ADMINISTRATOR'S MANUAL
RELEASE 5.0

unix* system

Western Electric

*Trademark of Bell Laboratories

301-926 ISS. 1

UNIX System Administrator's Manual

Release 5.0

June 1982

This manual was set on an AUTOLOGIC, Inc. APS-5 phototypesetter driven by the TROFF formatter operating under the UNIX system.

INTRODUCTION

This manual is intended to supplement the information contained in the *UNIX System User's Manual* and to provide an easy reference volume for those who must administer a UNIX system. Accordingly, only those commands and descriptions deemed appropriate for system administrators have been included here.

This manual is divided into three sections:

1. System Maintenance Commands and Application Programs
7. Special Files.
8. System Maintenance Procedures.

Throughout this volume, each reference of the form *name*(1M), *name*(7), or *name*(8), refers to entries in this manual, while all other references to entries of the form *name*(N), where *N* is a number possibly followed by a letter, refer to entry *name* in Section *N* of the *UNIX System User's Manual*.

Section 1 (*System Maintenance Commands and Application Programs*) contains system maintenance programs such as *fsck*, *mkfs*, etc., which generally reside in the directory **/etc**; these entries carry a sub-class designation of "1M" for cross referencing reasons.

Section 7 (*Special Files*) discusses the characteristics of each system file that actually refers to an input/output device. The names in this section generally refer to device names for the hardware, rather than to the names of the special files themselves.

Section 8 (*System Maintenance Procedures*) discusses crash recovery and boot procedures, facility descriptions, etc.

Each section consists of a number of independent entries of a page or so each. The name of the entry appears in the upper corners of its pages. Entries within each section are alphabetized, with the exception of the introductory entry that begins each section. The page numbers of each entry start at 1. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "major" name.

All entries are based on a common format, not all of whose parts always appear:

> The NAME part gives the name(s) of the entry and briefly states its purpose.
>
> The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (*Commands*):
>
>> **Boldface** strings are literals and are to be typed just as they appear.
>>
>> *Italic* strings usually represent substitutable argument prototypes and program names found elsewhere in the manual (they are underlined in the typed version of the entries).
>>
>> Square brackets **[ ]** around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file", it always refers to a *file* name.
>>
>> Ellipses **...** are used to show that the previous argument prototype may be repeated.
>>
>> A final convention is used by the commands themselves. An argument beginning with a minus **−**, plus **+**, or equal sign **=** is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with **−**, **+**, or **=**.

The **DESCRIPTION** part discusses the subject at hand.

The **EXAMPLE(S)** part gives example(s) of usage, where appropriate.

The **FILES** part gives the file names that are built into the program.

The **SEE ALSO** part gives pointers to related information.

The **DIAGNOSTICS** part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The **WARNINGS** part points out potential pitfalls.

The **BUGS** part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents and a permuted index precede Section 1. The permuted index contains entries from both the *UNIX System User's Manual* and this volume, and on each line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call.

On most systems, all entries are available on-line via the *man*(1) command, q.v.

TABLE OF CONTENTS

1. System Maintenance Commands and Application Programs

7. Special Files

8. System Maintenance Procedures

PERMUTED INDEX

INDEX

INDEX

I N D E X

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

NAME

intro — introduction to system maintenance commands and application programs

DESCRIPTION

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

COMMAND SYNTAX

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

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

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

1

SEE ALSO

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

DIAGNOSTICS

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

BUGS

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

NAME

abt — abort on-line diagnostics

SYNOPSIS

/dgn/bin/abt slot

DESCRIPTION

Abt is a diagnostic command which terminates the diagnostic request indicated by the *slot*. *Slot* is a number from 0 to 9 reported by the Maintenance Input Request Administrator (MIRA) whenever a *dgn*(1M) or *rst*(1M) command is invoked.

SEE ALSO

dgn(1M), dstart(1M), rst(1M).
3B DMERT Output Messages, OM-4C000-01.

WARNING

Diagnostic commands are intended for use only by trained hardware maintenance personnel.

1

NAME

accept, reject − allow/prevent LP requests

SYNOPSIS

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

DESCRIPTION

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

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

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

FILES

/usr/spool/lp/*

SEE ALSO

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

NAME

acctdisk, acctdusg, accton, acctwtmp — overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk

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

/usr/lib/acct/accton [file]

/usr/lib/acct/acctwtmp "reason"

DESCRIPTION

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

Connect time accounting is handled by various programs that write records into **/usr/adm/utmp**, as described in *utmp*(4). The programs described in *acctcon*(1M) convert this file into session and charging records, which are then summarized by *acctmerg*(1M).

1

Process accounting is performed by the UNIX 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 command usage. Current process data may be examined using *acctcom*(1).

Process accounting and connect time accounting (or any accounting records in the format described in *acct*(4)) can be merged and summarized into total accounting records by *acctmerg* (see **tacct** format in *acct*(4)). *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 disk blocks and converts them to total accounting records that can be merged with other accounting records.

Acctdusg reads its standard input (usually from **find / −print**) and computes disk resource consumption (including indirect blocks) 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 disk 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**.

Accton alone turns process accounting off. If *file* is given, it must be the name of an existing file, to which the kernel appends process accounting records (see *acct*(2) and *acct*(4)).

Acctwtmp writes a *utmp*(4) record to its standard output. The record contains the current time and a string of characters that describe the *reason*. A record type of ACCOUNTING is assigned (see *utmp*(4)). *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 ID conversions
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual

/usr/adm/pacct	current process accounting file
/etc/wtmp	login/logoff history file

SEE ALSO

acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).
UNIX Accounting System in the *UNIX System Administrator's Guide*.

1

NAME

acctcms — command summary from per-process accounting records

SYNOPSIS

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

DESCRIPTION

Acctcms reads one or more *files*, normally in the form described in *acct*(4). 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, and "hog factor", 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.

1

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), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

NAME

acctcon1, acctcon2 — connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]

/usr/lib/acct/acctcon2

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 terminiation of the login shell generate a logoff records, so that the number of logoffs is often three to four times the number of sessions. See *init*(1M) and *utmp*(4).

−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*(4)).

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), runacct(1M), acct(2), acct(4), utmp(4).

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] . . .

DESCRIPTION

Acctmerg reads its standard input and up to nine additional files, all in the **tacct** format (see *acct*(4)), or an ASCII version thereof. It merges these inputs by adding records whose keys (normally 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.

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

acctmerg −v <file1 >file2
edit file2 as desired ...
acctmerg −a <file2 >file1

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

NAME

acctprc1, acctprc2 − process accounting

SYNOPSIS

/usr/lib/acct/acctprc1 [ctmp]

/usr/lib/acct/acctprc2

DESCRIPTION

Acctprc1 reads input in the form described by *acct*(4), 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 64-byte 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), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

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).

NAME

chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct — shell procedures for accounting

SYNOPSIS

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

/usr/lib/acct/ckpacct [blocks]

/usr/lib/acct/dodisk

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct number

/usr/lib/acct/nulladm file

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily [mmdd]

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

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

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

/usr/lib/acct/startup

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

1

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.

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 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. Previous daily reports are cleaned up and therefore inaccessible after each invocation of *monacct*.

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.

1

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/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

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

NAME

acuset — connect ACUs and communication lines

SYNOPSIS

/etc/**acuset** filen

DESCRIPTION

The *acuset* command provides a means for dynamically associating *tn8* ACU minor devices with communication lines. The connections are specified in *filen*; the format of this file is described below. Until these connections have been made, a program cannot dial out on an ACU. The connections can be changed dynamically. The only processes affected are those trying to dial out on the connections being changed.

Filen consists of one or more lines of the following form:

/dev/acu? unit port [line]

where */dev/acu?* is the ACU minor device name, *unit* is the ACU unit number, *port* is the port number, and *line* is the optional line number in an ACU sharing arrangement.

Here is a sample file for four ACUs with no sharing arrangements.

```
/dev/acu0 0 0
/dev/acu1 0 1
/dev/acu2 0 2
/dev/acu3 0 3
```

Here is a sample file for one ACU in a sharing arrangement with twelve data sets.

```
/dev/acu0 0 0 1
/dev/acu1 0 0 2
/dev/acu2 0 0 3
/dev/acu3 0 0 4
/dev/acu4 0 0 5
/dev/acu5 0 0 6
/dev/acu6 0 0 9
/dev/acu7 0 0 10
/dev/acu8 0 0 11
/dev/acu9 0 0 12
/dev/acu10 0 0 13
/dev/acu11 0 0 14
```

The line numbers correspond to physical slot numbers in the ACU sharing hardware.

SEE ALSO

acu(7).

NAME

atb — attach to an Address Translation Buffer

SYNOPSIS

/etc/atb command args

DESCRIPTION

An Address Translation Buffer (ATB) is an associative memory that is used to speed up the conversion of a virtual memory address to a physical memory address. The 3B20S contains eight ATBs. ATB-0 is used by the operating system, ATB-1 is shared by all user processes, ATB-2 through ATB-7 are normally unused.

The *atb* command "attaches" itself to an unused ATB, i.e. becomes the sole process using it, and then overlays itself with *command.* A process "attached to" an ATB will run slightly faster if measured over a long period of time.

WARNING

Super-user privileges are required.

SEE ALSO

sys3b(2).

1

NAME

bcopy — interactive block copy

SYNOPSIS

/etc/bcopy

DESCRIPTION

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

The following questions are asked:

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

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

Two consecutive carriage returns terminate *bcopy*.

SEE ALSO

cpio(1), dd(1).

NAME

brc, bcheckrc, rc, powerfail — system initialization shell scripts

SYNOPSIS

/etc/brc

/etc/bcheckrc

/etc/rc

/etc/powerfail

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 appropriate. It also logs the fact that a power failure occurred.

SEE ALSO

init(1M), shutdown(1M), inittab(4), vpm(7).

NAME

checkall — faster file system checking procedure

SYNOPSIS

/etc/checkall

DESCRIPTION

The *checkall* procedure is a prototype and must be modified to suit local conditions. The following will serve as a example:

```
# check the root file system by itself
fsck /dev/rp0

# dual fsck of drives 0 and 1
dfsck /dev/rrp[12345] — /dev/rrp11
```

In the above example (where **/dev/rrp11** is 320K blocks and **/dev/rrp[12345]** are each 65K or less), a previous sequential *fsck* took 19 minutes. The *checkall* procedure takes 11 minutes.

1

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

Due to the file system load balancing required for dual checking, the *dfsck* command should always be executed through the *checkall* shell procedure.

In a practical sense, the file systems are divided up as follows:

```
dfsck file_systems_on_drive_0 — file_systems_on_drive_1
dfsck file_systems_on_drive_2 — file_systems_on_drive_3
  . . .
```

A three drive system can be handled by this more concrete example (assumes two large file systems per drive):

```
dfsck /dev/dsk31 /dev/dsk[14] — /dev/dsk1[14] /dev/dsk34
```

Note that the first drive 3 file system is first in the *filesystems1* list and is last in the *filesystems2* list assuring that references to that drive will not overlap at execution time.

WARNINGS

1. Do not use *dfsck* to check the *root* file system.
2. On a check that requires a scratch file (see **−t** above), be careful not to use the same temporary file for the two groups (this is sure to scramble the file systems).
3. The *dfsck* procedure is useful only if the system is set up for multiple physical I/O buffers.

SEE ALSO

fsck(1M).
Setting up UNIX in the *UNIX System Administrator's Guide*.

NAME

chmap — change the diagnostic spooler map file

SYNOPSIS

/dgn/bin/chmap

DESCRIPTION

Chmap informs the on-line diagnostic spooler to reread the spooler map file. The spooler map file, **/dgn/dgnc/map**, contains a list of at most 10 file names. Each file name is contained on a separate line. All diagnostic output messages will be appended to each file that is specified within the map file. If the first line of the map file is the character string **stamp**, then all diagnostic output messages are prefixed with a time stamp.

FILES

/dgn/dgnc/map

WARNING

Diagnostic commands are intended for use only by trained hardware maintenance personnel.

1

NAME

chroot − change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

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

Notice that:

chroot newroot command >x

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

This command is restricted to the super-user.

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

SEE ALSO

chdir(2).

BUGS

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

NAME

clri — clear i-node

SYNOPSIS

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

DESCRIPTION

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

Read and write permission is required on the specified *file-system* device. The i-node becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to *zap* an i-node which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated i-node, so the whole cycle is likely to be repeated again and again.

1

SEE ALSO

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

BUGS

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

NAME

config − configure a UNIX system

SYNOPSIS

/etc/config [system [master]]

DESCRIPTION

Config is a program that takes a description of a UNIX system and generates the necessary configuration information for the operating system. This includes hardware, driver and parameter specifications. *System* is used for the description file. The default file is **/etc/system**. Information defining the allowable configuration is kept in the *master* file. The default file is **/etc/master**.

The user must supply the system definition file; the supplied version contains the minimal configuration for the processor.

FILES

/etc/system	default system description file
/etc/master	default input master device table
conf.c	output configuration table file

SEE ALSO

sysdef(1M), master(4), system(4).
Setting up UNIX in the *UNIX System Administrator's Guide*.

DIAGNOSTICS

Diagnostics are routed to the standard error output and are self-explanatory.

1

NAME

config — configure a UNIX system

SYNOPSIS

/etc/config [**−n**] [**−t**] [**−l** file] [**−c** file] [**−m** file] dfile

DESCRIPTION

Config is a program that takes a description of a UNIX system and generates two files. One file provides information regarding the interface between the hardware and device handlers. The other file is a C program defining the configuration tables for the various devices on the system.

The **−n** option produces a non-separated **I** and **D** space **low.s** core image for the PDP-11 (this is for small systems, i.e., PDP11/23 and 11/34).

The **−l** option specifies the name of the hardware interface file; **low.s** is the default name on the PDP-11; **univec.c** is the default name on the VAX-11.

The **−c** option specifies the name of the configuration table file; **conf.c** is the default name.

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

The **−t** option requests a short table of major device numbers for character and block type devices. This can facilitate the creation of special files.

The user must supply *dfile*; it must contain device information for the user's system. This file is divided into two parts. The first part contains physical device specifications. The second part contains system-dependent information. Any line with an asterisk (*) in column 1 is a comment.

All configurations are assumed to have the following devices:

> one DL11 (for the system console)
> one KW11-L line clock or KW11-P programmable clock

with standard interrupt vectors and addresses. These two devices *must not* be specified in *dfile*. Note that UNIX needs only one clock, but can handle both types.

First Part of *dfile*

Each line contains four or five fields, delimited by blanks and/or tabs in the following format:

> devname vector address bus number

where *devname* is the name of the device (as it appears in the **/etc/master** device table), *vector* is the interrupt vector location (octal), *address* is the device address (octal), *bus* is the bus request level (4 through 7), and *number* is the number (decimal) of devices associated with the corresponding controller; *number* is optional, and if omitted, a default value which is the maximum value for that controller is used.

There are certain drivers that may be provided with the system, that are actually *pseudo-device* drivers; that is, there is no real hardware associated with the driver. Drivers of this type are identified on their respective manual entries. When these devices are specified in the description file, the interrupt *vector*, device *address*, and *bus* request level must all be zero.

If the device is a VAX-11 massbus adapter, then *vector* is the adapter nexus number, and *address* must be zero.

Second Part of *dfile*

The second part contains three different types of lines. Note that *all* specifications of this part *are required*, although their order is arbitrary.

1. *Root/pipe/dump device specification*

   Three lines of three fields each:

root	devname	minor
pipe	devname	minor
dump	devname	minor

   where *minor* is the minor device number (in octal).

2. *Swap device specification*

   One line that contains five fields as follows:

   swap devname minor swplo nswap

   where *swplo* is the lowest disk block (decimal) in the swap area and *nswap* is the number of disk blocks (decimal) in the swap area.

3. *Parameter specification*

   Several lines of two fields each as follows (*number* is decimal):

buffers	number	
sabufs	number	(zero on the VAX-11)
inodes	number	
files	number	
mounts	number	
coremap	number	(PDP-11 only)
swapmap	number	
calls	number	
procs	number	
maxproc	number	
texts	number	
clists	number	
hashbuf	number	
physbuf	number	
x25links	number	
x25bufs	number	
x25map	number	
x25bytes	number	
iblocks	number	(PDP-11 only)
power	0 or 1	
mesg	0 or 1	
sema	0 or 1	
shmem	0 or 1	(VAX-11 only)
maus	0 or 1	(PDP-11 only)

EXAMPLE

To configure a PDP-11/70 system with the following devices:

one RP06 disk drive controller with 6 drives
one DH11 asynchronous multiplexer with 16 lines (default number)
one DM11 modem control with 16 lines (for the DH11)
one DH11 asynchronous multiplexer with 8 lines
one DM11 modem control with 8 lines (for the DH11)
one LP11 line printer
one TU16 tape drive controller with 2 drives
one DL11 asynchronous interface

Note that UNIX only supports DH11 units that require corresponding DM11 units. It is wise to specify them in DH-DM pairs to facilitate understanding the configuration. Note also that, in the preceding case, the DL11 that is specified is *in addition* to the DL11 that was part of the initial system. We must also specify the following parameter information:

root device is an RP06 (drive 0, section 0)
pipe device is an RP06 (drive 0, section 0)
swap device is an RP06 (drive 1, section 4),
    with a swplo of 6000 and an nswap of 2000
dump device is a TU16 (drive 0)
number of buffers is 35
number of *system addressable* buffers is 12
number of processes is 150
maximum number of processes per user ID is 25
number of mounts is 8
number of inodes is 120
number of files is 120
number of calls is 30
number of texts is 35
number of character buffers is 150
number of coremap entries is 50
number of swapmap entries is 50
power fail recovery is to be included
messages are to be included
semaphores are to be included
one psuedo device driver for the Operating System Profiler

The actual system configuration would be specified as follows:

```
rp06        254     776700   5      6
dh11        320     760020   5
dm11        300     770500   4
dh11        330     760060   5      8
dm11        304     770510   4      8
lp11        200     775514   5
tu16        224     772440   5      2
dl11        350     775610   5
prf         0       0        0
root        rp06    0
pipe        rp06    0
swap        rp06    14       6000   2000
dump        tu16    0
* Comments may be inserted in this manner
buffers     35
sabufs      12
procs       150
maxproc     25
mounts      8
inodes      120
files       120
calls       30
texts       35
clists      150
coremap     50
swapmap     50
power       1
msg         1
sema        1
```

1

FILES

/etc/master	default input master device table
low.s	default output hardware interface file for PDP-11
univec.c	default output hardware interface file for the VAX-11
conf.c	default output configuration table file

SEE ALSO

sysdef(1M), master(4).
Setting up UNIX in the *UNIX System Administrator's Guide*.

DIAGNOSTICS

Diagnostics are routed to the standard output and are self-explanatory.

BUGS

The **−t** option does not know about devices that have aliases. For example, an TE16 (an alias for an TU16) will show up as an TU16; however, the major device numbers are always correct.

1

NAME

crash — examine system images

SYNOPSIS

/etc/crash [system] [namelist]

DESCRIPTION

Crash is an interactive utility for examining an operating system core image. It has facilities for interpreting and formatting the various control structures in the system and certain miscellaneous functions that are useful when perusing a dump.

The arguments to *crash* are the file name where the *system* image can be found and a *namelist* file to be used for symbol values.

The default values are **/dev/mem** and **/unix**; hence, *crash* with no arguments can be used to examine an active system. If a *system* image file is given, it is assumed to be a system core dump and the default process is set to be that of the process running at the time of the crash. This is determined by a value stored in a fixed location by the dump mechanism.

COMMANDS

Input to *crash* is typically of the form:

command [options] [structures to be printed].

When allowed, *options* will modify the format of the printout. If no specific structure elements are specified, all valid entries will be used. As an example, **proc — 12 15 3** would print process table slots 12, 15 and 3 in a long format, while **proc** would print the entire process table in standard format.

In general, those commands that perform I/O with addresses assume hexadecimal on 32-bit machines and octal on 16-bit machines.

The current repertory consists of:

user [list of process table entries]
Aliases: **uarea, u_area, u.**
Print the user structure of the named process as determined by the information contained in the process table entry. If no entry number is given, the last executing process's information will be printed. Swapped processes produce an error message.

trace [**−r**] [list of process table entries]
Aliases: **t.**
Generate a kernel stack trace of the current process. If the **−r** option is used, the trace begins at the saved stack frame pointer in **kfp**. Otherwise the trace starts at the bottom of the stack and attempts to find valid stack frames deeper in the stack. If no entry number is given, the last executing process's information will be printed.

kfp [stack frame pointer]
Aliases: **r5, fp.**
Print the program's idea of the start of the current stack frame (set initially from a fixed location in the dump) if no argument is given, or set the frame pointer to the supplied value.

stack [list of process table entries]
Aliases: **stk, s, kernel, k.**
Format a dump of the kernel stack of a process. The addresses shown are virtual system data addresses rather than true physical locations. If no entry number is given, the last executing process's information will be printed.

proc [−[r]] [list of process table entries]
Aliases: **ps**, **p**.
Format the process table. The **−r** option causes only runnable processes to be printed. The − alone generates a longer listing.

pcb [list of process table entries]
Print the process control block of the current process. The process control block is a part of the user area (VAX-11/780 only). If no entry number is given, the last executing process's information will be printed.

inode [−] [list of inode table entries]
Aliases: **ino**, **i**.
Format the inode table. The − option will also print the inode data block addresses.

file [list of file table entries]
Aliases: **files**, **f**.
Format the file table.

mount [list of mount table entries]
Aliases: **mnt**, **m**.
Format the mount table.

