to be sent to the current process.

The *trapno* value, trap name, and description are listed below for each possible trap condition. By convention, trap numbers are shown in octal.

VALUE	NAME: DESCRIPTION
01	Bounds Violation: An address is outside the limits for the program, stack, or global data segments. [2]
02	Check Trap: A user value is outside a prescribed range. [1]
03	Breakpoint Trap: Debugging trap. [1]
04	Machine Instruction Trap: Used by the operating system.
05	String Trap: Illegal string operation or data. [2]
06	Unused.
07	Unused.
010	Reset: Used by the operating system.
011	Page Table Violation: The page table entry referenced is beyond the current length of the page table. [2]
012	Inconsistent Registers: An attempt was made to set up an inconsistent set of registers describing the global data segment, stack segment, or program segment. [2]
013	External Data Segment Bounds Violation: An address is outside the limits of an external data segment. [2]
014	System Error: Used by the operating system.
015	External Data Segment Pointer Violation: Illegal data segment pointer; probably a pointer between 0 and 524 287 decimal. [2]
016	Pointer Conversion Violation: An attempt was made to form a data segment pointer with an offset which is too large for the type of pointer being used. [2]
017	External Program Pointer Violation: Illegal procedure pointer. [2]
020	Unimplemented Instruction: Attempt to execute an undefined instruction. [1]
021	STT Violation: Illegal procedure pointer. [2]
022	CST Violation: Illegal procedure pointer. [2]
023	DST Violation: Illegal segment number in an external data segment pointer. [2]
024	Stack Overflow: The operating system normally handles this trap by extending the stack segment.
025	Stack Underflow: An attempt to pop a word from the local stack when the local stack is empty. [2]
026	Privileged Mode Violation: An attempt to execute a privileged instruction or return to a privileged procedure while in unprivileged mode. [2]

027	Privileged Mode Data Violation: An attempt to reference a privileged data segment while in unprivileged mode. [2]
030	Unexpected Pointer Type: An instruction has encountered a pointer type which it cannot handle. [2]
031	User Traps: Integer divide by zero. [1]
032	Illegal Decimal Number: A decimal math instruction has been supplied an illegal operand. [2]
033	Exponent Size Trap: Exponent too large during a number conversion instruction. [2]
034	Floating Point Operand Trap: Attempt to operate on illegal numbers, divide by zero, or convert a 64 bit number to a 32–bit number which cannot accommodate the exponent. [1]
035	Floating Point Result Trap: Floating point overflow; also caused by an explicit request to trap on an inexact result. [1]
036	Unexpected External Data Segment Type: A paged external data segment was encountered when an unpaged segment was expected, or vice versa. [2]
037	Absent Code Segment: Handled by the operating system.
040	Absent Page: Handled by the operating system.
041	Uncallable Procedure: Attempt to call an uncallable privileged procedure while in unprivileged mode. [2]
042	Absent Data Segment: Handled by the operating system.
043	Absent Page Table: Handled by the operating system.
044	Start–of–Line: Debugging trap. [1]
045	Variable Trace: Debugging trap. [1]
046	Start–of–Procedure: Debugging trap. [1]
047	End–of–Procedure: Debugging trap. [1]
050	Start–of–Subroutine: Debugging trap. [1]
051	End–of–Subroutine: Debugging trap. [1]
052	Code Segment Violation: Attempt to modify a code segment. [2]
053	Branch Violation: Illegal branch instruction. [2]
054	Message Trap: Used internally by the operating system.
055	Instruction Sequencing Bounds Violation: Program destination is out of bounds; probably a stack marker has been incorrectly modified.
056	Start–of–Line–Check Trap: Debugging trap. [1]
057	Data Segment Write Violation: Attempt to modify a write-protected data segment. [2]
060	System semaphore trap on up; relative pointer. [1]
061	System semaphore trap on up; absolute pointer. [1]
062	System semaphore trap on down; relative pointer. [1]
063	System semaphore trap on down; absolute pointer. [1]
064	Invalid internal math transformation. [1]

The footnotes are as follows:

[1]:   If the program returns from the trap (signal) handler, execution will resume with the next instruction.

[2]:   If the program returns from the trap (signal) handler, execution will resume at the current instruction.

SEE ALSO

err(1), signal(2).

WARNING

    Trapno is intended for diagnostic purposes only. Values and meanings may change in future releases of HP–UX.

NAME

truncate - truncate a file to a specified length

SYNOPSIS

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

ftruncate(fd, length)
int fd, length;

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: UCB

Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION

Truncate causes the file named by *path* or referenced by *fd* to be truncated to at most *length* bytes in size. If the file previously was larger than this size, the extra data is lost. With *ftruncate*, the file must be open for writing; for *truncate* the user must have write permission for the file.

RETURN VALUES

A value of 0 is returned if the call succeeds. If the call fails a -1 is returned, and the global variable *errno* specifies the error.

ERRORS

Truncate succeeds unless:

[ENOENT]   The pathname was too long.

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

[ENOENT]   The named file does not exist.

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

[EACCES]   Write permission is denied on the file.

[EISDIR]   The named file is a directory.

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

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

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

Ftruncate succeeds unless:

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

[EINVAL]   The *fd* references a file that was opened without write permission.

SEE ALSO

open(2)

BUGS

Partial blocks discarded as the result of truncation are not zero filled; this can result in holes in files which do not read as zero.

These calls should be generalized to allow ranges of bytes in a file to be discarded.

NAME

  ulimit - get and set user limits

SYNOPSIS

  long ulimit (cmd, newlimit)
  int cmd;
  long newlimit;

HP-UX COMPATIBILITY

  Level: HP-UX/RUN ONLY

  Origin: System V

DESCRIPTION

  This function provides for control over process limits. The *cmd* values available are:

  1   Get the file size limit of the process. The limit is in units of 512-byte blocks and is inherited
          by child processes. Files of any size can be read.

  2   Set the file size limit of the process to the value of *newlimit*. Any process may decrease this
          limit, but only a process with an effective user ID of super-user may increase the limit. Note
          that the limit must be specified in units of 512-byte blocks.

  3   Get the maximum possible break value. See *brk*(2). *Ulimit* will fail if one or more of the
          following conditions is true.

  [EINVAL]   *cmd* is not in the correct range.

  [EPERM]   *Ulimit* will fail and the limit will be unchanged if a process with an effective user
                ID other than super-user attempts to increase its file size limit.

RETURN VALUE

  Upon successful completion, a non-negative value is returned. Errors return a –1, with *errno* set
  appropriately.

SEE ALSO

  brk(2), write(2).

NAME

umask - set and get file creation mask

SYNOPSIS

int umask (cmask)
int cmask;

HP–UX COMPATIBILITY

Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION

Umask sets the process's file mode creation mask to *cmask* and returns the previous value of the mask. Only the low–order 9 bits of *cmask* and the file mode creation mask are used.

The bits that are set in *cmask* specify which permission bits to turn off in the mode of the created file. For example, suppose a value of 007 is specified for *cmask*. Then, if a file is normally created with permissions of 0777, its mode after creation would be 0770.

RETURN VALUE

The previous value of the file mode creation mask is returned.

SEE ALSO

mkdir(1), sh(1), chmod(2), creat(2), mknod(2), open(2), mknod(8).

NAME

umount - unmount a file system

SYNOPSIS

int umount (spec)
char *spec;

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System V

DESCRIPTION

Umount requests that a previously mounted file system contained on the block special device identified by *spec* be unmounted. *Spec* is a pointer to a path name. After unmounting the file system, the directory upon which the file system was mounted reverts to its ordinary interpretation.

Umount may be invoked only by the super–user.

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

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

[ENOENT]   *Spec* does not exist.

[ENOTBLK]   *Spec* is not a block special device.

[EINVAL]   *Spec* is not mounted.

[EBUSY]   A file on *spec* is busy.

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

[ENXIO]   The device associated with *spec* does not exist.

[ENOTDIR]   A component of *spec* is not a directory.

[ENOENT]   *Spec* is null.

RETURN VALUE

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

HARDWARE DEPENDENCIES

Integral PC:

The super–user capabilites are provided to the normal user.

SEE ALSO

mount(1), mount(2).

BUGS

If *umount* is called from the program level (i.e. not from the *mount*(1) level), the table of mounted devices contained in /etc/mnttab is not updated.

NAME

uname - get name of current HP UX system

SYNOPSIS

#include <sys/utsname.h>

int uname (name)
struct utsname *name;

HP–UX COMPATIBILITY

Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION

Uname stores information identifying the current HP-UX system in the structure pointed to by *name*.