1

text [list of text table entries]
Aliases: **txt**, **x**.
Format the text table.

tty [type] [−] [list of tty entries]
Aliases: **term**, **dz**, **dh**.
Print the tty structures. The *type* argument determines which structure will be used (such as **kl**, **dh**, **dz**, or **dzb**; the last *type* is remembered). The − option prints the *stty*(1) parameters for the given line.

stat Print certain statistics found in the dump. These include the panic string (if a panic occurred), time of crash, system name, and the registers saved in low memory by the dump mechanism.

var Aliases: **tunables**, **tunable**, **tune**, **v**.
Print the tunable system parameters.

buf [list of buffer headers]
Aliases: **hdr**, **bufhdr**.
Format the system buffer headers.

buffer [format] [list of buffers]
Alias: **b**.
Print the data in a system buffer according to *format*. If *format* is omitted, the previous *format* is used. Valid formats include **decimal**, **octal**, **hex**, **character**, **byte**, **directory**, **inode**, and **write**. The last creates a file in the current directory (see *FILES*) containing the buffer data.

callout Aliases: **calls**, **call**, **c**, **timeout**, **time**, **tout**.
Print all entries in the callout table.

map [list of map names]
Format the named system map structures.

nm [list of symbols]
Print symbol value and type as found in the *namelist* file.

ts [list of text addresses]
Find the closest text symbols to the given addresses.

ds [list of data addresses]
Find the closest data symbols to the given addresses.

od [symbol name or address] [count] [format]
Aliases: **dump, rd**.
Dump *count* data values starting at the symbol value or address given according to *format*. Allowable formats are **octal, longoct, decimal, longdec, character, hex,** or **byte**.

! Escape to shell.

q Exit from *crash*.

? Print synopsis of commands.

ALIASES

There are built in aliases for many of the *formats* as well as those listed for the commands. Some of them are:

byte	b.
character	char, c.
decimal	dec, e.
directory	direct, dir, d.
hexadecimal	hexadec, hex, h, x.
inode	ino , i.
longdec	ld, D.
longoct	lo, O.
octal	oct, o.
write	w.

1

FILES

/usr/include/sys/*.h	header files for table and structure info
/dev/mem	default system image file
/unix	default namelist file
buf.#	files created containing buffer data

SEE ALSO

mount(1M), nm(1), ps(1), sh(1), stty(1), crash(8).

BUGS

Most flags are abbreviated and will have little meaning to the uninitiated user. A source listing of the system header files at hand would be most useful while using *crash*.

Stack tracing of the current process on a running system doesn't work.

NAME

cron — clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times according to the instructions in the file **/usr/lib/crontab**. Because *cron* never exits, it should be executed only once. This is best done by running *cron* from the initialization process through the file **/etc/rc** (see *init*(1M)).

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

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

Each of these patterns may contain:

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

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

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

FILES

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

SEE ALSO

init(1M), sh(1).

DIAGNOSTICS

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

BUGS

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

NAME

dcopy — copy file systems for optimal access time

SYNOPSIS

/etc/dcopy [**−s**X] [**−an**] [**−d**] [**−v**] [**−f**fsize:isize] inputfs outputfs

DESCRIPTION

Dcopy copies file system *inputfs* to *outputfs*. *Inputfs* is the existing file system; *outputfs* is an appropriately sized file system, to hold the reorganized result. For best results *inputfs* should be the raw device and *outputfs* should be the block device. *Dcopy* should be run on unmounted file systems (in the case of the root file system, copy to a new pack). With no arguments, *dcopy* copies files from *inputfs* compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap. The possible options are

−s*X* supply device information for creating an optimal organization of blocks in a file. The forms of *X* are the same as the **−s** option of *fsck*(1M).

−a*n* place the files not accessed in *n* days after the free blocks of the destination file system (default for *n* is 7). If no *n* is specified then no movement occurs.

−d leave order of directory entries as is (default is to move subdirectories to the beginning of directories).

−v currently reports how many files were processed, and how big the source and destination freelists are.

−f*fsize*[:*isize*]
specify the *outputfs* file system and inode list sizes (in blocks). If not given, the values from the *inputfs* are used.

Dcopy catches interrupts and quits and reports on its progress. To terminate *dcopy*, send a quit signal and *dcopy* will no longer catch interrupts or quits. *Dcopy* also attempts to modify its command line arguments so its progress can be monitored with *ps*(1).

SEE ALSO

fsck(1M), mkfs(1M), ps(1).

1

NAME

devnm — device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

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

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

EXAMPLE

The command:

```
/etc/devnm /usr
```

produces

```
rp1 /usr
```

if **/usr** is mounted on **/dev/rp1**.

FILES

/dev/rp*, /dev/dsk*
/etc/mnttab

SEE ALSO

bcheckrc(1M), setmnt(1M).

NAME

df − report number of free disk blocks

SYNOPSIS

df [**−t**] [**−f**] [file-systems]

DESCRIPTION

Df prints out the number of free blocks and free i-nodes available for on-line file systems by examining the counts kept in the super-blocks; *file-systems* may be specified either by device name (e.g., **/dev/dsk1**) or by mounted directory name (e.g., **/usr**). If the *file-systems* argument is unspecified, the free space on all of the mounted file systems is printed.

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

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

FILES

/dev/dsk*
/etc/mnttab

SEE ALSO

fs(4), mnttab(4).

1

NAME

dgn − initiate on-line diagnostics

SYNOPSIS

/dgn/bin/dgn name unit [options]

DESCRIPTION

Dgn initiates on-line diagnostics on the device indicated by *name* and *unit*. *Options* is a string of keyword parameters separated from each other by white space. *Dgn* parses the parameter string *options* and verifies that each keyword parameter does not contain any missing components or values that are out-of-range.

The following *options* are recognized, each as a separate argument:

raw Print the diagnostic results of every phase and all failures. By default, only the final results and the first five failures of each failing phase will be printed.

ucl Unconditionally execute the diagnostic with no early termination (i.e., the diagnostic will be run to completion in spite of failures). By default, the diagnostic will terminate after the first failing phase.

1

ph=*x*[−*y*] Execute only the specified phase numbers. May be either a single decimal number or a range of numbers. The letter *x* denotes the beginning phase number and *y* the ending phase number.

rpt=*x* Repeats the diagnostic *x* times. The maximum value allowed is 256.

tlp Executes the Trouble Location Procedure at the conclusion of the diagnostic. This process analyzes diagnostic failures and generates a weighted list of faulty circuit packs. This option must *not* be used in conjunction with the **ucl** option.

file=*filename* Routes all output messages into a file named *filename*, instead of the user's terminal. *Filename* is opened for appending and is relative to the directory **/dgn/dgnc** unless a full pathname is specified.

cont This option is effective only when *name* and *unit* is an IOP By default, after an IOP is diagnosed, all of its Peripheral Controllers (PCs) are diagnosed automatically. The **cont** option causes only the IOP diagnostics to be run. Note that MHDs are never automatically diagnosed when *name* and *unit* is a DFC.

hu=*name unit* This *option* allows a helper unit identified by *name* and *unit* to be specified. For example, when diagnosing the magnetic tape controller (i.e., UN32), a diagnostic test tape with a write ring must be mounted on the specified helper unit. The following example shows how one might invoke diagnostics using the helper unit option:

```
dgn un32 0 ph=5 hu=mt 2
```

SEE ALSO

rmv(1M), rst(1M).
3B DMERT Output Messages, OM-4C000-01.

WARNING

Diagnostic commands are intended for use only by trained hardware maintenance personnel.

NAME

don, doff, disp — device logically on, logically off or display status

SYNOPSIS

/etc/don unit unitnum [pump-file]
/etc/don all [sysfile]

/etc/doff unit unitnum

/etc/disp unit unitnum
/etc/disp all [sysfile]
/etc/disp all − [sysfile]
/etc/disp all **−i** [sysfile]
/etc/disp all **−c** [incr] [sysfile]

DESCRIPTION

Don restores to service (logically connects to the system) a hardware *unit*. *Unitnum* is the unit number of that particular *unit*. For example,

don tn4 1

restores to service the **tn4** whose unit number is **1**. *Pump-file* in directory **/firm** is pumped into that specified device. The default pump-file is **unit**. *Don all* reads *sysfile*, default is **/etc/system**, and performs a *don unit unitnum [pump-file]* on each IOP, DFC and associated peripherals listed in that file. Lines prefixed with a # (comment) or ! (no-pump) will be skipped. *Don all* is primarily useful when the system is brought to multi-user mode.

Doff removes from service (logically disconnects) *unitnum* of type *unit*.

Disp prints the status of *unitnum* of type *unit* (e.g., "out of service", "undergoing diagnostics"). *Disp all* reads *sysfile*, default is **/etc/system**, and performs a *disp unit unitnum* on each IOP, DFC and associated peripherals listed in that file. Lines prefixed with a # are skipped. Output is in the form of:

unit-unitnum chan dev status

for an IOP or DFC and

slot unit-unitnum status

for each device on that IOP or DFC. If the − argument is given, a status diagram of the hardware is printed on the terminal. Known *terminals* from the environment parameter **$TERM** (see *environ*(5)) are:

$TERM Value	*Terminal Type*
4420	TTY 4420
vt100	VT 100
2621	HP 2621
2645	HP 2645

Peripherals out of service are displayed in inverse video and invalid entries are blinked (shown by * and |, respectively, under the device slot on Hewlett Packard terminals). The **i** flag makes the program interactive; the **c** flag redraws the status of the machine every *incr* seconds, default is 30 (a **?** is printed under each entry that has changed status since the invocation of the program).

FILES

/etc/master default table for hardware specifications
/etc/system default system configuration file

SEE ALSO

config(1M), master(4), system(4).

1

NAME

dskfmt, dskvfy − format and verify disk packs

SYNOPSIS

/etc/dskfmt unit [start [end]]

/etc/dskvfy unit [start [end]]

DESCRIPTION

Dskfmt formats a disk pack and *dskvfy* verifies the format of a disk pack. *Unit* specifies the unit number of the disk drive to be used. Note that this drive must be in the out of service state and the controller for this drive must be in the in service state. *Start* and *end* specify the starting and ending cylinders, inclusive, for the operation to be done. If no arguments are given the default for *start* is 0 and for *end* is the last cylinder on the disk.

FILES

/dev/dgn/mhd
/dev/dgn/dfc

SEE ALSO

dsk(7).

DIAGNOSTICS

If *dskvfy* finds an error in the format of the disk the numbers of the cylinders found to be bad will be printed.

NAME

dstart, dstop, dstat — start, stop and find status of on-line diagnostics

SYNOPSIS

/dgn/bin/dstart

/dgn/bin/dstop

/dgn/bin/dstat

DESCRIPTION

Dstart enables on-line diagnostics to be run by automatically starting both the Output Message Spooler Program (SPOOLER) and the Maintenance Request Input Administrator Program (MIRA), respectively. These two diagnostics programs are only started if they're not already running. Also, both program's process ID numbers are reported in parentheses. On-line diagnostics require that both these programs be started before any diagnostics requests are accepted.

The spooler arranges for all diagnostic output to be logged in **/dgn/dgnc/log**. When the spooler is restarted, **/dgn/dgnc/log** is moved to **/dgn/dgnc/oldlog** and a new **/dgn/dgnc/log** is started. All output is also appended to each file mentioned in the map file, **/dgn/dgnc/map** (see *chmap*(1M)).

Dstop stops both the SPOOLER and MIRA programs only if they are currently running. Otherwise, no explicit action is taken. In either case, an appropriate message is reported indicating what action did occur.

Dstat reports the current status of both the SPOOLER and MIRA diagnostic programs. If both programs are currently running a message indicating that they are running is reported along with their respective process ID numbers. Otherwise, a message indicating that they are not running is reported.

FILES

/dgn/dgnc/log	spooler output message log.
/dgn/dgnc/map	list of file names for routing spooler output messages.

SEE ALSO

dgn(1M), rmv(1M), rst(1M).

WARNING

Diagnostic commands are intended for use only by trained hardware maintenance personnel.

1

NAME

emulcntrl — perform 3270 emulation control functions

SYNOPSIS

/etc/emulcntrl device function [arg]

DESCRIPTION

Emulcntrl is used to communicate with the 3270 emulation controller driver. *Device* is the name of the emulation controller to use (e.g., **/dev/emc0**). *Function* is a string indicating the operation to perform. Some functions require an additional argument *arg*. Valid *function* strings and additional arguments are as follows:

on Start the 3270 emulation script associated with *device*.

off Stop the 3270 emulation script associated with *device*.

ascii This 3270 emulation controller is to be ASCII. The ASCII 3270 script must be loaded on the associated physical device.

ebcdic This 3270 emulation controller is to be EBCDIC. The EBCDIC 3270 script must be loaded on the associated physical device. Controllers are EBCDIC by default.

pollid Change the POLL character for this controller to *arg*. *Arg* must be the decimal value of the character desired.

selid Change the SELECT character for this controller to *arg*. *Arg* must be the decimal value of the character desired.

delay Set the time delay before transmitting EOT's to *arg*/10 seconds. The default is 2 seconds.

trace Force the script to trace certain events.

Except for starting and stopping, these functions should be performed **before** starting the script.

FILES

/dev/emc?	3270 emulation controller devices
/lib/a3270scr	ASCII 3270 script
/lib/e3270scr	EBCDIC 3270 script

SEE ALSO

emulload(1M), emulstat(1M), vpmset(1M), emulio(7).

DIAGNOSTICS

Emulcntrl fails if the function cannot be performed, e.g., changing the POLL character on a running controller.

NAME

emulload — load and start 3270 emulation script

SYNOPSIS

/etc/emulload

DESCRIPTION

The *emulload* command file is used to load the 3270 emulation protocol script into the physical device, set the proper options, and start execution of the script. *Emulload* will need local modification to use the proper hardware device, set the proper options, or to start more than one emulation controller.

As distributed, *emulload* contains the following:

```
/etc/vpmset /dev/emc0 /dev/un53.0
/etc/emulcntrl /dev/emc0 ascii
/etc/vpmstart /dev/un53.0 6 /lib/a3270scr
/etc/emulcntrl /dev/emc0 on
```

This command file will connect the emulation controller and physical line, set the controller to ASCII mode, load the ASCII emulation script, and start execution of the script. Other controller options are described in *emulcntrl*(1M).

The **/etc/rc** file should call **/etc/emulload** when going to multi-user state. The **/etc/shutdown** file should halt any controllers that were started in **/etc/rc**. For example, the entry in **/etc/shutdown** for the *emulload* command shown above would be:

```
/etc/emulcntrl /dev/emc0 off
```

FILES

/dev/emc?	3270 emulation controller devices
/lib/a3270scr	ASCII 3270 script
/lib/e3270scr	EBCDIC 3270 script

SEE ALSO

emulcntrl(1M), emulstat(1M), vpmset(1M), emulio(7).

NAME

emulstat — get 3270 emulation controller/terminal status

SYNOPSIS

/etc/**emulstat** device

DESCRIPTION

Emulstat reports the status of *device*. *Device* may be a 3270 emulation controller or terminal. The status is reported as hexadecimal values representing the following:

flags The value of the *device* flags. Possible *flag* values for controllers or terminals are given in *emulio*(7).

code A value used by the driver for indicating certain error conditions or return values.

station For terminals, this is the value of the *station* (controller) identification byte. For controllers, it is the value of the *Polling* byte used by the remote system.

terminal For terminals, this is the value of the *terminal* identification byte. For controllers, it is the value of the *Selection* byte used by the remote system.

dev This value indicates the physical hardware device being used by this controller (e.g., the *un53* minor device number).

Emulstat will fail if the controller has not been started.

FILES

/dev/emc?	3270 emulation controller devices
/dev/emt*	3270 emulation terminal devices
/lib/a3270scr	ASCII 3270 script
/lib/e3270scr	EBCDIC 3270 script

SEE ALSO

emulcntrl(1M), emulload(1M), vpmset(1M), emulio(7).

NAME

errdead — extract error records from dump

SYNOPSIS

/etc/errdead dumpfile [namelist]

DESCRIPTION

When hardware errors are detected by the system, an error record that contains information pertinent to the error is generated. If the error-logging daemon *errdemon*(1M) is not active or if the system crashes before the record can be placed in the error file, the error information is held by the system in a local buffer. *Errdead* examines a system dump (or memory), extracts such error records, and passes them to *errpt*(1M) for analysis.

The *dumpfile* specifies the file (or memory) that is to be examined. The system namelist is specified by *namelist*; if not given, **/unix** is used.

FILES

/unix	system namelist
/usr/bin/errpt	analysis program
/usr/tmp/errXXXXXX	temporary file

DIAGNOSTICS

Diagnostics may come from either *errdead* or *errpt*. In either case, they are intended to be self-explanatory.

SEE ALSO

errdemon(1M), errpt(1M).

1

NAME

errdemon — error-logging daemon

SYNOPSIS

/usr/lib/errdemon [file]

DESCRIPTION

The error logging daemon *errdemon* collects error records from the operating system by reading the special file **/dev/error** and places them in *file*. If *file* is not specified when the daemon is activated, **/usr/adm/errfile** is used. Note that *file* is created if it does not exist; otherwise, error records are appended to it, so that no previous error data is lost. No analysis of the error records is done by *errdemon*; that responsibility is left to *errpt*(1M). The error-logging daemon is terminated by sending it a software kill signal (see *signal*(2)). Only the super-user may start the daemon, and only one daemon may be active at any time.

FILES

/dev/error	source of error records
/usr/adm/errfile	repository for error records

DIAGNOSTICS

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

SEE ALSO

errpt(1M), errstop(1M), kill(1), err(7).

1

NAME

errpt — process a report of logged errors

SYNOPSIS

errpt [options] [files]

DESCRIPTION

Errpt processes data collected by the error logging mechanism (*errdemon*(1M)) and generates a report of that data. The default report is a summary of all errors posted in the files named. Options apply to all files and are described below. If no files are specified, *errpt* attempts to use **/usr/adm/errfile** as *file*.

A summary report notes the options that may limit its completeness, records the time stamped on the earliest and latest errors encountered, and gives the total number of errors of one or more types. Each device summary contains the total number of unrecovered errors, recovered errors, errors unabled to be logged, I/O operations on the device, and miscellaneous activities that occurred on the device. The number of times that *errpt* has difficulty reading input data is included as read errors.

Any detailed report contains, in addition to specific error information, all instances of the error logging process being started and stopped, and any time changes (via *date*(1)) that took place during the interval being processed. A summary of each error type included in the report is appended to a detailed report.

1

A report may be limited to certain records in the following ways:

−s *date* Ignore all records posted earlier than *date*, where *date* has the form *mmddhhmmyy*, consistent in meaning with the *date*(1) command.

−e *date* Ignore all records posted later than *date*, whose form is as described above.

−a Produce a detailed report that includes all error types.

−d *devlist* A detailed report is limited to data about devices given in *devlist*, where *devlist* can be one of two forms: a list of device identifiers separated from one another by a comma, or a list of device identifiers enclosed in double quotes and separated from one another by a comma and/or more spaces. *Errpt* is familiar with the common form of identifiers (e.g., rs03, RS04, hs; see Section 7 of this volume). For the 3B20S the devices for which errors are logged are DFC, IOP, and MT. For Digital Equipment Corporation machines, the (block) devices for which errors are logged are RP03, RP04, RP05, RP06, RP07, RS03, RS04, TS11, TU10, TU16, TU78, RK05, RK06, RK07, RM05, RM80, and RF11. Additional identifiers are **int** and **mem** which include detailed reports of stray-interrupt and memory-parity type errors respectively.

−p *n* Limit the size of a detailed report to *n* pages.

−f In a detailed report, limit the reporting of block device errors to unrecovered errors.

FILES

/usr/adm/errfile default error file

SEE ALSO

errdemon(1M), errfile(4).

NAME

errstop — terminate the error-logging daemon

SYNOPSIS

/etc/errstop [namelist]

DESCRIPTION

The error-logging daemon *errdemon*(1M) is terminated by using *errstop*. This is accomplished by executing *ps*(1) to determine the daemon's identity and then sending it a software kill signal (see *signal*(2)); **/unix** is used as the system namelist if none is specified. Only the super-user may use *errstop*.

FILES

/unix default system namelist

DIAGNOSTICS

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

SEE ALSO

errdemon(1M), ps(1), kill(2).

1

NAME

ff − list file names and statistics for a file system

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

Ff reads the i-list and directories of the *special* file, assuming it to be a file system, saving i-node data for files which match the selection criteria. Output consists of the path name for each saved i-node, plus any other file information requested using the print *options* below. Output fields are positional. The output is produced in i-node order; fields are separated by tabs. The default line produced by *ff* is:

path-name i-number

With all *options* enabled, output fields would be:

path-name i-number size uid

The argument *n* in the *option* descriptions that follow is used as a decimal integer (optionally signed), where +*n* means more than *n*, −*n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hour period.

−I	Do not print the i-node number after each path name.
−l	Generate a supplementary list of all path names for multiply linked files.
−p *prefix*	The specified *prefix* will be added to each generated path name. The default is ..
−s	Print the file size, in bytes, after each path name.
−u	Print the owner's login name after each path name.
−a *n*	Select if the i-node has been accessed in *n* days.
−m *n*	Select if the i-node has been modified in *n* days.
−c *n*	Select if the i-node has been changed in *n* days.
−n *file*	Select if the i-node has been modified more recently than the argument *file*.
−i *i-node-list*	Generate names for only those i-nodes specified in *i-node-list*.

EXAMPLES

To generate a list of the names of all files on a specified file system:

ff −I /dev/diskroot

To produce an index of files and i-numbers which are on a file system and have been modified in the last 24 hours:

ff −m −1 /dev/diskusr > /log/incbackup/usr/tuesday

To obtain the path names for i-nodes 451 and 76 on a specified file system:

ff −i 451,76 /dev/rrp7

SEE ALSO

finc(1M), find(1), frec(1M), ncheck(1M).

BUGS

Only a single path name out of any possible ones will be generated for a multiply linked i-node, unless the −l option is specified. When −l is specified, no selection criteria apply to the names generated. All possible names for every linked file on the file system will be included in the output.

On very large file systems, memory may run out before *ff* does.

NAME

filesave, tapesave — daily/weekly UNIX file system backup

SYNOPSIS

/etc/filesave.?
/etc/tapesave

DESCRIPTION

These shell scripts are provided as models. They are designed to provide a simple, interactive operator environment for file backup. *Filesave.?* is for daily disk-to-disk backup and *tapesave* is for weekly disk-to-tape.

The suffix **.?** can be used to name another system where two (or more) machines share disk drives (or tape drives) and one or the other of the systems is used to perform backup on both.

SEE ALSO

shutdown(1M), volcopy(1M).

1

NAME

finc — fast incremental backup

SYNOPSIS

finc [selection-criteria] file-system raw-tape

DESCRIPTION

Finc selectively copies the input *file-system* to the output *raw-tape* . The cautious will want to mount the input *file-system* read-only to insure an accurate backup, although acceptable results can be obtained in read-write mode. The tape must be previously labelled by *labelit* (see *volcopy*(1M)). The selection is controlled by the *selection-criteria*, accepting only those inodes/files for whom the conditions are true.

It is recommended that production of a *finc* tape be preceded by the *ff* command, and the output of *ff* be saved as an index of the tape's contents. Files on a *finc* tape may be recovered with the *frec* command.

The argument **n** in the *selection-criteria* which follow is used as a decimal integer (optionally signed), where $+n$ means more than n, $-n$ means less than n, and n means exactly n. A day is defined as a 24 hours.

−a *n*	True if the file has been accessed in *n* days.
−m *n*	True if the file has been modified in *n* days.
−c *n*	True if the i-node has been changed in *n* days.
−n *file*	True for any file which has been modified more recently than the argument *file*.

EXAMPLES

To write a tape consisting of all files from file-system /**usr** modified in the last 48 hours:

```
finc −m −2 /dev/rdiskusr /dev/rtp0
```

SEE ALSO

cpio(1), ff(1M), frec(1M), volcopy(1M).

1

NAME

format — format and/or check RP06 and RM05 disk packs

DESCRIPTION

Format will format new RP06 or RM05 packs and check used packs (with write inhibited). The program reports the location and type of errors encountered, including ECC correctable error burst sizes.

EXECUTION

The following example shows how to load *format* on a VAX-11/780 with a UNIX updated floppy disc:

>>>H<cr>
HALTED AT *nnnnnnnn*

>>>B<cr>
CPU HALTED
INIT SEQ DONE
HALT INST EXECUTED
HALTED AT *nnnnnnnn*
LOAD DONE, *nnnnnnnnn* BYTES LOADED

$$

To execute *format*, type **/stand/format** after the standalone shell prompt **$$**. The formatter will print out its command vocabulary, and proceed inter-actively. If one wishes to format a pack on disk drive 1, for example, the command is **d1f**. The program will double check format requests, as pack contents will be destroyed.

COMMANDS

m *n*	MBA with drive doing the format is *n*. (defaults to 0)
d *n*	drive with the pack to be formatted or checked is *n*. (drive number must be between 1 and 7)
f	format pack
c	check pack format
q	quit
v	print vocabulary
R *n*	set the error report level to *n*.
X	will tell you about the available report levels.

The X command will explain the Report Level options the first time it is executed. Subsequent execution by the operator or by the program during error logging, will merely print the information defined by the current report level.

FILES

/stand/format

SEE ALSO

780ops(8).

NAME

frec – recover files from a backup tape

SYNOPSIS

/etc/frec [**−p** path] [**−f** reqfile] raw-tape i-number:name ...

DESCRIPTION

Frec recovers files from the specified *raw-tape* backup tape written by *volcopy*(1M) or *finc*(1M), given their *i-numbers*. The data for each recovery request will be written into the file given by *name* .

The **−p** option allows you to specify a default prefixing *path* different from your current working directory. This will be prefixed to any *names* that are not fully qualified, i.e. that do not begin with / or ./. If any directories are missing in the paths of recovery *names* they will be created.

−p *path* Specifies a prefixing *path* to be used to fully qualify any names that do not start with / or ./.

−f *reqfile* Specifies a file which contains recovery requests. The format is i-number:newname, one per line.

EXAMPLES

To recover a file, i-number 1216 when backed-up, into a file named **junk** in your current working directory:

frec /dev/rmt0 1216:junk

To recover files with i-numbers 14156, 1232, and 3141 into files **/usr/src/cmd/a**, **/usr/src/cmd/b** and **/usr/joe/a.c**:

frec −p /usr/src/cmd /dev/rmt0 14156:a 1232:b 3141:/usr/joe/a.c

SEE ALSO

cpio(1), ff(1M), finc(1M), volcopy(1M).

BUGS

While paving a path (i.e. creating the intermediate directories contained in a pathname) *frec* can only recover inode fields for those directories contained on the tape and requested for recovery.

1

NAME

fsck, dfsck — file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [**−y**] [**−n**] [**−s**X] [**−S**X] [**−t** file] [**−q**] [**−D**] [**−f**] [file-systems]

/etc/dfsck [options1] filsys1 ... − [options2] filsys2 ...

DESCRIPTION

Fsck

Fsck audits and interactively repairs inconsistent conditions for UNIX file systems. If the file system is consistent then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *fsck* will default to a **−n** action.

Fsck has more consistency checks than its predecessors *check*, *dcheck*, *fcheck*, and *icheck* combined.

The following options are interpreted by *fsck*.

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

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

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

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

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

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

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

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

1

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

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

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

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

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Incorrect number of blocks.
   Directory size not 16-byte aligned.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated inode.
   Inode number out of range.
8. Super Block checks:
   More than 65536 inodes.
   More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. If it is empty, *fsck* will silently remove them. *Fsck* will force the reconnection of nonempty directories. The name assigned is the inode number. The only restriction is that the directory **lost+found** must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster and should be used with everything but the *root* file system.

Dfsck

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

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

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

FILES

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

SEE ALSO

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

BUGS

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

DIAGNOSTICS

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

1

NAME

fscv — convert files between PDP-11 and VAX-11/780 systems

SYNOPSIS

/etc/fscv **−v** ispecial [ospecial]
/etc/fscv **−p** ispecial [ospecial]

DESCRIPTION

Fscv converts file systems between PDP-11 and VAX-11/780 formats. The super block, free list, and inodes are converted to the format of the output file. *Fscv* may be executed on PDP-11 and VAX processors. The mandatory flag specifies the format of the converted file system:

−v Convert file system from PDP-11 to VAX format.

−p Convert file system from VAX to PDP-11 format.

Ispecial is the name of a special file containing a file system to be converted (e.g.; **/dev/rrp1**). The optional *ospecial* is the name of the special file to receive the results of the conversion. If *ospecial* is specified the entire contents of *ispecial* are copied to *ospecial* before the conversion is performed. If *ospecial* is not specified an in-place conversion of *ispecial* is performed. The following items should be noted before executing *fscv*:

1. A file system consistency check (*fsck*(1M)) should be performed on *ispecial* immediately prior to executing *fscv*.
2. Neither *ispecial* nor the optional *ospecial* should contain a mounted file system during execution of *fscv*. Modification to either the input or the output file system while *fscv* is executing will probably corrupt the converted file system.
3. A backup of *ispecial* (see *volcopy*(1M)) is highly recommended if an in-place conversion is to be performed. System crashes, I/O errors, etc., during execution of *fscv* may destroy the file system contained in *ispecial*. Also, if the optional *ospecial* is specified any data contained in that special file will be over written.
4. If the optional *ospecial* is specified, this special file must be large enough to contain the entire contents of *ispecial*. See the appropriate special files in section 4.

EXAMPLES

Copy and convert a file system from PDP-11 to VAX format:

    /etc/fscv −v /dev/rrp0 /dev/rrp10

Perform an in-place conversion from VAX to PDP-11 format:

    /etc/fscv −p /dev/rrp10

BUGS

The boot block is not modified during conversion. The resulting file system will not be bootable. No data contained in the files of the file system are modified.

SEE ALSO

fsck(1M), volcopy(1M).

1

NAME

fsdb − file system debugger

SYNOPSIS

/etc/fsdb special [−]

DESCRIPTION

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

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

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

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

The symbols recognized by *fsdb* are:

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

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

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

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

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

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

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

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current working i-node.
ln=4	changes the link count for the working i-node to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working i-node.
2i.fd	prints the first 32 directory entries for the root i-node of this file system.
d5i.fc	changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.
512B.p0o	prints the superblock of this file system in octal.
2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.

1

d7.nm="name"	changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.
a2b.p0d	prints the third block of the current inode as directory entries.

SEE ALSO

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

1

NAME

fts — Field Test Set interface

SYNOPSIS

/etc/fts find util-id
/etc/fts stat file
/etc/fts set hex-num command args

DESCRIPTION

Fts provides an interface to the 3B20S Field Test Set (FTS). The FTS is a hardware device for tracing the execution of a process based on its utility ID.

For UNIX, the utility ID of a process is a 24 bit quantity divided into two fields. By default: the low order 16 bits contain the i-number of the process file, and the high order 8 bits contain the minor device number of the filesystem on which the process file exists.

After a *fork*(2) system call, the child process's utility ID is the same as the parent's. After an *exec*(2) system call, if the process's utility ID had previously been modified (see below), it remains unchanged, otherwise it is set to the default value.

The following options are recognized by *fts*:

find *util-id* Prints on the standard output, the device name and path name of a file that has utility ID of *util-id*. *Util-id* is interpreted as a hexadecimal constant.

stat *file* Prints on the standard output, the utility ID of *file*.

set*hex-num command args*
Changes its own utility ID, and then overlays itself with *command*. The new utility ID is as follows: the high order 8 bits have the value −1 (all bits set), and the low order 16 bits are set to *hex-num*. *Hex-num* is interpreted as a hexadecimal constant.

SEE ALSO

exec(2), fork(2), sys3b(2).

NAME

fuser — identify processes using a file or file structure

SYNOPSIS

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

DESCRIPTION

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

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

1

EXAMPLES

fuser −ku /dev/dsk1?

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

fuser −u /etc/passwd

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

fuser −ku /dev/dsk1? −u /etc/passwd

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

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

FILES

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

SEE ALSO

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

NAME

fwtmp, wtmpfix — manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [**−ic**]
/usr/lib/acct/wtmpfix [files]

DESCRIPTION

Fwtmp

Fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in **wtmp** to formated 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.

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 **/etc/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.

In addition to correcting time/date stamps, *wtmpfix* will check the validity of the name field to ensure that it consists soley of alphanumeric characters, a **$** 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), acct(2), acct(4), utmp(4).

1

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

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

The default terminal is **nonep**; i.e., any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition. The optional fourth argument, *linedisc*, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, **LDISC0**.

When given no optional arguments, *getty* sets the *speed* of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, newline characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(2)).

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

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

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

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

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

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

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

BUGS

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

1

NAME

init, telinit — process control initialization

SYNOPSIS

/etc/init [0123456SsQq]

/etc/telinit [0123456sSQqabc]

DESCRIPTION

Init

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

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

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

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

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

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

If a **0** through **6** is entered *init* enters the corresponding *run-level*. Any other input will be rejected and the user will be 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.*

1

NAME

install − install commands

SYNOPSIS

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

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.

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*, 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.

1

SEE ALSO

make(1), mk(8).

NAME

ipb — read the EAI Input Parameter Buffer

SYNOPSIS

/etc/ipb

DESCRIPTION

Ipb prints the settings of the various fields in the EAI Input Parameter Buffer. Information displayed includes the method used to boot the system, whether the backup root file system is being used, whether certain hardware error checks are enabled and whether minimal configuration has been specified.

FILES

/usr/include/sys/ipb.h

SEE ALSO

3B20ops(8).

1

NAME

killall — kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

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

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

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

FILES

/etc/shutdown

SEE ALSO

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

NAME

link, unlink — exercise link and unlink system calls

SYNOPSIS

/etc/link file1 file2
/etc/unlink file

DESCRIPTION

Link and *unlink* perform their respective system calls on their arguments, abandoning all error checking. These commands may only be executed by the super-user, who (it is hoped) knows what he or she is doing.

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]

DESCRIPTION

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Restrictions.

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

Models.

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

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

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

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

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

- **−c** compressed print
- **−e** expanded print

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

1

EXAMPLES

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

   /usr/lib/lpadmin −php2 −mhp

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

   lp −dhp2 −o−c files

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

   /usr/lib/lpadmin −pst1 −v/dev/tty20 −m1640

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

   nroff −T450 files | lp −dst1 −of
   nroff −T450−12 files | lp −dst1 −of
   nroff −T37 files | col | lp −dst1

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

   lp −dst1 −o12 /etc/passwd

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

FILES

/usr/spool/lp/*

SEE ALSO

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

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

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

1

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

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

FILES

/usr/spool/lp/*

SEE ALSO

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

NAME

mkboot — convert a.out file to boot image

SYNOPSIS

/etc/mkboot a.out-file boot-file

DESCRIPTION

Mkboot creates *boot-file* as a main-memory image of the *a.out-file*. *Mkboot* creates the boot-file with the text first, null byte padding from the end of the text to the start of the data, the data, null byte data for the bss, and null byte padding to bring the boot-file size up to a multiple of 512.

DIAGNOSTICS

Mkboot prints the starting and ending addresses for text, data, and bss on the standard error output.

Self-explanatory complaints about bad arguments and bad *a.out* format.

1

NAME

mkfs − construct a file system

SYNOPSIS

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

DESCRIPTION

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

1

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 (see *3B20boot*(8) or *unixboot*(8)). The second token is a number specifying the size of the created file system in *physical* disk blocks. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of inodes in the file system. The maximum number of inodes configurable is 65500. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **−bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or − to specify set-user-id mode or not. The third is **g** or − for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions (see *chmod*(1)).

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

If the file is a regular file, the next token is a path name whence the contents and size are copied. If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
/stand/diskboot
4872 110
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

In both command syntaxes, the rotational *gap* and the number of *blocks/cyl* can be specified. The following values are recommended:

Device	Gap Size	Blks/Cyl
RL01/02	7	40
RP03	5	200
RP04/05/06	7	418
RP07	7	400
RM03	7	160
RM05	7	608
RM80	9	434
3B20S MHD	7	608
default	7	400

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

SEE ALSO

dir(4), fs(4), unixboot(8), 3B20boot(8).

BUGS

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

1

NAME

mknod − build special file

SYNOPSIS

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

DESCRIPTION

Mknod makes a directory entry and corresponding i-node for a special file. The first argument is the *name* of the entry. In the first case, the second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g. unit, drive, or line number), which may be either decimal or octal.

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

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

SEE ALSO

mknod(2).

1

NAME

mount, umount — mount and dismount file system

SYNOPSIS

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

/etc/**umount** special

DESCRIPTION

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

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

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

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

1

FILES

/etc/mnttab mount table

SEE ALSO

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

DIAGNOSTICS

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

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

BUGS

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

NAME

msi — memory system diagnostic interface

SYNOPSIS

/etc/msi
/etc/msi rmv c a p
/etc/msi rst c a p
/etc/msi clr c a p
/etc/msi enb
/etc/msi dis
/etc/msi find c a p
/etc/msi kill c a p

DESCRIPTION

Msi provides the facility for controlling the memory system of the processor. The granularity for memory management is a 2K page. A physical memory board (or array) can contain from 64 to 512 pages, depending on board type. There can be 16 arrays on a memory controller and 2 controllers on a system. Hence, to completely specify a memory page requires the controller number, array number and page number indicated by *c*, *a*, and *p* respectively, in the argument list. These numbers are supplied by the operating system in the event of a memory system error.

1

Msi will become interactive if invoked without arguments. Valid arguments are:

Rmv will queue the addressed page for removal. The page cannot be removed immediately if it is currently in use, but must be delayed until the process claiming it moves or terminates. Certain memory system errors will automatically queue a page for removal.

Rst will return a previously removed page back to the system for re-use.

Clr will clear the addressed page, typically removing any parity errors in the page at the expense of lost data.

Enb will enable the hardware refresh and correctable parity error detection for the entire memory system.

Dis will disable the hardware error detection. This must be done before a new array is installed to prevent a flood of refresh parity errors before the new pages are cleared.

Find will search for the first process claiming the addressed page.

Kill will terminate all processes using the addressed page.

SEE ALSO

3B20ops(8).

NAME

mvdir — move a directory

SYNOPSIS

/etc/**mvdir** dirname name

DESCRIPTION

Mvdir 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).

Only super-user can use *mvdir*.

SEE ALSO

mkdir(1).

NAME

ncheck − generate names from i-numbers

SYNOPSIS

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

DESCRIPTION

Ncheck with no argument generates a path name vs. i-number list of all files on a set of default file systems. Names of directory files are followed by /.. The **−i** option reduces the report to only those files whose i-numbers follow. The **−a** option allows printing of the names . and .., which are ordinarily suppressed. The **−s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

A file system may be specified.

The report is in no useful order, and probably should be sorted.

SEE ALSO

fsck(1M), sort(1).

DIAGNOSTICS

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

1

NAME

newboot — load VTOC, prom patch, or lboot

SYNOPSIS

/etc/**newboot** boot-special [**−v** vtoc] [**−u** prompat] [**−l** lboot]

DESCRIPTION

Newboot replaces the entries specified by its options on the given *boot-special* section of a disk. *Newboot* verifies that each given file will fit in the specified entry, and calls *dd*(1) to move it in.

Lboot is a file containing the boot program that is loaded by the 3B20S firmware and executed to boot UNIX.

Prompat is a file containing patches to the system microcode that are read in by the 3B20S firmware when the machine is booted.

Vtoc is a file containing a volume table of contents used by the 3B20S firmware to find the prom patch and lboot locations on disk and is used by *lboot* to find the **root** and backup **root** file systems.

SEE ALSO

dd(1), mkboot(1M), 3B20boot(8).

DIAGNOSTICS

FILE too large (BLKS blocks max.)	FILE is too big for the specified entry.
unknown option *X*	Option *X* not recognized.
can't open file FILE	FILE not found.

WARNINGS

Installing a bad *vtoc*, *prompat*, or *lboot* may make the affected disk pack unbootable. Be sure you have a good backup disk before *newboot* is run.

1

NAME

nscloop − perform the NSC local network loopback functions

SYNOPSIS

/usr/nsc/nscloop [netname...] [**−l**] [**−s**] [**−c**] [**−u** units] [**−d** names] [**−f** file] [**−m** login]

DESCRIPTION

Nscloop uses the message loopback feature of the NSC adapter hardware to gather network statistics and to operationally query the availability of remote adapters. *Nscloop* generates and prints a brief report on the standard output. The first argument to *nscloop* may be one or more network names (see *nscmon(1M)* for a complete description of a network). If no network is given, all the networks specified in the network file, **/usr/nsc/nets** will be accessed. If more than one network is specified, the **−f, −d, −u** options are disabled. For each network specified, the **−l, −s,** and/or **−c** functions are performed on all known adapters, as determined by the network topological file, **/etc/nsc.** *Nscloop* recognizes the following options:

−l Loop a message and associated data block off each specified adapter. Compare each byte sent with each byte returned and report comparison errors. This is the default mode.

−s Gather and report the trunk statistics for the specified adapters.

−c Gather, report, and clear the trunk statistics for the specified adapters. This function is reserved to the super-user.

−f file Use *file* as the network topological file for the specified network. This file contains the symbolic names of each machine on the network. A report is generated for each unique adapter that configured hosts are connected to. *File* format is assumed to be as follows:

machine_name:anything:anything:device

−d names Perform the indicated function to only those adapters on the specified network where the host *name* is connected.

−u units Perform the indicated functions to only those adapters on the specified network whose unit number is *unit.*

−m login Send mail to *login* if any error is detected.

By default, option **−l** is enabled. If more that one of **−l**, **−s**, or **−c** are specified, each function will be performed on the indicated adapters. If no adapters are explicitly selected for reporting, *nscloop* will query all adapters for the specified network found in the network topological file, **/etc/nsc**.

FILES

/etc/nsc the NSC network topological file

SEE ALSO

nscmon(1M).

DIAGNOSTICS

All error messages are designed to be self-explanatory.