Uname uses the structure defined in **<sys/utsname.h>** whose members include:

```
#define UTSLEN              9
#define SNLEN               I5

        char        sysname[UTSLEN];
        char        nodename[UTSLEN];
        char        release[UTSLEN];
        char        version[UTSLEN];
        char        machine[UTSLEN];
        char        idnumber[SNLEN];
```

Uname returns a null–terminated string in each field. *Sysname* contains "HP–UX". Similarly, *nodename* contains the name that the system is known by on a communications network and is accessible via *hostname*(1), *sethostname*(2), and *gethostname*(2). *Release* contains the release number of the operating system, e.g. "1.0" or "3.0.1". *Version* contains additional information about the operating system. *Machine* contains a standard name that identifies the hardware on which the HP–UX system is running. *Idnumber* contains an identification number which is unique within that class of hardware, possibly a hardware or software serial number. This field may return the null string to indicate the lack of an identification number.

[EFAULT] *Uname* will fail if *name* points to an invalid address.

HARDWARE DEPENDENCIES

Series 200/500:

The first character of the *version* field is set to "A" for single user, and "B" for 16–user.

Series 500 only:

The first character of the *version* field is set to "C" for 32–user systems, and "D" for 64–user systems.

RETURN VALUE

Upon successful completion, a non–negative value is returned. Otherwise, -1 is returned and *errno* is set to indicate the error.

SEE ALSO

hostname(1), uname(1), gethostname(2), sethostname(2).

NAME

unlink - remove directory entry; delete file

SYNOPSIS

int unlink (path)
char *path;

HP–UX COMPATIBILITY

Level: HP–UX/RUN ONLY

Origin: System V

DESCRIPTION

Unlink removes the directory entry named by the path name pointed to by *path*.

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

[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	The named file does not exist.
[EACCES]	Search permission is denied for a component of the path prefix.
[EACCES]	Write permission is denied on the directory containing the link to be removed.
[EPERM]	The named file is a directory and the effective user ID of the process is not super–user.
[EBUSY]	The entry to be unlinked is the mount point for a mounted file system.
[ETXTBSY]	The entry to be unlinked is the last link to a pure procedure (shared text) file that is being executed.
[EROFS]	The directory entry to be unlinked is part of a read–only file system.
[EFAULT]	*Path* points outside the process's allocated address space.
[ENOENT]	*Path* is null.
[ENOENT]	A component of *path* does not exist.

When all links to a file have been removed and no process has the file open, the space occupied by the file is freed and the file ceases to exist. If one or more processes have the file open when the last link is removed, the removal is postponed until all references to the file have been closed.

HARDWARE DEPENDENCIES

Series 500:

The last link to a directory cannot be unlinked if the directory is not empty.

RETURN VALUE

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

SEE ALSO

rm(1), close(2), link(2), open(2).

NAME
     ustat - get file system statistics

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

     int ustat (dev, buf)
     dev_t dev;
     struct ustat *buf;

HP–UX COMPATIBILITY
     Level: HP–UX/RUN ONLY

     Origin: System V

DESCRIPTION
     Ustat returns information about a mounted file system. *Dev* is a device number identifying a
     device containing a mounted file system. *Buf* is a pointer to a *ustat* structure (defined in
     ustat.h) that includes the following elements:

               daddr_t f_tfree;   /* Total free blocks*/
               ino_t   f_tinode;   /* Number of free inodes */
               char   f_fname[6]; /* Filsys name */
               char   f_fpack[6]; /* Filsys pack name */
               int   f_blksize;   /* Block size */

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

     [EINVAL]   *Dev* is not the device number of a device containing a mounted file system.

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

HARDWARE DEPENDENCIES
     Series 500:
          In the above structure, f_fname[6] is the driver name, not the file system name.

     Series 200:
          f_tfree and f_blksize are reported in fragment size units.

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

SEE ALSO
     touch(1), stat(2), fs(5).

NAME
utime - set file access and modification times

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>
int utime (path, times)
char *path;
struct utimbuf *times;

HP-UX COMPATIBILITY
Level: HP-UX/RUN ONLY

Origin: System V

DESCRIPTION
Path points to a path name naming a file. *Utime* sets the access and modification times of the named file.

If *times* is **NULL**, the access and modification times of the file are set to the current time. A process must be the owner of the file or have write permission to use *utime* in this manner.

If *times* is not **NULL**, *times* is interpreted as a pointer to a *utimbuf* structure and the access and modification times are set to the values contained in the designated structure. Only the owner of the file or the super-user may use *utime* this way.