1

NAME

nscmon − operationally control the NSC local network

SYNOPSIS

/etc/nscmon options

DESCRIPTION

Nscmon provides the operational interface to control the NSC local network. *Nscmon* starts and stops all network transfers; *nscmon* enables and inhibits transfers to individual nodes. All operations are in the eyes of the local node only.

The *nusend*(1C) software allows the support of more than one adapter on a host, where each adapter defines a separate network. The network file **/usr/nsc/nets** contains all the networks known to the local node. Most operations require the specification of one or more networks. If more than one netname is given, the operation is performed on each network in turn. Every *option* that requires a netname may optionally take the special case **all**. In that case, the *option* will perform its operation for all the networks known, as specified in the network file **/usr/nsc/nets**. The following *options* are recognized:

−**start** netname
: Start up the *nusend*(1C) software on the local node for the specified network. This command initializes the NSC listener process for each network, marks all the currently configured nodes online, and enables the routing of file transfers across the NSC network. This command will not clear a hung adapter or NSC driver. The converse of this option is −**stop**.

−**stop** netname
: Terminate the *nusend*(1C) software on the local node for the specified network. Any files currently queued as well as all subsequent jobs will be routed across the RJE link (if it exists). This command inhibits any incoming or outgoing file transfers. This command will not clear a hung adapter or hung NSC driver.

−**cancel** netname
: Cancel the current active adapter operation (within the driver) for the driver associated with the specified network. The operation is marked as though it had failed. An error will be returned to the user process and suspended processes will continue normally. This command is especially useful for clearing hung processes within the driver.

−**halt** netname
: Disable (via software) all operations to the adapter for the specified network. The driver will process opens normally, but all functions to the adapter will be inhibited. This command does not clear a hung driver or hung processes, but inhibits all operations to the adapter. The converse of this command is −**restart**.

−**restart** netname
: Enable (via software) all adapter operations for the adapter associated with the specified network. This command restarts any suspended processes within the driver. This function is the converse of −**halt**.

1

−t netname Turn off the NSC adapter to adapter protocol process tracing for the adapter associated with the specified network. The binary trace files may be found in **/usr/nsc/log/nsclog.***, where the * is the process ID of the read/send process.

+t netname Turn on the NSC adapter protocol process tracing for the adapter associated with the specified network.

−e netname Turn off the NSC adapter to adapter protocol error logging for the adapter associated with the specified network. The binary error files may be found in **/usr/nsc/log/nscerr.***, where the * is the process ID of the read/send process.

+e netname Turn on the NSC adapter protocol error logging for the adapter associated with the specified network.

−ps Print certain information about active *nusend*(1C) processes. The format of the listing is:

PID TIME CMD

Nscsend and *nscread* processes are listed under their parents. The format for orphan processes is:

PID PPID TIME CMD

The cumulaive execution time (TIME) is not displayed on the UNIX/370 implementation.

−on netname names

Mark all nodes in the *name* list for the specified network online and notify the node to forward all queued files to the local machine. If name is the special case **all** all nodes for the specified network are marked up, as configured in the network topological file, **/etc/nsc**. Any files currently queued for the named node(s) and all subsequent submitted transfers to the named node(s) will be routed across the NSC network. This function is automatically performed if the **−start** option is used.

−off netname names

Mark all the nodes in the *name* list for the specified network offline. If name is the special case **all** all the nodes for the specified network are marked offline, as configure in the network topological file, **/etc/nsc**. Any files currently queued for the named node(s) and all subsequent jobs submitted to the specified node(s) will be routed across the RJE link (if it exists).

−p netname names

Same as **−on** option.

−clear Clear the process table of (kill off) all *nusend*(1C) processes that did not die normally.

−loop Perform the NSC local loopback function. Same as **nscloop −l**.

−stat Query the operational status of the NSC network. Same as **nscstat −l**.

All options may be freely interdispersed; the operations will be performed in the order given on the command line.

1

FILES

/etc/nsc	the network topological file
/usr/nsc/nets	the networks known to the local node and the associated devices
/usr/nsc/log/nsclog.*	binary trace log
/usr/nsc/log/nscerr.*	binary NSC error log
/usr/nsc/online/*	the NSC network is enabled for this network
/usr/nsc/cons/*	remote nodes currently considered online locally
/usr/nsc/rvchan	nodes currently configured on the network
/usr/nsc/nsctorje	program that routes jobs on inactive nodes across the RJE line
/usr/nsc/nsccmd	program that sends a message to a remote machine telling it to send any queued jobs to the local machine
/usr/nsc/nsclisten	the NSC network listen daemon
/usr/nsc/nscd	the NSC network send daemon
/usr/nsc/nscrecv	the NSC network receive daemon

SEE ALSO

nscloop(1M), nscstat(1C), nsctorje(1C).

DIAGNOSTICS

All error messages are designed to be self explanatory.

1

NAME

pcldaemon — PCL link monitor

SYNOPSIS

/usr/lib/pcldaemon

DESCRIPTION

Pcldaemon monitors the *pcl*(7) control channel, servicing requests as they arrive. Requests are transmitted via *net*(1C).

FILES

/dev/pcl/*?*[0-7] PCL channel interfaces for system *?*.
/dev/pcl/ctrl PCL control channel.
/usr/adm/pcllog activity log.

SEE ALSO

net(1C), pcl(7).

DIAGNOSTICS

cannot open pcl control channel
Another *pcldaemon* is running.

WARNINGS

Running *pcldaemon* may present security hazards. A super-user may *net*(1C) to any system on the PCL bus that is running *pcldaemon* and execute any command on that system.

NAME

prm — send a Processor Recovery Message

SYNOPSIS

/etc/prm message

DESCRIPTION

Prm sends a Processor Recovery Message (PRM) to the Emergency Action Interface (EAI).

Message is converted to a 16 nibble sequence and must therefore contain only the digits **1** through **9** and the characters **a** through **f**. After the command, *message* will appear in the **PRM** field of the EAI display.

SEE ALSO

3B20ops(8).

BUGS

Because of the design of the EAI it is possible to miss a **PRM** if insufficient time has passed since the last message.

NAME

prfld, prfstat, prfdc, prfsnap, prfpr — operating system profiler

SYNOPSIS

/etc/prfld [namelist]
/etc/prfstat [**on** | **off**]
/etc/prfdc file [period [off_hour]]
/etc/prfsnap file
/etc/prfpr file [cutoff [namelist]]

DESCRIPTION

Prfld, *prfstat*, *prfdc*, *prfsnap*, and *prfpr* form a system of programs to facilitate an activity study of the UNIX operating system.

Prfld is used to initialize the recording mechanism in the system. It generates a table containing the starting address of each system subroutine as extracted from *namelist*.

Prfstat is used to enable or disable the sampling mechanism. Profiler overhead is less than 1% as calculated for 500 text addresses. *Prfstat* will also reveal the number of text addresses being measured.

1

Prfdc and *prfsnap* perform the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed. *Prfdc* will store the counters into *file* every *period* minutes and will turn off at *off_hour* (valid values for *off_hour* are **0**−**24**). *Prfsnap* collects data at the time of invocation only, appending the counter values to *file*.

Prfpr formats the data collected by *prfdc* or *prfsnap*. Each text address is converted to the nearest text symbol (as found in *namelist*) and is printed if the percent activity for that range is greater than *cutoff*.

FILES

/dev/prf	interface to profile data and text addresses
/unix	default for namelist file

SEE ALSO

prf(7).

NAME

pwck, grpck — password/group file checkers

SYNOPSIS

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

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 is derived from Setting up UNIX in the *UNIX System Administrator's Guide*. 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(4), passwd(4).
Setting up UNIX in the *UNIX System Administrator's Guide*.

DIAGNOSTICS

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

1

NAME

reboot — reboot the system

SYNOPSIS

/etc/reboot

DESCRIPTION

Reboot generates a Maintenance Reset Function (MRF), causing the processor to enter its system bootstrap code thereby rebooting the system. It can be used to reboot the processor remotely, but this is practical only if a terminal line is enabled in **/etc/inittab** for state 1 so that the file systems can be checked and state 2 entered.

Since the boot sequence will prompt for a pathname at the system console if **PROMPT UNIX** is set in the **EAI**, make sure that this is not the case by using *ipb*(1M) beforehand.

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

SEE ALSO

ipb(1M), 3B20ops(8).

1

NAME

rmv — remove unit from service before on-line diagnostics

SYNOPSIS

/dgn/bin/rmv name unit

DESCRIPTION

Rmv removes the device specified by *name* and *unit* from service. For example, the following command line removes DFC 1 (Disk File Controller 1) from service:

rmv dfc 1

SEE ALSO

dgn(1M), rst(1M).
3B DMERT Output Messages, OM-4C000-01.

WARNING

Diagnostic commands are intended for use only by trained hardware maintenance personnel.

1

NAME

rst — restore unit to service after on-line diagnostics

SYNOPSIS

/dgn/bin/rst name unit [options]

DESCRIPTION

Rst restores the device identified by *name* and *unit* into service according to the *options* specified. A device can be restored to service conditionally or unconditionally. By default, *rst* restores a device into service conditionally, unless the the option **ucl** is specified. A conditional restore implies that diagnostics will first be performed, and the final results must be ATP (All Tests Passed) before the device will be restored to service. Otherwise the device is left out of service. If the device is restored unconditionally (i.e., **ucl** option is specified), then no diagnostics are performed.

The following *options* are recognized, each as a separate argument:

raw Print the diagnostic results of every phase and all failures. By default, only the final results and first five failures of each failing phase will be printed.

ucl Unconditionally restore the device specified by *name* and *unit* into service. Note this *option* implies that no diagnostics will be performed.

tlp Executes the Trouble Location Procedure at the conclusion of the diagnostic. This procedure analyzes all diagnostic failures and generates a weighted list of faulty circuit packs. This *option* must *not* be used in conjunction with **ucl**.

file=*filename*
Routes all output messages into a file named *filename* instead of the user's terminal. *Filename* is opened for appending and is relative to the directory **/dgn/dgnc** unless a full pathname is specified.

cont This *option* is only effective when *name* and *unit* is an Input-Output Processor (IOP). By default, *rst* will restore an IOP and its associated peripheral controllers (PCs) into service. Use of this *option* restores only the IOP and *not* its PCs. Note that restoring a Disk File Controller (DFC) never implies restoring its Moving Head Disks (MHDs).

hu=*name unit*
This *option* allows a helper unit identified by *name* and *unit* to be specified. For example, when diagnosing the magnetic tape controller (i.e., UN32), a diagnostic test tape with a write ring must be mounted on the specified helper unit. The following example shows how one might invoke diagnostics using the helper unit option:
dgn un32 0 ph=5 hu=mt 2

EXAMPLES

The following two examples show how one might invoke this command for either an unconditional or conditional restore, respectively.

Example 1:
rst dfc 1 ucl
would restore the device **dfc 1** into service unconditionally. The option **ucl** is the only valid *option* for an unconditional restore request.

Example 2:
rst dfc 1 raw tlp file=filename
would restore the device **dfc 1** to service if all diagnostics

results were ATP. The remaining *options* are applied as described above.

SEE ALSO

dgn(1M), rmv(1M).
3B DMERT Output Messages, OM-4C000-01.

WARNING

Diagnostic commands are intended for use only by trained hardware maintenance personnel.

1

NAME

runacct — run daily accounting

SYNOPSIS

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

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/src/cmd/acct/tacct.h
/usr/src/cmd/acct/ctmp.h
/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), acct(2), acct(4), utmp(4).
UNIX Accounting System in the *UNIX System Administrator's Guide*.

DIAGNOSTICS

The accounting system will start complaining with *****RECOMPILE pnpsplit WITH NEW HOLIDAYS***** after the last holiday of the year. See *The UNIX Accounting System* for more on how to correct this condition. Other diagnostics are placed in various error and log files.

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.

1

NAME

sa1, sa2, sadc — system activity report package

SYNOPSIS

/usr/lib/sa/sadc [t n] [ofile]

/usr/lib/sa/sa1 [t n]

/usr/lib/sa/sa2 [**−ubdycwaqvm**] [**−s** time] [**−e** time] [**−i** sec]

DESCRIPTION

System activity data can be accessed at the special request of a user (see *sar*(1)) and automatically on a routine basis as described here. The operating system contains a number of counters that are incremented as various system actions occur. These include CPU utilization counters, buffer usage counters, disk and tape I/O activity counters, TTY device activity counters, switching and system-call counters, file-access counters, queue activity counters, and counters for inter-process communications.

Sadc and shell procedures *sa1* and *sa2* are used to sample, save and process this data.

1

Sadc, the data collector, samples system data *n* times every *t* seconds and writes in binary format to *ofile* or to standard output. If *t* and *n* are omitted, a special record is written. This facility is used at system boot time to mark the time at which the counters restart from zero. The */etc/rc* entry:

```
su sys −c "/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`&"
```

writes the special record to the daily data file to mark the system restart.

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file **/usr/adm/sa/sa***dd* where *dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted. The entries in **crontab** (see *cron*(1M)):

```
0 * * * 0,6 su sys −c "/usr/lib/sa/sa1"
0 8−17 * * 1−5 su sys −c "/usr/lib/sa/sa1 1200 3"
0 18−7 * * 1−5 su sys −c "/usr/lib/sa/sa1"
```

will produce records every 20 minutes during working hours and hourly otherwise.

The shell script *sa2*, a variant of *sar*(1), writes a daily report in file **/usr/adm/sa/sar***dd*. The options are explained in *sar*(1). The **crontab** entry:

```
5 18 * * 1−5 su adm −c "/usr/lib/sa/sa2 −s 8:00 −e 18:01 −i 3600 −A"
```

will report important activities hourly during the working day.

The structure of the binary daily data file is:

```
struct sa {
      struct sysinfo si;  /* see /usr/include/sys/sysinfo.h */
      int  szinode;       /* current entries of inode table */
      int  szfile;        /* current entries of file table */
      int  sztext;        /* current entries of text table */
      int  szproc;        /* current entries of proc table */
      int  mszinode;      /* size of inode table */
      int  mszfile;       /* size of file table */
      int  msztext;       /* size of text table */
      int  mszproc;       /* size of proc table */
      long  inodeovf;     /* cumul. overflows of inode table */
      long  inodeovf;     /* cumul. overflows of file table */
      long  textovf;      /* cumul. overflows of text table */
      long  procovf;      /* cumul. overflows of proc table */
      time_t  ts;         /* time stamp, seconds */
      long  devio[NDEVS][4];    /* device info for up to NDEVS units */
#define IO_OPS        0         /* cumul. I/O requests */
#define IO_BCNT       1         /* cumul. blocks transferred */
#define IO_ACT        2         /* cumul. drive busy time in ticks */
#define IO_RESP       3         /* cumul. I/O resp time in ticks */
};
```

FILES

/usr/adm/sa/sa*dd*	daily data file
/usr/adm/sa/sar*dd*	daily report file
/tmp/sa.adrfl	address file

SEE ALSO

sag(1G), sar(1), timex(1).

1

NAME

setmnt — establish mount table

SYNOPSIS

/etc/setmnt

DESCRIPTION

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

filesys node

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

FILES

/etc/mnttab

SEE ALSO

mnttab(4).

BUGS

Evil things will happen if *filesys* or *node* are longer than 10 characters.
Setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.

1

NAME

setmrf — override system MRF action

SYNOPSIS

/etc/setmrf [d | D | h | H | r]

DESCRIPTION

Setmrf overrides the default action taken by the system in the event of a Maintenance Reset Function (MRF). A MRF can be caused by a hardware fault in the processor or as a result of a *panic* call in the operating system. The default action is to reboot the processor.

The **h** flag will cause the processor to enter an idle loop, resetting the internal sanity timer. The **H** flag will execute a *halt* instruction, causing all processor activity to stop. However, if the sanity timer is not inhibited at the EAI, the processor will reboot when the timer expires.

The **d** flag sets the MRF action to its default value of dump to disk followed by reboot. The **D** flag causes the processor to idle after the dump is taken.

The **r** flag causes the processor to reboot immediately, without taking a dump.

All other values are not implemented and are treated as in the default case.

SEE ALSO

3B20ops(8).

NAME

shutdown − terminate all processing

SYNOPSIS

/etc/shutdown

DESCRIPTION

Shutdown is part of the UNIX 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*(1)). 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 ALSO

mount(1M), sync(1).

NAME

ssr, setssr, clrssr — print or modify the System Status Register

SYNOPSIS

/etc/ssr

/etc/setssr [bit ...]

/etc/clrssr [bit ...]

DESCRIPTION

The System Status Register (SSR) serves as both a display for certain processor information as well as a mechanism for controlling various processor actions. Although implemented with negative logic in the hardware, these commands function in the normal logic sense. The *bit* argument is a decimal integer specifying the bit position in the SSR.

Ssr prints the current value of the SSR.

Setssr asserts the specified *bit* positions in the SSR.

Clrssr clears the specified *bit* positions in the SSR.

Extreme caution must be exercised when using these commands as the System Status Register can alter processor behavior. In particular, certain *bits* will isolate the I/O system from the processor, causing the system to crash.

Setssr and *clrssr* are most commonly used with bit **13** as an argument in order to enable or disable the cache bypass.

FILES

/usr/include/sys/ssr.h

SEE ALSO

3B20ops(8).

NAME

st — synchronous terminal control

SYNOPSIS

/etc/stload
/etc/stcntrl control action
/etc/stprint line device

DESCRIPTION

The *stload* command file is used to load the synchronous terminal prototype script, **/lib/stscr**, into the designated VPM hardware, and start execution of the script. As supplied, *stload* uses VPM hardware unit 0 and **/dev/st0**; it will need local modification to use a different hardware unit or to start more than one synchronous communications line.

The *stcntrl* command is used to activate and deactivate synchronous communications lines. The line that will be acted on is specified by *control*, (e.g. **/dev/st0**). The *action* argument may be either **on**, to activate the line, or **off**, to deactivate the line. The activation of a started line or the deactivation of a stopped line will result in an error. Note that *stload* activates the lines associated with the scripts that it loads.

The **/etc/rc** file should contain the following multi-user entry:

    /etc/stload

while each active synchronous line should be deactivated in **/etc/shutdown** by a line similar to the following:

    /etc/stcntrl /dev/st0 off

The *stprint* command associates a **/dev/sp*** file with a printer on synchronous communication line *line* with the ASCII device address character *device*. The *stprint* command prints the associated file name on its standard output.

FILES

/lib/stscr	synchronous terminal prototype script
/dev/un53.?	TN82/UN53 peripheral controller pair (3B20S only)
/dev/kmc?	KMC11-B microprocessor (DEC only)
/dev/st?	synchronous communications line control channels
/dev/tty*	synchronous terminal user channels
/dev/sp*	synchronous printer user channels

SEE ALSO

kmc(7), st(7), trace(7), un53(7), vpm(7).

NAME

sta — find status of pending on-line diagnostic requests

SYNOPSIS

/dgn/bin/sta

DESCRIPTION

The diagnostic command *sta* reports the status of all currently pending diagnostic requests within the Maintenance Input Request Administrator (MIRA). The contents of both the waiting and active diagnostic requests are printed along with their respective slot numbers.

SEE ALSO

dgn(1M), rmv(1M), rst(1M).
3B DMERT Output Messages, OM-4C000-01.

WARNING

Diagnostic commands are intended for use only by trained hardware maintenance personnel.

1

NAME

stgetty — wait on synchronous login line for use

SYNOPSIS

/etc/stgetty name type delay

DESCRIPTION

Stgetty is normally invoked by *init*(1M) as the first step in allowing users to login to the system. Lines in **/etc/inittab** tell *init* to invoke *stgetty* with the proper arguments.

Name should be the name of a terminal in **/dev** (e.g., **tty93**); *type* should be a single character chosen from **—**, which is used to start up a line, or **!**, which tells *stgetty* to update **/etc/utmp** and exit; *delay* is relevant for dial-up ports only. It specifies the time in seconds that should elapse before the port is disconnected if the user does not respond to the **login:** request.

Stlogin(1) is called with *delay* as an argument.

SEE ALSO

stlogin(1), init(1M), inittab(4), utmp(4), stermio(7).

NAME

sysdef — system definition

SYNOPSIS

/etc/sysdef [opsys [master]]

DESCRIPTION

Sysdef analyzes the named operating system file and extracts configuration information. This includes all hardware devices as well as system devices and all tunable parameters.

The output of *sysdef* can usually be used directly by *config*(1M) to regenerate the appropriate configuration files.

FILES

/unix	default operating system file
/etc/master	default table for hardware specifications

SEE ALSO

config(1M), master(4).

BUGS

For devices that have interrupt vectors but are not interrupt-driven, the output of *sysdef* cannot be used for *config*. Because information regarding *config* aliases is not preserved by the system, device names returned might not be accurate.

1

NAME

uuclean — uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [options]

DESCRIPTION

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

The following options are available.

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

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

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

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

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

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

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

FILES

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

SEE ALSO

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

NAME

uusub − monitor uucp network

SYNOPSIS

/usr/lib/uucp/uusub [options]

DESCRIPTION

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

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

The meanings of the connections report are:

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

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

The meanings of the traffic statistics are:

sfile sbyte rfile rbyte

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

The command:

uusub −c all −u 24

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

FILES

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

SEE ALSO

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

1

NAME

vcf — VAX-11/780 configuration verification program

DESCRIPTION

This program scans hardware registers and software configuration tables in order to verify device availability and addressing.

With the system halted, any of the console commands may be executed as described in *780ops*(8) under *Console Operation*. The following is an example of execution of *vcf* as seen on the console, starting with a halted system:

>>>H<cr>
    HALTED AT *nnnnnnnn*

>>>B<cr>
    CPU HALTED
    INIT SEQ DONE
    HALT INST EXECUTED
    HALTED AT *nnnnnnnn*
    LOAD DONE, *nnnnnnnn* BYTES LOADED

$$

The **$$** prompt indicates that the stand-alone shell (*sash*) is ready to accept commands. To execute the configuration verification program, type:

$$ stand/vcf [*unix_a.out*]

Default for the *unix_a.out* file is **/unix**.

Vcf will scan the VAX machine registers looking for Memory, MASSBUS Adapters (MBAs), and UNIBUS Adapters (UBAs). For memory and MASSBUS devices, hardware status information is reported. Information on UNIBUS devices is obtained from configuration information in the UNIX executable, and an attempt is made to verify device address and interrupt vectors.

FILES

/unix (or other UNIX executable)
/stand/vcf

SEE ALSO

780ops(8).

NAME

vlx — VAX-11/780 LSI console floppy interface

SYNOPSIS

vlx key [files]

DESCRIPTION

Vlx is used to maintain the console floppy. The floppy is in DEC RT-11 format. Hence, a *file* name is restricted to a 1- to 6-character alphanumeric name optionally followed by a **.** character separator and a 1- to 3-character alphanumeric extension. Upper and lower cases are mapped together. Only the last component of a path name is used.

Key is one character from the set **drtx**, optionally concatenated with one or both of **vf**. The meanings of the *key* characters are:

d Delete the named files from the floppy.

r Replace the named files on the floppy.

t Print a table of contents of the floppy. If no names are given, all files are tabled. If names are given, only those files are tabled.

x Extract the named files from the floppy. If no names are given, all files are extracted.

v Verbose. When used with **t**, it gives a long listing of all information about the files. When used with **x**, it precedes each file with a name.

f Use the next name as the floppy file name, instead of the default **/dev/conflp**.

FILES

/dev/conflp console floppy

SEE ALSO

780ops(8).

BUGS

Dependent on knowledge and correctness of DEC software.

1

NAME

volcopy, labelit — copy file systems with label checking

SYNOPSIS

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

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

DESCRIPTION

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

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

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

Other *options* are used only with tapes:

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

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

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

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

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

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

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

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

FILES

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

SEE ALSO

fs(4).

BUGS

Only device names beginning **/dev/rmt** (on DEC systems) or **/dev/rtp** (on 3B20S systems) are treated as tapes.

NAME

vpmc — compiler for the virtual protocol machine

SYNOPSIS

vpmc [**−mrcx**] [**−s** sfile] [**−l** lfile] [**−i** ifile] [**−o** ofile] file

DESCRIPTION

Vpmc is the compiler for a language that is used to describe communications link protocols. The output of *vpmc* is a load module for the virtual protocol machine (VPM), which is a software construct for implementing communications link protocols (e.g., BISYNC) on the DEC KMC11-B microprocessor. VPM is implemented by an interpreter in the KMC which cooperates with a driver in the UNIX host computer to transfer data over a communications link in accordance with a specified link protocol. UNIX user processes transfer data to or from a remote terminal or computer system through VPM using normal UNIX *open*, *read*, *write*, and *close* operations. The VPM program in the KMC provides error control and flow control using the conventions specified in the protocol.

The language accepted by *vpmc* is essentially a subset of C; the implementation of *vpmc* uses the RATFOR preprocessor (*ratfor*(1)) as a front end; this leads to a few minor differences, mostly syntactic.

There are two versions of the interpreter. The appropriate version for a particular application is selected by means of the **−i** option. The BISYNC version (**−i bisync**) supports half-duplex, character-oriented protocols such as the various forms of BISYNC. The HDLC version (**−i hdlc**) supports full-duplex, bit-oriented protocols such as HDLC. There is a separate HDLC interpreter for the KMS11 eight-line multiplexor; this version is selected by **−i hdlc/kms**. The communications primitives used with the BISYNC version are character-oriented and blocking; the primitives used with the HDLC versions are frame-oriented and non-blocking.

Options

The meanings of the command-line options are:

- **−m** Use *m4*(1) instead of *cpp* as the macro preprocessor.
- **−r** Produce RATFOR output on the standard output and suppress the remaining compiler phases.
- **−c** Compile only (suppress the assembly and linking phases).
- **−x** Retain the intermediate files used for communication between passes.
- **−s** *sfile* Save the generated VPM assembly language on file *sfile*.
- **−l** *lfile* Produce a VPM assembly-language listing on file *lfile*.
- **−i** *ifile* Use the interpreter version specified by *ifile* (default **bisync**).
- **−o** *ofile* Write the executable object file on file *ofile* (default **a.out**).

These options may be given in any order.

Programs

Input to *vpmc* consists of a (possibly null) sequence of array declarations, followed by one or more function definitions. The first defined function is invoked (on command from the UNIX VPM driver) to begin program execution.

Functions

A function definition has the following form:

function *name*()
statement_list
end

1

Function arguments (formal parameters) are not allowed. The effect of a function call with arguments can be obtained by invoking the function via a macro that first assigns the value of each argument to a global variable reserved for that purpose. See *EXAMPLES* below.

A *statement_list* is a (possibly null) sequence of labeled statements. A *labeled_statement* is a statement preceded by a (possibly null) sequence of labels. A *label* is either a name followed by a colon (:) or a decimal integer optionally followed by a colon.

The statements that make up a statement list must be separated by semicolons (;). (A semicolon at the end of a line can usually be omitted; refer to the description of RATFOR for details.) Null statements are allowed.

Statement Syntax

The following types of statements are allowed:

expression
lvalue = *expression*
lvalue += *expression*
lvalue −= *expression*
lvalue |= *expression*
lvalue &= *expression*
lvalue ^= *expression*
lvalue <<= *expression*
lvalue >>= *expression*
if(*expression*)*statement*
if(*expression*)*statement* else *statement*
while(*expression*)*statement*
for(*statement*; *expression*; *statement*)*statement*
repeat *statement*
repeat *statement* until *expression*
break
next
switch(*expression*){*case_list*}
return(*expression*)
return
goto *name*
goto *decimal_constant*
{*statement_list*}

repeat is equivalent to the **do** keyword in C; **next** is equivalent to **continue**.

A *case_list* is a sequence of statement lists, each of which is preceded by a label of the form:

case *constant*:

The label for the last *statement_list* in a *case_list* may be of the form:

default:

Unlike C, RATFOR supplies an automatic **break** preceding each new case label.

Expression Syntax

A *primary_expression* (abbreviated *primary*) is an lvalue or a constant. An *lvalue* is one of the following:

name
name[*constant*]

A *unary_expression* (abbreviated *unary*) is one of the following:

primary
name( )
system_call
++*lvalue*
−−*lvalue*
(*expression*)
!*unary*
~*unary*

The following types of expressions are allowed:

unary
unary+*primary*
unary−*primary*
unary|*primary*
unary&*primary*
unary&~*primary*
unary^*primary*
unary<<*primary*
unary>>*primary*
unary==*primary*
unary!=*primary*
unary>*primary*
unary<*primary*
unary>=*primary*
unary<=*primary*

Note that the right operand of a binary operator can only be a constant, a name, or a name with a constant subscript.

System Calls

A VPM program interacts with a communications device and a driver in the host computer by means of system calls (primitives).

The following primitives are available only in the BISYNC version of the interpreter:

atoe(*primary*)
Translate ASCII to EBCDIC. The returned value is the EBCDIC character that corresponds to the ASCII character represented by the value of the primary expression. The translation tables reflect the prejudices of a particular installation.

crc16(*primary*)
The value of the primary expression is combined with the cyclic redundancy check-sum at the location passed by a previous **crcloc** system call. The CRC-16 polynomial ($x^{16}+x^{15}+x^2+1$) is used for the check-sum calculation.

crcloc(*name*)
The two-byte array starting at the location specified by *name* is cleared. The address of the array is recorded as the location to be updated by subsequent **crc16** system calls.

etoa(*primary*)
Translate EBCDIC to ASCII. The returned value is the ASCII character that corresponds to the EBCDIC character represented by the value of the primary expression. The translation tables reflect the prejudices of a particular installation.

get(*lvalue*)
Get a byte from the current *transmit* buffer. The next available

1

byte, if any, is copied into the location specified by *lvalue*. The returned value is zero if a byte was obtained, otherwise it is non-zero.

getrbuf(*name*)
Get (open) a *receive* buffer. The returned value is zero if a buffer is available, otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by *name*. The array should be large enough to hold at least three bytes. The meaning of the buffer parameters is driver-dependent. If a receive buffer has previously been opened via a **getrbuf** call but has not yet been closed via a call to **rtnrbuf**, that buffer is reinitialized and remains the current buffer.

getxbuf(*name*)
Get (open) a *transmit* buffer. The returned value is zero if a buffer is available, otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by *name*. The array should be large enough to hold at least three bytes. The meaning of the buffer parameters is driver-dependent. If a transmit buffer has previously been opened via a **getxbuf** call but has not yet been closed via a call to **rtnxbuf**, that buffer is reinitialized and remains the current buffer.

1

put(*primary*)
Put a byte into the current *receive* buffer. The value of the primary expression is inserted into the next available position, if any, in the current receive buffer. The returned value is zero if a byte was transferred, otherwise it is non-zero.

rcv(*lvalue*)
Receive a character. The process delays until a character is available in the input silo. The character is then moved to the location specified by *lvalue* and the process is reactivated.

rsom(*constant*)
Skip to the beginning of a new *receive* frame. The receiver hardware is cleared and the value of *constant* is stored as the receive sync character. This call is used to synchronize the local receiver and remote transmitter when the process is ready to accept a new receive frame.

rtnrbuf(*name*)
Return a *receive* buffer. The original values of the buffer parameters for the current receive buffer are replaced with values from the array specified by *name*. The current receive buffer is then released to the driver.

rtnxbuf(*name*)
Return a *transmit* buffer. The original values of the buffer parameters for the current transmit buffer are replaced with values from the array specified by *name*. The current transmit buffer is then released to the driver.

xeom(*constant*)
Transmit end-of-message. The value of the constant is transmitted, then the transmitter is shut down.

xmt(*primary*)
Transmit a character. The value of the primary expression is transmitted over the communications line. If the output silo is full, the process waits until there is room in the silo.

xsom(*constant*)
Transmit start-of-message. The transmitter is cleared, then the value of *constant* is transmitted six times. This call is used to synchronize the local transmitter and the remote receiver at the beginning of a frame.

The following primitives are available only with the HDLC version of the interpreter:

abtxfrm( )
The current transmission, if any, is aborted, if possible, by sending a frame-abort sequence (seven one bits, followed immediately by a terminating flag). This operation is not feasible with some hardware interfaces, in which case this primitive is a no-operation.

getxfrm(*primary*)
Get a transmit buffer. If the transmit-buffer queue is *not* empty, the buffer at the head of the queue is removed from the queue and attached to the sequence number specified by the value of the primary expression If the sequence number is greater than seven or the sequence number already has a buffer attached, the process is terminated in error. The returned value is zero if a buffer was obtained, otherwise non-zero.

norbuf()
Test for the availability of an empty receive buffer. The returned value is **true** (non-zero) if the queue of empty receive buffers is currently empty; otherwise the returned value is **false** (zero).

rcvfrm(*name*)
Get a completed receive frame. If the queue of completed receive frames is non-empty, the frame at the head of the queue is removed and becomes the current receive frame. If a frame is obtained, the first five bytes of the frame are copied into the array specified by *name*. The returned value is **true** (non-zero) if a frame was obtained; otherwise, it is **false** (zero). The rightmost four bits of the returned value indicate the frame length as follows: if the value of the rightmost four bits is equal to fifteen, the frame length is greater than or equal to 15; otherwise the frame length is equal to the value of the rightmost four bits. The frame length includes the two CRC bytes at the end of the frame and any control information at the beginning of the frame. Bytes following the first two bytes of the frame, but not including the two CRC bytes, are copied into a receive buffer, if one is available at the time the frame is received. Bit 020 of the returned value is zero if a receive buffer was available, otherwise non-zero. The values of the leftmost three bits of the returned value are currently unspecified. If a frame was obtained, the first five bytes of the frame are copied into the array specified by *name*. Frames with errors are discarded; a count is kept for each type of error. Frames may be discarded for any of the following reasons: (1) CRC error, (2) frame too short (less than four bytes), (3) frame too long (buffer size exceeded), or (4) no receive buffer available. If a frame with a buffer attached was previously obtained with **rcvfrm**, but the buffer has not been released to the driver with **rtnrfrm**, that buffer is returned to the queue of empty receive buffers. At most one receive frame with no buffer attached is retained by the interpreter; if a new frame arrives before the frame with no buffer attached has been obtained with **rcvfrm**, the new frame is discarded.

rtnrfrm( )
Return a receive buffer. The current receive buffer (the one obtained by the most recent **rcvfrm** primitive) is returned to the driver. If there is no current receive buffer, the process is terminated in error.

rsxmtq( )
Reset the transmit-buffer queue. The sequence number assignment is removed from all transmit buffers. If a transmission is currently in progress, the transmission is aborted, if possible.

rtnxfrm(*primary*)
Return a transmit buffer. The transmit buffer currently attached to the sequence number specified by the value of the primary expression is returned to the driver and the sequence number assignment is removed from that buffer. If the specified sequence number does not have a buffer attached, the process is terminated in error. Transmit buffers must be returned in the same sequence in which they were obtained, otherwise the process is terminated in error.

1

setctl(*name*,*primary*)
Specify transmit-control information. The number of bytes specified by the primary expression are copied from the array specified by *name* and saved for use with subsequent **xmtfrm** or **xmtctl** primitives. If the transmitter is currently busy, the process is terminated in error.

xmtbusy( )
Test for transmitter busy. If a frame is currently being transmitted, the returned value is **true** (non-zero); otherwise the returned value is **false** (zero).

xmtctl( )
Transmit a control frame. If a transmission is not already in progress, a new transmission is initiated. The transmitted frame will contain the control information specified by the most recent **setctl** primitive, followed by a two-byte CRC. The CRC-CCITT polynomial ($x^{16}+x^{12}+x^5+1$) is used for the CRC calculation. The returned value is zero if a new transmission was initiated, otherwise non-zero.

xmtfrm(*primary*)
Transmit an information frame. If a transmission is not already in progress, a new transmission is initiated. The transmitted frame will contain the control information specified by the most recent **setctl** primitive, followed by the contents of the buffer which is currently attached to the sequence number specified by the value of the primary expression followed by a two-byte CRC. The CRC-CCITT polynomial ($x^{16}+x^{12}+x^5+1$) is used for the CRC calculation. The returned value is zero if a new transmission was initiated, otherwise non-zero. If the sequence number is greater than seven or the sequence number does not have a buffer attached, the process is terminated in error.

The following primitives are available with all versions of the interpreter:

dsrwait( )
Wait for modem-ready and then set modem-ready mode. The process delays until the modem-ready signal from the modem interface is asserted. If the modem-ready signal subsequently drops, the process is terminated. If **dsrwait** is never invoked, the modem-ready

signal is ignored.

exit(*primary*)
Terminate execution. The process is halted and the value of the primary expression is passed to the driver.

getcmd(*name*)
Get a command from the driver. If a command has been received from the driver since the last call to **getcmd**, four bytes of command information are copied into the array specified by *name* and a value of **true** (non-zero) is returned. If no command is available, the returned value is **false** (zero).

getopt( )
Get the script options. Script options are passed from the protocol driver to the VPM interpreter by means of the common synchronous interface (CSI). These bits are recorded by CSI; the most recent value is passed to the VPM interpreter each time the protocol script is started. This value is saved by the interpreter and can be retrieved as many times as desired using the *getopt* primitive. The low-order bit of this value is used by the BX.25 level 2 script (cslapb.r) to determine whether to use address A or address B.

pause( )
Return control to the dispatcher. This primitive informs the dispatcher that the virtual process may be suspended until the next occurrence of an event that might affect the state of the protocol for this line. Examples of such events are: (1) completion of an output transfer, (2) completion of an input transfer, (3) timer expiration, and (4) a buffer-in command from the driver. In a multi-line implementation, the **pause** primitive allows the process for a given line to give up control to allow the processor to service another line. In a single-line implementation this primitive has no effect.

snap(*name*)
Create a *snap* event record. Four bytes from the array specified by *name* are passed to the driver, which prefixes a time stamp and sequence number and creates a trace event record containing the data. If minor device 1 of the trace driver is currently open, the record is placed on the read queue for that device; otherwise the event record is discarded. The information passed via the *snap* primitive can be displayed using the *vpmsave* and *vpmfmt* commands (see *vpmsave*(1M)).

rtnrpt(*name*)
Return a report to the driver. Four bytes from the array specified by *name* are transferred to the driver. The process delays until the transfer is complete.

testop(*primary*)
Test for odd parity. The returned value is **true** (non-zero) if the value of the primary expression has odd parity, otherwise the returned value is **false** (zero).

timeout(*primary*)
Schedule or cancel a timer interrupt. If the value of the primary expression is non-zero, the current values of the program counter and stack pointer are saved and a timer is loaded with the value of the primary expression. The system call then returns immediately with a value of **false** (zero) as the returned value. The timer is

decremented each tenth of a second thereafter. If the timer is decremented to zero, the saved values of the program counter and stack pointer are restored and the system call returns with a value of **true** (non-zero). The effect of the timer interrupt is to return control to the code immediately following the **timeout** system call, at which point a non-zero return value indicates that the timer has expired. The **timeout** system call with a non-zero argument is normally written as the condition part of an **if** statement. A **timeout** system call with a zero argument value cancels all previous **timeout** requests, as does a **return** from the function in which the **timeout** system call was made. A **timeout** system call with a non-zero argument value overrides all previous **timeout** requests. The maximum permissible value for the argument is 255, which gives a timeout period of 25.5 seconds.

timer(*primary*)
Start a timer or test for timer expiration. If the value of the primary expression is non-zero, a software timer is loaded with the value of the primary expression and a value of **true** (non-zero) is returned. The timer is decremented each tenth of a second thereafter until it reaches zero. If the value of the primary expression is zero, the returned value is the current value of the timer; this will be **true** (non-zero) if the value of the timer is currently non-zero, otherwise **false** (zero). The timer used by this primitive is different from the timer used by the **timeout** primitive.

trace(*primary*[,*primary*])
The values of the two primary expressions and the current value of the script location counter are passed to the driver, which prefixes a sequence number and creates a trace event record containing the data. If minor device 0 of the trace driver is currently open, the record is placed on the read queue for that device; otherwise the event record is discarded. The information passed via the *trace* primitive can be displayed using the *vpmsave* and *vpmfmt* commands (see *vpmsave*(1M)). If the second argument is omitted, a zero is used instead. The process delays until the values have been accepted by the host computer.

Constants

A *constant* is a decimal, octal, or hexadecimal integer, or a single character enclosed in single quotes. A token consisting of a string of digits is taken to be an octal integer if the first digit is a zero, otherwise the string is interpreted as a decimal integer. If a token begins with **0x** or **0X**, the remainder of the token is interpreted as a hexadecimal integer. The hexadecimal digits include **a** through **f** or, equivalently, **A** through **F**.

Variables

Variable names may be used without having been previously declared. All names are global. All values are treated as 8-bit unsigned integers.

Arrays of contiguous storage may be allocated using the **array** declaration:

array *name*[*constant*]

where *constant* is a decimal integer. Elements of arrays can be referenced using constant subscripts:

name[*constant*]

Indexing of arrays assumes that the first element has an index of zero.

Names

A *name* is a sequence of letters and digits; the first character must be a letter. Upper- and lower-case letters are considered to be distinct. Names longer than 31 characters are truncated to 31 characters. The underscore (_) may be used within a name to improve readability, but is discarded by RATFOR.

Preprocessor Commands

If the **−m** option is omitted, comments, macro definitions, and file inclusion statements are written as in C. Otherwise, the following rules apply:

1. If the character # appears in an input line, the remainder of the line is treated as a comment.
2. A statement of the form:

   define(*name*,*text*)

   causes every subsequent appearance of *name* to be replaced by *text*. The defining text includes everything after the comma up to the balancing right parenthesis; multi-line definitions are allowed. Macros may have arguments. Any occurrence of **$***n* within the replacement text for a macro will be replaced by the *n*th actual argument when the macro is invoked.
3. A statement of the form:

   include(*file*)

   inserts the contents of *file* in place of the **include** command. The contents of the included file is often a set of definitions.