The times in the following structure, found in <unistd.h>, are measured in seconds since 00:00:00 GMT, Jan. 1, 1970.

```
struct utimbuf {
    time_t  actime;    /* access time */
    time_t  modtime;   /* modification time */
};
```

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

[ENOENT]	The named file does not exist.
[ENOTDIR]	A component of the path prefix is not a directory.
[EACCES]	Search permission is denied by a component of the path prefix.
[EPERM]	The effective user ID is not super-user and not the owner of the file and *times* is not **NULL**.
[EACCES]	The effective user ID is not super-user and not the owner of the file and *times* is **NULL** and write access is denied.
[EROFS]	The file system containing the file is mounted read-only.
[EFAULT]	*Times* is not **NULL** and points outside the process's allocated address space.
[EFAULT]	*Path* points outside the process's allocated address space.

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

SEE ALSO
touch(1), stat(2).

NAME

vfork - spawn new process in a virtual memory efficient way

SYNOPSIS

int vfork()

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: UCB

Remarks: *Vfork* is provided as a higher performance version of *fork* on those systems which
choose to provide it and for which there is a performance advantage.

Vfork differs from *fork* only in that the child process may share code and data with
the calling process (parent process). This speeds the cloning activity significantly at a
risk to the integrity of the parent process if *vfork* is misused.

The use of *vfork* for any purpose except as a prelude to an immediate *exec* or *exit* is
not supported. Any program which relies upon the differences between *fork* and *vfork*
is not portable across HP–UX systems.

All implementations of HP UX must provide the entry *vfork*, but it is permissible for
them to treat it identically to *fork*. Some implementations may not choose to distin-
guish the two because their implementation of fork is as efficient as possible, and oth-
ers may not wish to carry the added overhead of two similar calls.

DESCRIPTION

Vfork can be used to create new processes without fully copying the address space of the old pro-
cess. If a forked process is simply going to do an *exec*(2), the data space copied from the parent to
the child by *fork(2)* is not used. This is particularly inefficient in a paged environment. *Vfork* is
useful in this case. Depending upon the size of the parent's data space, it can give a significant
performance improvement over *fork*.

Vfork differs from *fork* in that the child borrows the parent's memory and thread of control until
a call to *exec* or an exit (either by a call to *exit*(2) or abnormally.) The parent process is
suspended while the child is using its resources.

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

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

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

[EAGAIN]   The system–wide limit on the total number of processes under execution would be
exceeded.

[EAGAIN]   The system–imposed limit on the total number of processes under execution by a sin-
gle user would be exceeded.

HARDWARE DEPENDENCIES

Series 200 and 500: The [vfork,exec] window begins at the *vfork* call and ends when the child
completes its *exec* call.

Series 500:

Process times for the parent and child processes within the [vfork,exec] window may be inaccu-
rate.

Shared memory segments generated with the *EMS* intrinsics will be inherited over *vfork*. Private memory segments will not be copied over *vfork*.

Vfork will also fail in the following cases:

[ENOMEM]
  There is not enough physical memory to create the new process.

[EAGAIN] The child process attempts to do a second *vfork* or a *fork* while in the [vfork,exec] window.

The parent and child processes share the same stack space within the [vfork,exec] window. If the size of the stack has been changed within this window by the child process (return from or call to a function, for example), it is likely that the parent and child processes will be killed with signal SIGSEGV.

A child process which does not call *exec* will not generate a process accounting record.

Series 200:
  A call to *signal(2)* in the [vfork,exec] window which is used to catch a signal can affect handling of the signal by the parent. This is not true if the signal is set to SIG_DFL or SIG_IGN, or if *sigvector(2)* is used.

Integral PC:
  On the Integral PC, *vfork*ed children have a unique 2K–byte stack allocated to them. Any stack space used beyond this 2K limit is shared between the child and the parent.

RETURN VALUE

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

SEE ALSO

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

NAME

vsadv - advise system about backing store usage

SYNOPSIS

#include <sys/ems.h>

vsadv (index);
int index;

HP-UX COMPATIBILITY

Level: Backing Store Control – HP–UX/STANDARD

Origin: HP

Remarks: *Vsadv* is not currently implemented on the Series 200.

DESCRIPTION

This call requests that all future backing store space allocated for this process be placed on the backing store device specified by *index* (see *vson*(2)). It may be used to tune an application to the local system environment. This request remains in effect until the next call to *vsadv* by this process. An *index* of –1 will set backing store allocation back to the default system policy.