EXAMPLES

These examples require the use of the **−m** option.

```
# The function defined below transmits a frame in transparent BISYNC.
# A transmit buffer must be obtained with getxbuf before the function
# is invoked.
#
# Define symbolic constants:
#
define(DLE,0x10)
define(ETB,0x26)
define(PAD,0xff)
define(STX,0x02)
define(SYNC,0x32)
#
# Define a macro with an argument:
#
define(xmtcrc,{crc16($1); xmt($1);})
#
# Declare an array:
#
array crc[2];
#
# Define the function:
#
function xmtblk()
        crcloc(crc);
        xsom(SYNC);
        xmt(DLE);
        xmt(STX);
```

1

```
        while(get(byte)==0){
                if(byte == DLE)
                        xmt(DLE);
                xmtcrc(byte);
        }
        xmt(DLE);
        xmtcrc(ETB);
        xmt(crc[0]);
        xmt(crc[1]);
        xeom(PAD);
end
#
# The following example illustrates the use of macros to simulate a
# function call with arguments.
#
# The macro definition:
#
define(xmtctl,{c=$1;d=$2;xmtctl1()})
#
# The function definition:
#
function xmtctl1()
        xsom(SYNC);
        xmt(c);
        if(d!=0)
                xmt(d);
        xeom(PAD);
end
#
# Sample invocation:
#
function test()
        xmtctl(DLE,0x70);
end
```

1

FILES

sas_temp*	temporaries
/tmp/sas_ta??	temporary
/tmp/sas_tb??	temporary
/usr/lib/vpm/pass*	compiler phases
/usr/lib/vpm/pl	compiler phase
/usr/lib/vpm/vratfor	compiler phase
/lib/cpp	preprocessor
/usr/bin/m4	preprocessor
/bin/kasb	KMC11-B assembler
/usr/lib/vpm/bisync/*	interpreter source for the BISYNC interpreter
/usr/lib/vpm/hdlc/*	interpreter source for the HDLC interpreter

SEE ALSO

m4(1), ratfor(1), vpmsave(1M), vpm(7).
C Reference Manual by D. M. Ritchie.
RATFOR—A Preprocessor for a Rational Fortran by B. W. Kernighan.
The M4 Macro Processor by B. W. Kernighan and D. M. Ritchie.
Software Tools by B. W. Kernighan and P. J. Plauger (pp. 28-30).

NAME

vpmc — compiler for the virtual protocol machine

SYNOPSIS

vpmc [**−m**] [**−c**] [**−s** sfile] [**−l** lfile] [**−o** ofile] file

DESCRIPTION

Vpmc is the compiler used for C programs written to describe communications link protocols. The output of *vpmc* is a load module for the virtual protocol machine (VPM), which is a software construct for implementing communications link protocols (e.g., BISYNC) on the UN53) device. VPM is implemented by a program in the UN53 which cooperates with a driver in the UNIX host computer to transfer data over a communications link in accordance with a specified link protocol. UNIX user processes transfer data to or from a remote terminal or computer system through VPM using normal UNIX *open*, *read*, *write*, and *close* operations. The VPM program in the UN53 provides error control and flow control using the conventions specified in the protocol.

Options

The meanings of the command-line options are:

- **−m** Use *m4*(1) instead of *cpp* as the macro preprocessor.
- **−c** Compile only (suppress the assembly and linking phases).
- **−s** *sfile* Save the generated assembly language on file *sfile*.
- **−l** *lfile* Produce an assembly-language listing on file *lfile*.
- **−o** *ofile* Write the executable object file on file *ofile* (default **a.out**).

These options may be given in any order.

Programs

Input to *vpmc* consists of a C program with one or more function definitions. The first defined function is invoked (on command from the UNIX VPM driver) to begin program execution.

System Calls

A VPM program interacts with a communications device and a driver in the host computer by means of system calls (primitives).

The following primitives are available:

atoe(*primary*)

Translate ASCII to EBCDIC. The returned value is the EBCDIC character that corresponds to the ASCII character represented by the value of the primary expression. The translation tables reflect the prejudices of a particular installation.

crc16(*primary*)

The value of the primary expression is combined with the cyclic redundancy check-sum at the location passed by a previous **crcloc** system call. The CRC-16 polynomial ($x^{16}+x^{15}+x^{2}+1$) is used for the check-sum calculation.

crcloc(*name*)

The two-byte array starting at the location specified by *name* is cleared. The address of the array is recorded as the location to be updated by subsequent **crc16** system calls.

etoa(*primary*)

Translate EBCDIC to ASCII. The returned value is the ASCII character that corresponds to the EBCDIC character represented by the value of the primary expression. The translation tables reflect the prejudices of a particular installation.

get(*lvalue*)
Get a byte from the current *transmit* buffer. The next available byte, if any, is copied into the location specified by *lvalue*. The returned value is zero if a byte was obtained, otherwise it is non-zero.

getrbuf(*name*)
Get (open) a *receive* buffer. The returned value is zero if a buffer is available, otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by *name*. The array should be large enough to hold at least three bytes. The meaning of the buffer parameters is driver-dependent. If a receive buffer has previously been opened via a **getrbuf** call but has not yet been closed via a call to **rtnrbuf**, that buffer is reinitialized and remains the current buffer.

getxbuf(*name*)
Get (open) a *transmit* buffer. The returned value is zero if a buffer is available, otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by *name*. The array should be large enough to hold at least three bytes. The meaning of the buffer parameters is driver-dependent. If a transmit buffer has previously been opened via a **getxbuf** call but has not yet been closed via a call to **rtnxbuf**, that buffer is reinitialized and remains the current buffer.

put(*primary*)
Put a byte into the current *receive* buffer. The value of the primary expression is inserted into the next available position, if any, in the current receive buffer. The returned value is zero if a byte was transferred, otherwise it is non-zero.

rcv(*lvalue*)
Receive a character. The process delays until a character is available in the input silo. The character is then moved to the location specified by *lvalue* and the process is reactivated.

rsom(*constant*)
Skip to the beginning of a new *receive* frame. The receiver hardware is cleared and the value of *constant* is stored as the receive sync character. This call is used to synchronize the local receiver and remote transmitter when the process is ready to accept a new receive frame.

rtnrbuf(*name*)
Return a *receive* buffer. The original values of the buffer parameters for the current receive buffer are replaced with values from the array specified by *name*. The current receive buffer is then released to the driver.

rtnxbuf(*name*)
Return a *transmit* buffer. The original values of the buffer parameters for the current transmit buffer are replaced with values from the array specified by *name*. The current transmit buffer is then released to the driver.

xeom(*constant*)
Transmit end-of-message. The value of the constant is transmitted, then the transmitter is shut down.

xmt(*primary*)
Transmit a character. The value of the primary expression is

1

transmitted over the communications line. If the output silo is full, the process waits until there is room in the silo.

xsom(*constant*)
Transmit start-of-message. The transmitter is cleared, then the value of *constant* is transmitted six times. This call is used to synchronize the local transmitter and the remote receiver at the beginning of a frame.

dsrwait( )
Wait for modem-ready and then set modem-ready mode. The process delays until the modem-ready signal from the modem interface is asserted. If the modem-ready signal subsequently drops, the process is terminated. If **dsrwait** is never invoked, the modem-ready signal is ignored.

exit(*primary*)
Terminate execution. The process is halted and the value of the primary expression is passed to the driver.

getcmd(*name*)
Get a command from the driver. If a command has been received from the driver since the last call to **getcmd**, four bytes of command information are copied into the array specified by *name* and a value of **true** (non-zero) is returned. If no command is available, the returned value is **false** (zero).

1

pause( )
Return control to the dispatcher. This primitive informs the dispatcher that the virtual process may be suspended until the next occurrence of an event that might affect the state of the protocol for this line. Examples of such events are: (1) completion of an output transfer, (2) completion of an input transfer, (3) timer expiration, and (4) a buffer-in command from the driver. In a multi-line implementation, the **pause** primitive allows the process for a given line to give up control to allow the processor to service another line. In a single-line implementation this primitive has no effect.

snap(*name*)
Create a *snap* event record. Four bytes from the array specified by *name* are passed to the driver, which prefixes a time stamp and sequence number and creates a trace event record containing the data. If minor device 1 of the trace driver is currently open, the record is placed on the read queue for that device; otherwise the event record is discarded. The information passed via the *snap* primitive can be displayed using the *vpmsnap* command (see *vpmstart*).

rtnrpt(*name*)
Return a report to the driver. Four bytes from the array specified by *name* are transferred to the driver. The process delays until the transfer is complete.

testop(*primary*)
Test for odd parity. The returned value is **true** (non-zero) if the value of the primary expression has odd parity, otherwise the returned value is **false** (zero).

timeout(*primary*)
Schedule or cancel a timer interrupt. If the value of the primary expression is non-zero, the current values of the program counter

and stack pointer are saved and a timer is loaded with the value of the primary expression. The system call then returns immediately with a value of **false** (zero) as the returned value. The timer is decremented each tenth of a second thereafter. If the timer is decremented to zero, the saved values of the program counter and stack pointer are restored and the system call returns with a value of **true** (non-zero). The effect of the timer interrupt is to return control to the code immediately following the **timeout** system call, at which point a non-zero return value indicates that the timer has expired. The **timeout** system call with a non-zero argument is normally written as the condition part of an **if** statement. A **timeout** system call with a zero argument value cancels all previous **timeout** requests, as does a **return** from the function in which the **timeout** system call was made. A **timeout** system call with a non-zero argument value overrides all previous **timeout** requests. The maximum permissible value for the argument is 255, which gives a timeout period of 25.5 seconds.

timer(*primary*)
Start a timer or test for timer expiration. If the value of the primary expression is non-zero, a software timer is loaded with the value of the primary expression and a value of **true** (non-zero) is returned. The timer is decremented each tenth of a second thereafter until it reaches zero. If the value of the primary expression is zero, the returned value is the current value of the timer; this will be **true** (non-zero) if the value of the timer is currently non-zero, otherwise **false** (zero). The timer used by this primitive is different from the timer used by the **timeout** primitive.

trace(*primary*[,*primary*])
The values of the two primary expressions and the current value of the script location counter are passed to the driver, which prefixes a sequence number and creates a trace event record containing the data. If minor device 0 of the trace driver is currently open, the record is placed on the read queue for that device; otherwise the event record is discarded. The information passed via the *trace* primitive can be displayed using the *vpmtrace* command (see *vpmsave*). If the second argument is omitted, a zero is used instead. The process delays until the values have been accepted by the host computer.

Preprocessor Commands

If the **−m** option is omitted, comments, macro definitions, and file inclusion statements are written as in C. Otherwise, the following rules apply:

1. If the character # appears in an input line, the remainder of the line is treated as a comment.

2. A statement of the form:

   define(*name*,*text*)

   causes every subsequent appearance of *name* to be replaced by *text*. The defining text includes everything after the comma up to the balancing right parenthesis; multi-line definitions are allowed. Macros may have arguments. Any occurrence of **$***n* within the replacement text for a macro will be replaced by the *n*th actual argument when the macro is invoked.

3. A statement of the form:

   include(*file*)

inserts the contents of *file* in place of the **include** command. The contents of the included file is often a set of definitions.

SEE ALSO

m4(1), vpmset(1M).
C Reference Manual by D. M. Ritchie.
The M4 Macro Processor by B. W. Kernighan and D. M. Ritchie.
Software Tools by B. W. Kernighan and P. J. Plauger (pp. 28-30).

NAME

vpmsave, vpmfmt — save and print VPM event traces

SYNOPSIS

/etc/vpmsave mask device

/etc/vpmfmt [−t]

DESCRIPTION

Vpmsave opens the minor device of the trace driver specified by *device*, enables the channels specified by *mask* (octal), and then reads event records and writes them to its standard output (unformatted) until killed. Bit 0 of *mask* enables channel zero, bit 1 channel one, etc. The Common Synchronous Interface (CSI) routines and the CSI-based protocol drivers use the CSI index number (modulo 16) to select their trace channels. Each protocol driver provides an *ioctl* command that can be used to get this number.

1

Vpmfmt reads its standard input, which it assumes was generated by *vpmsave*, and prints it (formatted) to its standard output until killed. The **−t** option when used with the level 2 script *trcslapb.r* provides a detailed event trace of the operation of level 2 of BX.25. When used with other scripts the result may not be meaningful.

Support for the commands *vpmtrace* and *vpmsnap* has been dropped. The function of *vpmtrace* can be obtained using *vpmsave* and *vpmfmt* as follows:

vpmsave mask device | vpmfmt

where *device* is the name of minor device 0 of the trace driver.

The event records generated by calls to the *vpmc snap* primitive are now directed to minor device 0 along with other event records. The need to keep the *snap* event record separate no longer exists since all event records now contain a time stamp.

EXAMPLE

vpmsave mask device > t &

vpmfmt < t

SEE ALSO

vpmc(1M), trace(7), vpm(7).

NAME

vpmset, vpmstart — connect/load VPM drivers and programmable communication devices

SYNOPSIS

/etc/vpmset [**−b**] [**−d**] [**−s**] tdev pdev [lineno]

/etc/vpmstart [**−r**] device n [filen]

DESCRIPTION

The *vpmset* command provides a means for associating dynamically a VPM protocol driver minor device with a particular synchronous line on a programmable communication device (PCD). *Tdev* is the protocol driver minor device name; *pdev* is the PCD minor device name; and *lineno* is the number of a synchronous line if a DEC KMS11 was specified. Until these connections have been made, a user program cannot open the VPM protocol driver minor device for reading or writing.

The **−b** option causes bit zero of the protocol option bits to be set. Some protocol scripts such as *cslapb.r* (level 2 of BX.25) use this bit to specify address B as the local address. These bits are stored by the protocol driver and passed to the PCD when the VPM protocol minor device is opened for reading and/or writing. These bits are available to the protocol script via the *getopt* primitive.

The **−d** option disconnects the VPM protocol minor device from the synchronous line on the PCD. This disconnect will fail if the VPM protocol minor device is open for reading and/or writing.

The **−s** option prints to *stdout* a message indicating which Common Synchronous Interface (CSI) index is associated with the protocol minor device. This number *modulo* 16 indicates the channel number used for tracing events with the *vpmsave*(1M) command.

Vpmstart writes *filen* (**a.out** by default) to the KMC11(DEC) or UN53(3B20S) specified by *device*. The argument *n* is a magic number that the PCD driver saves to identify the program. This number is checked when the VPM driver is opened to provide some assurance that the program loaded into the PCD is the one expected. The magic number for any standard VPM program running in the PCD is 6 (7 indicates a V.35 interface on the 3B20S). While *filen* may be any file that is executable by the PCD, it will normally have been prepared using *vpmc*(1M).

The PCD control program waits for the VPM protocol minor device to be opened for reading and/or writing before beginning execution of the protocol script. The **−r** option may be specified only when using a DEC KMC11. When this option is specified, the queue of commands to the KMC11 driver is not flushed prior to starting the PCD program. This option must be used to reload the KMC11 when recovering from a power-fail.

SEE ALSO

vpm(7), vpmc(1M).

NAME

vpmtest — test KMC lines

SYNOPSIS

/etc/vpmtest −t top **−k** kmc **−n** line

DESCRIPTION

Vpmtest performs a series of loopback tests on a specified synchronous line interface of a specified KMC. *Top* is a path name that specifies the VPM protocol driver minor device to be used for the test. *Kmc* is the path name for the KMC on which the test is to be run. *Line* specifies the particular line number (0-7) on a KMS; for a KMC equipped with a single-line line interface *line* should be 0. The KMC must have been previously loaded with the LAPB script (*cslapb.r*) using *vpmstart*. The *top* device must not be open for reading or writing; this is the same restriction as for doing a *vpmset*. The equivalent of a *vpmset* is performed to associate the specified *top* device with the specified KMC and line number.

The first test uses the maintenance mode of the line unit hardware to perform an internal loopback test; this test is independent of the modem to which the line is attached. A failure at this point indicates a problem with the KMC or line unit hardware or with the VPM or KMC software. The second test requires the local (near end) modem to be placed in the analog loopback mode; on a WECO 209A dataset this is accomplished by depressing the AL button on the front of the dataset. A failure at this point indicates a problem with the modem or with the interface between the modem and the line unit; a faulty or disconnected cable is the most likely possibility. The third test performs a loopback from the remote (far end) modem. This requires placing the *remote* modem in the digital loopback mode; on a WECO 209A dataset this is accomplished by depressing the DL button on the front of the dataset. The local (near end) modem should be in the normal mode for this test. A failure at this point indicates trouble with one or both of the modems or with the telephone line connecting them. At the end of each test the VPM error counters associated with the particular line are printed.

EXAMPLE

```
/etc/vpmtest −t /dev/vpm1 −k /dev/kmc1 −n 7
```

SEE ALSO

kmc(7), vpm(7).

NAME

wall — write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

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

Broadcast Message from ...

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

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

FILES

/dev/tty*

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

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

NAME

whodo — who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

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

SEE ALSO

ps(1), who(1).

1

NAME

x25pvc, x25lnk — install, remove, or get status for a BX.25 minor device or link

SYNOPSIS

/etc/x25pvc options

/etc/x25lnk options

DESCRIPTION

X25pvc may be used to install or remove a BX.25 Permanent Virtual Circuit (PVC) on a specified BX.25 interface (*link*), or to display the status of a specified BX.25 minor device (*slot*). Exactly one of the following *options* (i.e. **−i**, **−r**, **−s**, **−e**) must be used:

−i [**−S**] [**−R**] [**−N**] **−m** *slotname* **−c** *chno* **−l** *linkno*

Slotname is a path name that specifies a BX.25 minor device (slot). If that minor device is currently connected to some logical channel on some BX.25 interface (link), then first that minor device will be removed, if possible (see the **−r** *option*). If that minor device is now available, it is connected to logical channel *chno* on link number *linkno*. *Chno* must be in the range of 1 to 4,095 and must not currently be in use for any other BX.25 minor device associated with that link. Exactly one of three session-establishment *options* must be used: **−S**, which is the preferred *option*, indicates that session-layer connect/accept/disconnect qualified data messages are to be used; **−R** indicates that RESET in-order/out-of-order packets will be recognized but not transmitted; **−N** indicates that the "no protocol" session mode will be used. The **−R** and **−N** *options* are provided only for compatibility with non-UNIX implementations of BX.25. This command will fail if the link, channel, or minor device number is out of range, or if the slot is in use.

−r *slotname*

Remove the association between BX.25 minor device *slotname* and the link and channel number to which it is currently connected. The command will fail if the minor device number is out of range, the slot is not installed, the slot is open, packets are waiting to be transmitted, or there are unacknowledged packets outstanding.

−s *slotname*

Print abbreviated status information for BX.25 minor device *slotname*. The information printed consists of *slotname*, the logical channel number, the link number, and the session-establishment *option*. This command will fail if the minor device number is out of range or the slot is not installed.

−e *slotname*

Print extended status information for BX.25 minor device *slotname*. The information printed consists of most of the information that is stored in the internal data structures associated with this device. This information is useful for determining the state of the PVC associated with this device when hardware or software anomalies are suspected and is intended for use by developers and sophisticated users. This command will fail if the minor device number is out of range or the slot is not installed.

X25lnk is used to attach, detach, activate, deactivate, get status for and perform a changeover on a specified BX.25 interface (link). Exactly one of the following *options* (i.e. **−a**, **−d**, **−i**, **−h**, **−s**, **−c**) must be used:

−a [**−k**] **−m** *device* [**−n** *lineno*] [**−o** *modctl*] **−l** *linkno*

Attach the BX.25 link that is specified by *linkno* to the level 2 device whose name is *device*. This command makes the necessary connections between data structures. *Linkno* is the number of the BX.25 link to be attached; links are numbered starting with 0. If a line number must be specified for the device, (e.g. the device is a KMS11), the **−n** *option* is used. If the *device* is a KMS11 and modem control is needed, the **−o** *option* is used and *modctl* is the path name of the DM11-BA modem control unit associated with a particular KMC as part of a KMS11. If the **−k** *option* is used, this command will attach *device* as a backup for link *linkno*; otherwise, *device* is the primary. This command will fail if the link number is out of range, the link is already attached, or the device is already attached.

−d [**−k**] **−l** *linkno*

Detach the BX.25 link that is specified by *linkno*. This command removes the logical connections that were made by the **−a** *option*. If the **−k** *option* is used, this command will detach the backup device associated with *linkno*; otherwise the primary device is detached. This command will fail if the link number is out of range, the device is not attached, or the device has not been halted.

−i [**−k**] [**−b**] [**−p** *pktsize*] [**−f**] **−l** *linkno*

Activate the BX.25 link that is specified by *linkno*. The **−b** *option* specifies that the link-level protocol will use address B; the default is address A. The **−p** *option* specifies the packet size; if it is used, *pktsize* must be a number that is a power of 2 and lies between 16 and 1,024 inclusive. The default packet size is 128. The **−f** *option*, which is used only on the 3B20S, indicates that the speed of the device associated with link *linkno* is greater than 9.6KB. If the **−k** *option* is used, this command will start the level 2 protocol on the backup device associated with link *linkno*; otherwise, the BX.25 level 2 and level 3 protocols will be started on the primary device associated with link *linkno*. This command will fail if the link number is out of range, the link is not attached, the device is already started, or the packet size is invalid.

−h [**−k**] **−l** *linkno*

Halt the link specified by *linkno*. If the **−k** *option* is used, the level 2 protocol on the backup device will be stopped provided the level 3 protocol is not running on the backup device. If the **−k** *option* is not used, the level 3 protocol is stopped (wherever it is running) and the level 2 protocol on the primary device is stopped. If a backup device has been attached and started, the level 2 protocol on the backup will also be stopped. This command will fail if the link number is out of range or the link is not attached.

−s **−l** *linkno*

Print the status of the link specified by *linkno*. The information printed consists of the link number, the packet size used on that link, and internal status information including whether or not the level 2 queue is full, the restart state of the link, whether or not the high and low priority queues are empty, and whether or not a restart packet is on the level 2 queue. Information about level 2 status is printed for the primary device and also for the backup device, if it has been attached. The value of *csidev* modulo 16 indicates the channel number used for tracing events with the *vpmsave* command. This command will fail if the link number is out of range or the link is not attached to either a primary or backup device.

−c **−l** *linkno*

Changeover to the standby synchronous device associated with link *linkno*. If the standby device is synchronized at level 2, level 3 will now run on that device. This command will fail if the link number is out of range, the link is not attached to both a primary and backup device, or a backup device has not been started.

SEE ALSO

vpmsave(1M), nc(7), x25(7).

1

NAME

intro — introduction to special files

DESCRIPTION

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

BUGS

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

7

NAME

acu, dn — Automatic Call Unit (ACU) interface

DESCRIPTION

The ACU drivers support *open, close,* and *write* system calls. In addition, the *tn8* driver on the 3B20S supports an *ioctl* system call. The **acu?** and **dn?** files are write-only. The *write* system call sends the telephone number to be dialed to the ACU. The permissible codes are:

0-9	dial 0-9
*** or :**	dial *****
# or ;	dial **#**
—	4 second delay for second dial tone
e or <	end-of-number
w or =	wait for secondary dial tone
f	flash off hook for 1 second

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

The *ioctl* system call (*tn8* only) is invoked as follows:

```
#include <sys/acu.h>
int fildes, cmd;
struct acutab *acutp;
ioctl (fildes, cmd, acutp);
```

Acutab is a table specifying the connections between ACU minor devices and communication lines:

```
struct acutab {
        int minor;
        int unit;
        int port;
        int line;
} acutab[NACU];
```

The *NACU* parameter is a constant from *acu.h* that specifies the number of lines the TN8 ACUs can dial out on.

The *ioctl cmds* are:

ACUSDEV—Specify a connection between an ACU minor device and a telephone line. This command makes an entry in *acutab*, the table that specifies associations between ACU minor devices and dial-out lines. Before the ACUs can be used, and after any ACU reconfiguration, this table must be sent to the ACU peripheral controller via the ACUSTART command.

ACUSTART—Connect ACU minor devices to telephone lines. This command informs the ACU peripheral controller of the connections set up by the ACUSDEV command and enables it.

SEE ALSO

acuset(1M).

FILES

/dev/acu?	(3B20S only)
/dev/tn8	(3B20S only)
/dev/dn?	(DEC only)

NAME

cat − phototypesetter interface

DESCRIPTION

Cat provides the interface to a Wang Laboratories, Inc. C/A/T phototypesetter. Bytes written on the file specify font, size, and other control information as well as the characters to be flashed. The coding will not be described here.

Only one process may have this file open at a time. It is write-only.

FILES

/dev/cat

SEE ALSO

troff(1).

NAME

dgn — on-line diagnostic interface

DESCRIPTION

Files in the directory **/dev/dgn** provide the interface between on-line diagnostic commands and device drivers. The files in this directory are intended to be accessed only by diagnostic commands.

FILES

/dev/dgn/*

SEE ALSO

abt(1M), chmap(1M), dgn(1M), dstart(1M), rmv(1M), rst(1M), sta(1M).

WARNING

Diagnostic commands are intended for use only by trained hardware maintenance personnel.

7

NAME

dmc — communications link with built-in DDCMP protocol

DESCRIPTION

The DMC11 allows local connection of PDP-11 systems over high-speed (1Mb or 56kb) links and remote connection over leased (up to 19.2kb) or dial-up (up to 4,800b) lines. It implements in hardware the DDCMP data-link protocol, which includes error control. This driver handles two DMC11 devices.

FILES

/dev/dmc

BUGS

There are quite a few bugs in the DEC microcode for the different versions of the DMC11.

NAME

dmk — DM11-BA modem control multiplexor

DESCRIPTION

The files **/dev/dmk?** are used to access DM11-BA modem control units. Each DM11-BA provides modem control and status information for eight synchronous lines. The DM11-BA is an optional component of the KMS11 communications processor (see *vpm*(7)). Since the VPM software for the KMS11 does not provide any access to the DM11-BA, it is necessary to use the *dmk* driver if modem control is required with the KMS11.

The *ioctl*(2) function is used to provide access to the basic modem control capabilities:

```
#include <sys/dmk.h>
ioctl (fildes, command, arg)
struct dmkctl {
        short   line;
        short   mode;
} *arg;
```

The only *command* available is DMKSETM. The effect of this command is to set the control leads in the modem interface for the line (0-7) specified by *line* to the state specified by *mode*. The bits in *mode* specify control leads to be asserted as follows:

Name	*Bit*	*Meaning*
DMKDTR	002	Data Terminal Ready
DMKRTS	004	Request to Send
DMKNS	010	New Sync

FILES

/dev/dmk?

SEE ALSO

x25pvc(1M), vpm(7).
"KMS11-A/B Communications Processor Option Description", YM-C126C-00, Digital Equipment Corporation.
"DM11-BA Modem Control Multiplexor Option Description", YM-C138C-00, Digital Equipment Corporation.

NAME

dsk — 3B20S moving-head disk

DESCRIPTION

The files **dsk0, ..., dsk8** refer to sections of the disk drive unit number 0. The files **dsk10, ..., dsk18** refer to drive unit number 1, etc. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	1	494304
1	101	433504
2	251	342304
3	326	296704
4	476	205504
5	551	159904
6	701	68704
7	775	23712
8	0	495520

The start address is a cylinder address, with each cylinder containing 608 blocks on the 300 megabyte drive. Also it should be noted that the first cylinder is reserved for booting and the last cylinder for diagnostics.

The **dsk** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw disk files begin with **rdsk** and end with a number which selects the same disk section as the corresponding **dsk** file.

In raw I/O the buffer must begin on a word boundary, and counts must be a multiple of 64 bytes.

In addition to the sections defined above two other special files have been created to define areas on the disk. The first is **/dev/dump**?, where *?* is the disk unit number, which provides raw access to the section on the disk where dumps will be put. The second file is **/dev/boot**? which provides raw access to the boot section.

FILES

/dev/dsk*, /dev/rdsk*, /dev/dump?, /dev/boot?

NAME

du — DU-11 synchronous line interface

DESCRIPTION

The files **du0**, **du1**, etc., represent interfaces to synchronous modems such as the Bell System 200-series synchronous DATA-PHONE® sets. *Read* and *write* calls to **du?** are unlimited, but work best when restricted to less than 512 bytes. Each *write* call is sent as a single record. Seven bits from each byte are written, along with an eighth, odd-parity, bit. The "sync" characters must be supplied by the user. Each *read* call returns the characters read from a single record. Seven bits are returned unaltered; the eighth bit is set if the byte was not received in odd parity. An error is returned if *data-set ready* is not present.

FILES

/dev/du?

SEE ALSO

acu(7).

7

NAME

dz, dzb, dh — DZ-11, DZ-11/KMC-11B, DH-11 asynchronous multiplexers

DESCRIPTION

Each line attached to a DH-11 or DZ-11 communications multiplexer behaves as described in *termio*(7). Input and output for each line may independently be set to run at any of 16 speeds; see *termio*(7) for the encoding. (For DZ-11 lines, output speed is always the same as input speed. The *200* speed and the two externally clocked speeds (*exta*, *extb*) are missing on the DZ-11.) The behavior of *dzb* lines is indistinguishable from that of *dz* lines, except that on the *dzb* backspace delays are implemented using fill characters (rubouts) instead of timed delays.

Note that the DH-11 is considered obsolete and is not supported on the VAX-11/780.

FILES

/dev/tty*

SEE ALSO

kmc(7), termio(7).

7

NAME

emulio — 3270 emulation interface

SYNOPSIS

#include <sys/file.h>

DESCRIPTION

3270 emulation makes use of the UN53 driver and associated VPM software to simulate a 3270-type controller (e.g., a 40/4 controller). Users may communicate with a remote host as a 3270-type peripheral (e.g., a 40/4 terminal) through terminal emulation special files. 3270 emulation consists of two UNIX drivers and a protocol script. The *emc* driver provides an administrative interface to the "controller". The *em* driver provides the users interface to the remote system. The protocol script runs in the UN53 and handles the line protocol. Both ASCII (**/lib/a3270scr**), and EBCDIC (**/lib/e3270scr**) scripts are available. All structures described here are defined in **<sys/em.h>**.

Administrative Interface

This section describes the interface to the *emc* driver. Each emulated controller is represented by a character special file (**/dev/emc?**). To use 3270 emulation the *emc* device must be associated (using *vpmset*) with the physical device to be used. For example to associate controller 2 and device 3:

```
/etc/vpmset /dev/emc2 /dev/un53.3
```

The appropriate protocol script must also be loaded (using *vpmstart*) on the physical device. For example, to load the EBCDIC script on device 3:

```
/etc/vpmstart /dev/un53.3 6 /lib/e3270scr
```

The remaining administrative functions are performed using *ioctl*(2) calls on the *emc* device. The valid *ioctl requests* and the corresponding *arg* are as follows:

EMCNTRS	Return script error counters. *Arg* must be the address of a counters structure (see below).
EMINFO	Return information about this controller. *Arg* must be the address of a information structure (see below).
EMPOLL	Set the polling character to *arg*. The default is *space* (controller 0).
EMSELECT	Set the selection character to *arg*. The default is — (controller 0).
EMEOTD	Set the time delay before transmitting EOT's to *arg*/10 seconds. The default is 2 seconds.
EMSTART	Start the corresponding protocol script.
EMHALT	Halt the corresponding protocol script.
EMSETFL	Set the controller flags as specified in *arg*.
EMCLRFL	Clear the controller flags as specified in *arg*.

If *request* is EMCNTRS, *arg* must be the address of a structure with the following format:

```
struct emcntrs {
        short   rtmout;   /* 3 sec rcv timeouts */
        short   xtmout;   /* 1.5 sec timeouts, getxbuf */
        short   ptmout;   /* 1.5 sec timeout on POLL */
        short   rcvnak;   /* NAK's received */
        short   xmtnak;   /* NAK's transmitted */
        short   rcvenq;   /* ENQ's received */
        short   xmtenq;   /* ENQ's transmitted */
        short   crcerrs;  /* CRC errors */
        short   roflo;    /* receive blocks to large */
        short   rgarb;    /* Junk receive messages */
        short   xgarb;    /* Garbage xmit buffers */
        short   rparerr;  /* Bad parity on rcv blocks */
        short   xparerr;  /* Bad parity on xmit bufs */
        short   lrcerrs;  /* LRC errors */
        short   eotrmsg;  /* EOT's when block expected */
        short   cmgarb;   /* Junk in LISTEN state */
        short   gmgarb;   /* Junk in TRASH state */
};
```

If *request* is EMINFO, *arg* must be the address of a structure with the following format:

```
struct eminfo {
        short   em_flags;   /* Flags */
        short   em_code;    /* Code */
        char    em_staid;   /* Polling character */
        char    em_termid;  /* Selection character */
        char    em_rdev;    /* Real device */
};
```

The values used in the *em_flags* field are:

```
#define  EM_ASC     0x01   /* The controller is ASCII */
#define  EM_RUN     0x02   /* The controller is usable */
#define  EM_STATS   0x04   /* The cntrs are available */
#define  EM_RBUF    0x08   /* Rcv buffers are needed */
#define  EM_SCERR   0x10   /* Script error (ERRTERM) */
#define  EM_STERR   0x20   /* Startup error */
#define  EM_TRACE   0x40   /* Script tracing flag */
#define  EM_STOK    0x80   /* Started OK */
```

In general, all administrative functions can be performed from user level by using the *emulcntrl*(1M) and *emulstat*(1M) commands.

User Interface

This section describes the interface to the *em* driver. The *em* driver represents each terminal on a controller as a character special file (*/dev/emt**). Up to 32 terminals are allowed per controller. The minor device number of each terminal specifies the controller and terminal; The low-order 8 bits specify the terminal number, and the remaining high-order bits specify the controller number. The id character for each terminal is determined as follows:

Term	Id	Term	Id
0	SP	16	&
1	A	17	J
2	B	18	K
3	C	19	L
4	D	20	M
5	E	21	N
6	F	22	O
7	G	23	P
8	H	24	Q
9	I	25	R
10	[¢	26	] !
11	.	27	$
12	<	28	*
13	(	29	)
14	+	30	;
15	! |	31	^ ¬

Where 2 characters appear, the second is EBCDIC. UNIX user processes use the terminal files to simulate active terminals. To start a terminal the appropriate device is opened. Data transfers are performed using *read*(2), and *write*(2). The EMCNTRS and EMINFO *ioctl*(2) requests described in the previous section can be used in the same way with terminal files. The general operations are performed as follows:

Starting The *open*(2) call will wait for a physical connection to be established before returning. Immediate return is obtained using the FNDELAY *open* flag. This call will fail if the connection is not available (with FNDELAY flag), or the terminal is already in use.

Transfers Once a terminal has been opened, a user process may transmit a "screen" using *write*(2). Data written must be in the expected form (control fields, etc.) and must be surrounded by the start-of-text (STX) and end-of-text (ETX) or end-of-block (ETB) characters. When using the ETB end character, subsequent writes must complete the block according to the block protocol (i.e., the last block must end in ETX). The two bytes following the STX character (in the first block of a message) are reserved for the station and device identification characters. The proper values of these bytes are inserted by the driver, however the space in the block must be provided by the user. All block check characters are added internally. Remote messages are received using *read*(2). The format of these blocks is the same as received from the remote system (i.e., the blocks are passed directly). All line protocol, and verification is performed internally. Reads and writes will fail if the communications line has dropped.

Stopping To deactivate a terminal, the corresponding device is simply closed. Currently, any messages to be received by a deactivated (closed) terminal device are discarded.

FILES

/dev/emc?	3270 emulation controller devices
/dev/emt*	3270 emulation terminals
/lib/a3270scr	ASCII 3270 script
/lib/e3270scr	EBCDIC 3270 script

SEE ALSO

emulcntrl(1M), emulload(1M), emulstat(1M), vpmset(1M).

NAME

err — error-logging interface

DESCRIPTION

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

FILES

/dev/error special file

SEE ALSO

errdemon(1M).

7

NAME

gd — general driver for moving-head disks

DESCRIPTION

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

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

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

FILES

/dev/rp*, /dev/rrp*

SEE ALSO

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

7

NAME

gt — general driver for tape drives

DESCRIPTION

Gt provides a general interface to the TE16 and TU8 tape drives. In addition to the capability of mixing these mediums on the same controller, the driver will handle up to two controllers.

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

The configuration name of **tu1678** should be specified when generating a system with *config*(1M).

FILES

/dev/mt*, /dev/rmt*

SEE ALSO

config(1M), master(4). ht(7), tu78(7),

NAME

hm — RM05 moving-head disk

DESCRIPTION

The files **rp0, ..., rp7** refer to sections of the RM05 disk drive 0. The files **rp10, ..., rp17** refer to drive 1, etc. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	0	24320
1	40	476064
2	160	403104
3	280	330144
4	400	257184
5	520	184224
6	640	111264
7	0	500384

The start address is a cylinder address, with each cylinder containing 608 blocks. It is extremely unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter.

The **rp** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RP files begin with **rrp** and end with a number which selects the same disk section as the corresponding **rp** file.

7

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/rp*, /dev/rrp*

SEE ALSO

gd(7), hp(7), rm80(7), rp07(7).

NAME

hp — RP04/RP05/RP06 moving-head disk

DESCRIPTION

The files **rp0, ..., rp7** refer to sections of the RP04/RP05/RP06 disk drive 0. The files **rp10, ..., rp17** refer to drive 1, etc. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

RP04/05

section	*start*	*length*
0	0	18392
1	44	153406
2	201	87780
3	358	22154
4	—	—
5	—	—
6	—	—
7	0	171798

RP06

section	*start*	*length*
0	0	18392
1	44	322278
2	201	256652
3	358	191026
4	515	125400
5	672	59774
6	—	—
7	0	340670

The start address is a cylinder address, with each cylinder containing 418 blocks. It is extremely unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter.

The **rp** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RP files begin with **rrp** and end with a number which selects the same disk section as the corresponding **rp** file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/rp*, /dev/rrp*

SEE ALSO

gd(7), hm(7), hp(7), rm80(7), rp07(7).

NAME

hs — RH11/RJS03-RJS04 fixed-head disk file

DESCRIPTION

The files **hs0**, ..., **hs7** refer to RJS03 disk drives 0 through 7. The files **hs8**, ..., **hs15** refer to RJS04 disk drives 0 through 7. The RJS03 drives are each 1024 blocks long and the RJS04 drives are 2048 blocks long.

The **hs** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw HS files begin with **rhs**. The same minor device considerations hold for the raw interface as for the normal interface.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/hs*, /dev/rhs*

7

NAME

ht — TU16/TE16 magnetic tape interface

DESCRIPTION

The files **mt0**, ..., **mt15** refer to the Digital Equipment Corporation TU16 magnetic tape control and transports. The files **mt0**, ..., **mt7** are 800bpi, and the files **mt8**, ..., **mt15** are 1600bpi. The files **mt0**, ..., **mt3**, **mt8**, ..., **mt11** are designated normal-rewind on close, and the files **mt4**, ..., **mt7**, **mt12**, ..., **mt15** are no-rewind on close. When opened for reading or writing, the tape is assumed to be positioned as desired. When a file is closed, a double end-of-file (double tape mark) is written if the file was opened for writing. If the file was normal-rewind, the tape is rewound. If it is no-rewind and the file was open for writing, the tape is positioned before the second EOF just written. If the file was no-rewind and opened read-only, the tape is positioned after the EOF following the data just read. Once opened, reading is restricted to between the position when opened and the next EOF or the last write. The EOF is returned as a zero-length read. By judiciously choosing **mt** files, it is possible to read and write multi-file tapes.

A standard tape consists of several 512 byte records terminated by an EOF. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time (although very inadvisable).

The **mt** files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the "raw" interface is appropriate. The associated files are named **rmt0**, ..., **rmt15**. Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

FILES

/dev/mt*, /dev/rmt*

BUGS

If any non-data error is encountered, it refuses to do anything more until closed. The driver is limited to four transports.

SEE ALSO

gt(7), tu78(7).

7

NAME

kl — KL-11 or DL-11 asynchronous interface

DESCRIPTION

The discussion of typewriter I/O given in *tty*(7) applies to these devices.

Since they run at a constant speed, attempts to change the speed are ignored.

The on-line console typewriter is normally interfaced using a KL-11 or DL-11.

FILES

/dev/console

SEE ALSO

init(1M), tty(7).

BUGS

Modem control for the DL-11E is not implemented.

NAME

kmc — KMC-11B/KMS11 microprocessor

DESCRIPTION

The files **kmc?** are used to manipulate the KMC11-B or KMS11 microprocessor. The device handler provides the basic mechanism needed to load, run, and debug programs on the microprocessor. The open is exclusive; at most one open at a time.

Addresses 0−8195 reference the 4096 words of instructions in the control memory of the microprocessor. This portion is word oriented, that is, the address and byte count must be even.

Addresses 8196−12211 reference the 4096 bytes of data in the data memory of the microprocessor. The data portion may be read or written with no restrictions on addressing.

The *ioctl*(2) function is used to provide access to the basic microprocessor capabilities.

```
#include <sys/kmc.h>
ioctl (fildes, command, arg)
struct kmcntl {
        int     kmd;
        short   *kcsr;
        int     kval;
} *arg;
```

The only *command* available is KCSETA. The pointer *kcsr* contains the address of a 4 word buffer for the UNIBUS Control and Status Registers associated with the microprocessor. The value of *kmd* determines the function:

1	single step and return CSRs in *kcsr*.
2	maintenance step: execute *value* and then return CSRs.
3	return CSRs.
4	stop: clear the run bit.
5	reset: set then clear the master clear bit.
6	run: set the run bit and set the software state to *kval* and running.
7	line unit maintenance: set the line unit bits from *kval*.
8	set CSR sel6 to *kval*.
9	clear: first reset, then empty the input queue.

FILES

/dev/kmc?

SEE ALSO

kasb(1), dz(7), vpm(7).

NAME

lp — line printer

DESCRIPTION

Lp provides the interface to any of the standard Digital Equipment Corporation LP-11 UNIBUS line printers. When it is opened or closed, a suitable number of page ejects is generated. Bytes written are printed.

An internal parameter within the driver determines whether or not the device is treated as having a 96- or 64-character set. In half-ASCII mode, lower case letters are turned into upper case and certain characters are escaped according to the following table:

{	-(
}	-)
\`	-'
|	-!
~	-^

The driver correctly interprets carriage returns, backspaces, tabs, and form-feeds. A new-line that extends over the end of a page is turned into a form-feed. The default line length is 132 characters, indent is 4 characters and lines per page is 66. Lines longer than the line length minus the indent (i.e. 128 characters, using the above defaults) are truncated.

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

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

The *commands* are:

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

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

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

FILES

/dev/lp

SEE ALSO

lpr(1).

NAME

mem, kmem — core memory

DESCRIPTION

Mem is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system.

Byte addresses in *mem* are interpreted as memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file *kmem* is the same as *mem* except that kernel virtual memory rather than physical memory is accessed.