This call is advisory in nature and will never cause subsequent program failures (e.g. if the device has no room, the system will simply override the request and use another device).

This characteristic is inherited across *fork*(2) and *exec*(2).

This call may be reduced to a no–op.

HARDWARE DEPENDENCIES

This system call is supported on Series 500 only.

SEE ALSO

ems(2), vson(2).

NAME

vson, vsoff - advise OS about backing store devices

SYNOPSIS

#include <sys/ems.h>

int vson(pathname, size, q);
int size, q;
char *pathname;

int vsoff(index, force);
int index, force;

HP–UX COMPATIBILITY

Level: Backing Store Control – HP–UX/STANDARD

Origin: HP

Remarks: *Vson* and *vsoff* are not currently implemented on the Series 200.

DESCRIPTION

Vson is used to make the block special file *pathname* available for use by the system as a backing store device for whatever form of backing store is needed by the system. The call returns an id by which the backing store device may be referenced in subsequent *vsoff* or *vsadv*(2) calls. Multiple *vson* calls for the same device will return the same id (here "same device" means identical devno – major and minor – and not necessarily the same file name).

Pathname specifies a block special device file, which may or may not contain a mounted file system, and which must be a CS–80 device. If device does not contain a file system (i.e. an "empty" disc), *size* specifies the available backing storage space (in blocks) to be made available (the storage space is assumed to start at block 0 in this case). If *size* is set to -1 and the device does not contain a file system, the whole block special device will be used for backing store.

Q is a quality (i.e. performance) factor for the device. It is used by the system in load balancing decisions. Higher values suggest secondary choices for backing store devices. There is no inherent significance to the value of *q* other than its value relative to the *q* factor of the other devices in the list. This parameter may be ignored on some implementations.

Vsoff is used to remove a device from the list of backing store devices available to the system. *Index* is the value returned by *vson* when the device was added to the list.

If *force* is not set (i.e. is 0) the system attempts to "gracefully" eliminate backing store usage of device by migrating backing store space onto other devices. If *force* is set (if, for instance, the device has failed) no attempt is made to salvage images stored on the disc. Processes with images on the device will, in all probability, be rather ungracefully terminated in the near future (i.e. when the images are required).

Only the super–user may add or remove backing store devices. A normal user may call *vson* to get the id for a device already known to the system as a backing store device (for subsequent use in a *vsadv*(2) call).

RETURN VALUES

Upon successful completion, *vson* returns the index for the device and *vsoff* returns 0. If there is an error, a value of -1 is returned and *errno* is set to indicate the error.

HARDWARE DEPENDENCIES

This system call is supported on Series 500 only.

SEE ALSO

ems(2), memallc(2), vsadv(2)

NAME

wait - wait for child process to stop or terminate

SYNOPSIS

int wait (stat_loc)
int *stat_loc;

int wait ((int *)0)

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System V

DESCRIPTION

Wait suspends the calling process until one of the immediate children terminates or until a child that is being traced stops, because it has hit a break point. The *wait* system call will return prematurely if a signal is received. If a child process stopped or terminated prior to the call on *wait*, return is immediate.

If *stat_loc* (taken as an integer) is non–zero, 16 bits of information called status are stored in the low order 16 bits of the location pointed to by *stat_loc*. *Status* can be used to differentiate between stopped and terminated child processes. If the child process is terminated, status identifies the cause of termination and passes useful information to the parent. This is accomplished in the following manner:

> If the child process stopped, the high order 8 bits of status will contain the number of the signal that caused the process to stop and the low order 8 bits will be set equal to 0177.

> If the child process terminated due to an *exit* or _*exit* call, the low order 8 bits of status will be zero and the high order 8 bits will contain the low order 8 bits of the argument that the child process passed to *exit*; see *exit*(2).

> If the child process terminated due to a signal, the high order 8 bits of status will be zero and the low order 8 bits will contain the number of the signal that caused the termination. In addition, if the low order seventh bit (i.e., bit 0200) is set, a "core image" will have been produced; see *signal*(2).

If a parent process terminates without waiting for its child processes to terminate, the parent process ID of each child process is set to 1. This means the initialization process inherits the child processes; see *glossary*(9).

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

[ECHILD]   The calling process has no existing unwaited–for child processes. In this case, *wait* returns immediately.

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

NOTE

The behavior of *wait* is affected by setting the **SIGCLD** signal to SIG_IGN. See Special Signals section of *signal*(2)*for*details.

RETURN VALUE

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

SEE ALSO

Exit conditions (**$?**) in sh(1), exec(2), exit(2), fork(2), pause(2), ptrace(2), signal(2).

NAME

write, writev - write on a file

SYNOPSIS

int write (fildes, buf, nbyte)
int fildes;
char ∗buf;
unsigned nbyte;

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

int writev (fildes, iov, iovcnt)
int fildes;
struct iovec *iov;
int iovcnt;

HP–UX COMPATIBILITY

Level: write: HP–UX/RUN ONLY

            writev: HP–UX/STANDARD

Origin: System V

DESCRIPTION

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

Write attempts to write *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the *fildes*. *Writev* performs the same action, but gathers the output data from the *iovlen* buffers specified by the elements of the *iovec* array: iov[0], iov[1], ..., iov[*iovcnt* - 1].

For *writev* the iovec structure is defined as

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

Each *iovec* entry specifies the base address and length of an area in memory where data should be copied from. The *iovec* array maybe at most *MAXIOV* long.

On devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from *write*, the file pointer is incremented by the number of bytes actually written.

On devices incapable of seeking, writing always takes place starting at the device's current position. The value of a file pointer associated with such a device is undefined.

If the O__APPEND flag of the file status flags is set, the file pointer will be set to the end of the file prior to each write.

If a *write* requests that more bytes be written than there is room for (e.g., the *ulimit* (see *ulimit*(2)) or the physical end of a medium), only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return 20. The next write of a non–zero number of bytes will give a failure return (except as noted below).

If the file being written is a pipe (or FIFO), there is a system dependent maximum number of bytes which it can store (NPIPE). The minimum value of NPIPE on any HP–UX system is 4096. In writing a pipe, the following conditions apply:

If the O__NDELAY flag of the file flag word is set:

If *nbyte* is less than or equal to NPIPE and there is sufficient room in the pipe or FIFO, then the *write* is successful and returns the number of bytes written;

If *nbyte* is less than or equal to NPIPE but there is not enough room in the pipe or FIFO, the *write* returns without error, having written nothing, and with a return value of 0.

If *nbyte* is greater than NPIPE the *write* fails and returns -1. [EINVAL]

If the O_NDELAY flag of the file flag word is clear:

the *write* always executes correctly (blocking as necessary) and returns the number of bytes written.

ERRORS

Write will fail and the file pointer will remain unchanged if one of the following conditions is true and **errno** will be set accordingly:

[EBADF]   *Fildes* is not a valid file descriptor open for writing.

[EPIPE and SIGPIPE signal]

An attempt is made to write to a pipe that is not open for reading by any process.

[EFBIG]   The current file position (as set by *lseek*) is less than zero.

[EINTR]   A signal was caught during the *write* system call.

[EDEADLK]   A resource deadlock would occur as a result of this operation (see *lockf*(2)).

In addition, *writev* may return one of the following errors:

[EINVAL]
Iovcnt was less than or equal to 0, or greater then *MAXIOV*.

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

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

Write or *writev* will fail and the file pointer will be updated to reflect the amount of data transferred if one of the following conditions is true and errno will be set accordingly:

[EFBIG]
An attempt was made to write a file that exceeds the process's file size limit or the maximum file size. See *ulimit*(2).

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

RETURN VALUE

Upon successful completion, the number of bytes actually written is returned. Otherwise, **-1** is returned, and *errno* is set to indicate the error.

HARDWARE DEPENDENCIES

Series 500:

If you perform a write operation following an *lseek* past the previous end–of–file, all "unused" bytes from the previous end–of–file up to your new position are zeroed–out before writing your data.

Writev is not implemented on this release.

The size of a pipe (NPIPE) is currently 5120 bytes.

Series 200:

The size of a pipe (NPIPE) is currently 8192 bytes.

Integral PC:
>Under the conditions for which O_NDELAY is set, *nbyte* can be less than or equal to 10240 bytes.

BUGS
>The character special devices, and raw discs in particular, apply constraints on how *write* can be used. See the specific Section 4 entry for details on particular devices.

SEE ALSO
>creat(2), dup(2), lseek(2), open(2), pipe(2), ulimit(2), ustat(2), lockf(2).

Permuted Index

Permuted Index

Permuted Index

Permuted Index

Fe

Permuted Index

Permuted Index

Permuted Index

Permuted Index

Permuted Index