On the PDP-11, the I/O page begins at location 0160000 of *kmem* and per-process data for the current process begins at 0140000.

FILES

/dev/mem, /dev/kmem

BUGS

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

7

NAME

ml11 — ML11 solid-state disk

DESCRIPTION

The ML11 is a solid state disk manufactured by Digital Equipment Corporation. It has a capacity of one to four megabytes and a transfer rate of 0.25 to 2.00 megabytes per second. It is supported as part of the general disk driver (see *gd*(7)). The device is not usable as either a boot device or an initial load device. It is intended for use on the PDP-11 line for faster **/dev/swap**, **/bin** or **/tmp** operations.

FILES

/dev/rp*, /dev/rrp*, /dev/swap

SEE ALSO

gd(7), hm(7), hp(7), rp(7), rp07(7), rm80(7).

NAME

nc — network control

DESCRIPTION

The network control pseudo-device provides a means by which a privileged user process can install, remove, and get the status of a BX.25 Permanent Virtual Circuit (PVC), and attach, detach, start, stop, get the status of, and perform a changeover on a BX.25 link. Additional functions are planned for this driver when the virtual-call feature and additional layers of BX.25 are added to the UNIX BX.25 implementation. The BX.25 driver supports *open*(2), *close*(2), and *ioctl*(2) system calls. Definitions of constants and declarations for the data structures mentioned can be obtained by:

#include <sys/nc.h>

The network-control *ioctl* system call has the following form:

ioctl (fildes, cmd, arg)

where *fildes* is the file descriptor returned by the *open* of the **nc** device and *cmd* is one of the following constants:

NCPVCI — Install a PVC. This command creates one end of a PVC by connecting a minor device of the BX.25 driver (slot) to a particular logical channel on a specified link. *Arg* is a pointer to a *ncpvc* data structure where *slot* is the minor device number of the slot to be used as the end point of the PVC, *chno* is the logical channel number to be used, and *link* is the number of the BX.25 link to be used. Links are numbered starting with 0. *Chno* must be in the range 1 to 4,095 and must not be currently in use on this link. The following errors may be returned: ENXIO if the minor device number is out of range; ECHRNG if the channel number is out of range; ELNRNG if the link number is out of range; EBUSY if the slot is in use. The two low-order bits of *options* specify one of three possible session-establishment protocols:

PVC_SESS	session-layer open/close protocol
PVC_RST	reset in-order/out-of-order protocol
PVC_NONE	"no-protocol" session mode

These protocols are explained in *x25*(7). The constants PVC_SESS, PVC_RST, and PVC_NONE are defined in **/usr/include/sys/x25u.h**. If the link on which the PVC is installed is currently active (i.e., not in the halted state), the BX.25 reset procedure will be initiated for the logical channel. When the reset procedure is completed, the PVC is ready for data transfer.

NCPVCR — Remove a PVC. If *arg* is the minor device number of a slot that is currently associated with a PVC and is not open, the local end of that PVC is removed, i.e., disconnected. The slot and logical channel number become available for reuse. The following errors may be returned: ENXIO if the minor device number is out of range; ENODEV if the minor device is not installed; EBUSY if the slot is in use.

NCPVCSTAT — Get the status of a PVC. This command gets the connection and status information for slot *slot* and places it in the *ncpvcstat* data structure pointed to by *arg*. The following errors may be returned: ENXIO if the minor device number is out of range; ENODEV if the minor device is not installed.

NCATTACH, NCBKATTACH — Associate a link with a specified communications device and mark the device as primary (NCATTACH) or backup (NCBKATTACH). *Arg* is a pointer to an *ncattch* data structure that contains the link number and major/minor device number of the device. The following errors may be returned: ELNRNG if the link number is out of range;

EACCES if the link is already attached; EBUSY if the device is already attached; ENOCSI if no CSI structure is available; ENXIO if the minor device number of the communications device is out of range.

NCDETACH, NCBKDETACH — Disassociate the link specified by *arg* from its associated primary (NCDETACH) or backup (NCBKDETACH) device. The link and device become available for reuse. The following errors may be returned: ELNRNG if the link number is out of range; EUNATCH if the link is not attached to a device; ENODEV if the device is not attached; EBUSY if the device has not been halted.

NCSTART, NCBKSTART — Start a specified link. The level 2 protocol is started on the primary (NCSTART) or backup (NCBKSTART) device associated with the link specified. *Arg* is a pointer to an *ncstart* data structure that contains information such as the link number and packet size. The level 3 restart procedure is then initiated for the link if **NCSTART** is the command used. The following errors may be returned: ELNRNG if the link number is out of range; EUNATCH if the link is not attached to a device; EBUSY if the device is already started; EINVAL if the packet size specified with NCSTART is different from that specified with **NCBKSTART**.

NCSTOP, NCBKSTOP — Stop a BX.25 link. The **NCSTOP** command stops the level 3 protocol on the link specified by *arg* and the link data structure is reinitialized. For either command, the level 2 protocol is stopped on the associated primary or backup device. For NCSTOP, if a backup device has been attached, the level 2 protocol is also stopped on that device. The following errors may be returned: ELNRNG if the link number is out of range; EUNATCH if the link is not attached to a device; EBUSY if the level 3 protocol is running on the backup device (NCSTOP).

7

NCCHNGE — Changeover to the standby device associated with the link specified by *arg*. If the standby device is synchronized at level 2, the level 3 protocol will now run on that device. The following errors may be returned: ELNRNG if the link number is out of range; EUNATCH if the link is not attached to both a primary and backup device; EACCES if the backup device was not started.

NCLNKSTAT — Get the status of a link. This command gets the connections and status information for link *link* and places it in the *nclnkstat* data structure pointed to by *arg*. The following errors may be returned: ELNRNG if the link number is out of range; EUNATCH if the link is not attached to either a primary or backup device.

SEE ALSO

x25pvc(1M), x25(7).

NAME

nsc — NSC adapter interface specification

DESCRIPTION

The special files **nsc0, ..., nscn** refer to the control of a Network Systems Corporation (NSC) A-410 processor adapter. Each special file multiplexes across the transmission medium the full-duplex network operations of twenty (20) simultaneous opens.

Physical NSC network transmissions occur in two parts and in the following order: a 64-byte message block and an n-byte associated data block. The 64-byte message contains network control and routing information. The network message has the following structure:

```
struct nmsg
{
        char nm_adata;       /* associated data flag */
        char nm_trunk;       /* trunk selection */
        char nm_acode;       /* access code */
        char nm_vchan;       /* virtual channel */
        short nm_tonad;      /* "to" network address */
        short nm_frnad;      /* "from" network address */
        char nm_fnc;         /* protocol function */
        char nm_opcod;       /* adapter operation code */
        char nm_data[54];    /* control info */
};
```

The associated data block transfers large, variable-length data blocks. The NSC driver currently limits the associated data block size to 4096 bytes.

The driver issues the proper function code sequences to the A-410 adapter. The available function codes are defined as follows:

```
ATM      0005   /* Transmit Message */
ATD      0010   /* Transmit Data */
ATLSTD   0014   /* Transmit Last Data */
ATLM     0021   /* Transmit Local Message */
AIM      0045   /* Input Message */
AID      0050   /* Input Data   */
ASTAT    0101   /* Status       */
AMDP0    0140   /* Mark Down Port 0 */
AMDP1    0144   /* Mark Down Port 1 */
AMDP2    0150   /* Mark Down Port 2 */
AMDP3    0154   /* Mark Down Port 3 */
AMDR0    0160   /* Mark Down Port 0 & Reroute Msgs */
AMDR1    0164   /* Mark Down Port 1 & Reroute Msgs */
AMDR2    0170   /* Mark Down Port 2 & Reroute Msgs */
AMDR3    0174   /* Mark Down Port 3 & Reroute Msgs */
ARST     0241   /* Read Statistics */
ARCST    0245   /* Read & Clear Statistics */
ASTST    0300   /* Set Test     */
ASAL     0305   /* Set Address & Length */
AWA      0310   /* Write Adapter */
ARA      0314   /* Read Adapter        */
ACA      0340   /* Clear Adapter */
AEOP     0344   /* End Operation */
ACLWM    0346   /* Clear Wait For Message State */
AWAITM   0350   /* Wait Message        */
```

The driver always saves the network status bytes on failed or aborted transfer attempts. The user may retrieve the eight bytes of adapter status and perform the appropriate error recovery procedures. The eight adapter status bytes are defined by the following structure:

```
struct adptrst
{
      char st_afc;        /* last function code */
      char st_gsw;        /* general status word */
      char st_trkst;      /* trunk status */
      char st_trkrsp;     /* trunk response */
      char st_err;        /* adapter error code */
      char st_internal;   /* reserved for adapter use */
      char st_rtnrsp;     /* remote returned response */
      char st_spare[3];
};
```

After successful *open*(2) completion and before reading and writing to the network, the user must establish a virtual channel. Both the local and the remote machines must agree on this virtual channel to properly transfer data. A virtual channel is defined to be a destination network address and a virtual channel number. The driver enforces mutually exclusive virtual channels to properly route incoming network transmissions. There are currently 256 virtual channel numbers (0-255) supported. If the user specifies a zero destination address in the virtual channel, that process will receive the incoming transmissions from all remote nodes on the specified virtual channel number. If, however, another process establishes a virtual connection with the same virtual channel number but with a specific (non-zero) destination address, the specific connection will preempt the non-specific (zero destination address) connection and receive all incoming message transmissions from the particular remote node.

7

The NSC driver supports two modes of transfer: *data* and *control. Data* mode is the default mode. After virtual channel configuration, the user process performs simple reads and writes. The process does not need to know that it is transmitting across the NSC bus. Placement of the data into the message, the associated data block, or both is completely transparent. The user may optionally specify, however, that the first n-bytes of the transfer buffer be always placed into the NSC message ($0 <= n <= 52$). This may be particularly useful when transferring combinations of binary (i.e protocol headers) and character data (i.e. files) between heterogeneous machines (see the NSCIOASMB NSC *ioctl*(2) command). Reads and writes in data mode return the total number of user bytes transferred.

In *control* mode, the user has direct access to the NSC control information within the NSC message. The user process specifies to the driver two pointers to static buffer areas, one for reads and one for writes. For write operations, the user builds a modified version of the NSC message in the write static buffer area. When the *write*(2) system call is made, the driver retrieves the data from this buffer to build the outgoing NSC message. The structure for the write static buffer area is:

```
   struct nsctrl {
        short cn_flags;     /* associated data flag */
        short cn_tonad;     /* destination address */
        char  cn_fnc;       /* adapter function code */
        char  cn_opcode;    /* adapter operation code */
        char  cn_data[54];  /* unused data area */
```

If the user sets the associated data flag in **cn_flags** (NSCADATA), the buffer pointer in the *write*(2) call is sent in the associated data block. Otherwise, the driver transfers a message alone.

On *control* mode read operations, the driver places the entire 64-byte NSC message into the read static buffer area. If the message has an associated data block, the received data is placed into the buffer area specified in the *read*(2) system call.

For both *control* mode reads and writes, the NSC driver returns the number of bytes transferred in the associated data block. The driver returns a count of **one** (1) if a message alone was transmitted or received.

User processes configure parameters into the driver through *ioctl*(2). The driver recognizes the following *ioctl* requests:

NSCIODATA (struct datam *) argp

NSCIODATA places the virtual connection into *data* mode (the default mode for successful opens). The user specifies the number of data bytes always contained in the NSC message on reads and writes. *Argp* is a pointer to the following structure:

```
struct datam {
      short i_mbytes;   /* bytes in msg on reads */
      short o_mbytes;   /* bytes in msg on write */
};
```

NSCIOCTRL (struct ctrlm *) argp

NSCIOCTRL places the virtual connection into *control* mode. *Argp* is a pointer to the following structure:

```
struct ctrlm {
      struct nmsg   *i_mptr;   /* read static area */
      struct nsctrl *o_mptr;   /* write static area */
};
```

NSCIOVCHAN (struct nscvchan *) argp

NSCIOVCHAN configures the virtual channel for the specified open. *Argp* is a pointer to the following structure:

```
struct nscvchan {
      short v_tonad;  /* destination network addr */
      char  v_vchan;  /* virtual channel number */
      char  v_tmsk;   /* trunk mask */
      char  v_acode;  /* access code */
};
```

V_tonad and *v_vchan* configure the virtual channel for all subsequent reads and writes to the network. *V_tmsk* sets the trunk transfer mask for network transfers. For trunk transfers, each bit in the trunk mask is cleared in the trunk specification of the NSC network message *(nm_trunk)*. *V_acode* is the hardware access code place into the outgoing NSC network message *(nm_acode)*.

NSCIOESTAT (struct adptrst *) argp

NSCIOESTAT retrieves into the user buffer specified by *argp* the 8 bytes of adapter status from the last failed network operation. After the status bytes are retrieved, the buffer area in the driver is cleared.

NSCIOGETP (struct nscgetty *) argp

NSCIOGETP retrieves network connection parameters from the driver.

Argp is a pointer to the following structure:

```
struct nscgetty {
        short g_flags;    /* connection flags */
        char  g_openm;  /* file open mode */
        char  g_vchan;  /* virtual channel number */
        short g_taddr;   /* destination network addr */
        char  g_acode;  /* access code */
        char  g_tmsk;   /* trunk mask */
        struct nscasmb g_asm;  /* assembly modes */
};
```

NSCIOASMB (struct nscasmb *) argp
NSCIOASMB selects the assembly/disassembly modes for network message and associated data block transmission and reception. Assembly modes are not necessary unless data is transferred between heterogeneous processors. Assembly mode 0 causes the hardware to swap incoming and outgoing bytes. This one is used primarily when transferring character data. Assembly mode 1 causes no swapping. This one is used primarily when transferring binary data. Assembly mode 1 is defaulted for the NSC message; assembly mode 0 is defaulted for the associated data block. *Argp* is a pointer to the following structure:

```
struct nscasmb {
        unsigned i_msg : 2;     /* input msg */
        unsigned i_data : 2;    /* input data */
        unsigned o_msg : 2;     /* output msg */
        unsigned o_data : 2;    /* output data */
};
```

7

NSCIOADDR (short *) argp
NSCIOADDR returns to the calling process the network address of the local node. The local address is generated from data retrieved from the adapter.

NSCIOBYE (char *) 0
NSCIOBYE disconnects the user process from the driver. This function performs the necessary cleanup to ensure proper driver operation.

NSCIOFCODE (struct nscfcode *) argp
NSCIOFCODE allows the user to issue any function to the adapter. The super-user is allowed to issue any function; others may issue only the status function (ASTAT). *Argp* is a pointer to the following structure:

```
struct nscfcode {
        char  *f_base;  /* buffer area */
        short f_cnt;      /* no. of bytes to xfer */
        short f_fcode;  /* func. code to issue */
};
```

NSCIOCANCEL (char *) 0
NSCIOCANCEL is a super-user only function. This command cancels the currently active adapter operation and returns an error to the effected user process. This command is used to clear hung processes.

NSCIOOFFLINE (char *) 0
NSCIOOFFLINE is a super-user only function. This command inhibits via software all function code issuance. Opens will occur normally,

but all reads and writes will block. There is one exception: the super-user (by an NSCIOFCODE command) may issue any function.

NSCIOONLINE (char *) 0
NSCIOONLINE is a super-user only function. This command enables via software all function code issuance. This command is the converse of NSCIOOFFLINE.

FILES

/usr/src/cmd/nusend/nscdef.h
/usr/include/sys/nsc.h

SEE ALSO

nusend(1C), ioctl(2), read(2), write(2).

DIAGNOSTICS

Read(2) and *write*(2) both return the number of bytes successfully transferred. A −1 is returned on error.

BUGS

An error return does not necessarily mean that the network is down. Whenever an error occurs, adapter status should be retrieved from the driver. Most failed operations should be retried several times before giving up.

In *control* mode, 1 is returned if a message alone is transmitted or received.

7

NAME

null — the null file

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

7

NAME

osm — interface to UNIX system messages

DESCRIPTION

Operating system messages are stored in a circular buffer in the system and can be read or written using the special files **/dev/osm***. A read from the file **/dev/osm*** will return some portion of the data in the circular buffer. A write to the file **/dev/osm*** will add the user data to the current end of the circular buffer. Any number of users can use the *osm* interface at once.

In particular: Reads from the file **/dev/osm** start at the current end of the circular buffer and wait for new data to be added. Reads from the file **/dev/osm.cur** start at the begining of the circular buffer and return zero bytes when the current end of the circular buffer is reached. Reads from the file **/dev/osm.all** start at the begining of the circular buffer, go to the current end of the circular buffer, and then wait for new data to be added.

The easiest way to use the *osm* interface is by typing:

cat −u /dev/osm &

or by typing:

echo *message* > /dev/osm

FILES

/dev/osm*

7

NAME

pcl — parallel communications link interface

DESCRIPTION

Pcl provides the interface for up to two Digital Equipment Corporation PCL-11B network buses. Each bus can be used to interconnect up to 16 CPU's, providing relatively fast communication without individual point-to-point connections.

The interface permits simultaneous bi-directional communication between any machines on a single bus. Additionally, each such path is further subdivided into 8 independent channels. A single control interface is provided to reduce the line monitoring overhead for a daemon process.

The minor device number for a PCL channel is constructed as follows:

the low order 3 bits	specify a channel number.
the next 4 bits	specify one of 16 machines. (This number must be one less than the PCL Time Division Multiplexed bus number set in the hardware.)
the next bit	specifies one of 2 PCL's.

FILES

/dev/pcl/?[0-7]	normal machine and subchannel interface.
/dev/pcl/ctrl	control interface.

SEE ALSO

net(1C), pcldaemon(1M).

7

NAME

prf — operating system profiler

DESCRIPTION

The file **prf** provides access to activity information in the operating system. Writing the file loads the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file **prf** is a pseudo-device with no associated hardware.

FILES

/dev/prf

SEE ALSO

config(1M), profiler(1M).

NAME

rf — RF11/RS11 fixed-head disk file

DESCRIPTION

This file refers to the concatenation of all RS-11 disks.

Each disk contains 1024 256-word blocks. The length of the combined RF file is 1024×(minor+1) blocks. That is minor device zero is taken to be 1024 blocks long; minor device one is 2048, etc.

The **rf0** file accesses the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The name of the raw RF file is **rrf0**. The same minor device considerations hold for the raw interface as for the normal interface.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/rf0, /dev/rrf0

BUGS

The 512-byte restrictions on the raw device are not physically necessary, but are still imposed.

NAME

rk — RK-11/RK03 or RK05 disk

DESCRIPTION

The file **rk?** refers to an entire RK03 disk as a single sequentially-addressed file. Its 256-word blocks are numbered 0 to 4871.

The **rk** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RK files begin with **rrk** and end with a number which selects the same disk as the corresponding **rk** file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/rk*, /dev/rrk*

NAME

rl − RL-11/RL01 disk

DESCRIPTION

rl0, ..., **rl3** refer to an entire RL01 disk drive as a single sequentially-addressed file. Its 256-word blocks are numbered 0 to 10239.

The **rl** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O call and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RL files begin with **rrl** and end with a number which selects the same disk as the corresponding **rl** file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/rl*, /dev/rrl*

7

NAME

rm80 — RM80 moving-head disk

DESCRIPTION

The files **rp0, ..., rp7** refer to sections of the RM80 disk drive 0. The files **rp10, ..., rp17** refer to drive 1, etc. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	0	18228
1	42	224378
2	194	158410
3	346	92442
4	498	26474
5	—	—
6	—	—
7	0	242606

The start address is a cylinder address, with each cylinder containing 434 blocks. It is extremely unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter.

The **rp** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RM files begin with **rrp** and end with a number which selects the same disk section as the corresponding **rp** file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/rp*, /dev/rrp*

SEE ALSO

gd(7), hm(7), hp(7), rp(7), rp07(7).

NAME

rp — RP-11/RP03 moving-head disk

DESCRIPTION

The files **rp0**, ..., **rp7** refer to sections of the RP03 disk drive 0. The files **rp10**, ..., **rp17** refer to drive 1, etc. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	0	10000
1	50	71200
2	203	40600
3	—	—
4	—	—
5	—	—
6	—	—
7	0	81200

The start address is a cylinder address, with each cylinder containing 200 blocks. It is extremely unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter.

The **rp** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RP files begin with **rrp** and end with a number which selects the same disk section as the corresponding **rp** file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/rp*, /dev/rrp*

SEE ALSO

hp(7).

NAME

rp07 — RP07 non-removable medium moving-head disk

DESCRIPTION

The files **rp0, ..., rp7** refer to sections of the RP07 disk drive 0. The files **rp10, ..., rp17** refer to drive 1, etc. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	0	64000
1	40	944000
2	105	840000
3	210	672000
4	315	504000
5	420	336000
6	525	168000
7	0	1008000

The start address is a cylinder address, with each cylinder containing 1600 blocks. It is extremely unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter.

The **rp** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RP files begin with **rrp** and end with a number which selects the same disk section as the corresponding **rp** file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/rp*, /dev/rrp*

SEE ALSO

gd(7), hm(7), hp(7), rp(7), rm80(7).

7

NAME

st — synchronous terminal interface

DESCRIPTION

The synchronous terminal interface is a pseudo-device driver that enables a UNIX system to communicate with TELETYPE® Model 4540 compatible ASCII synchronous terminals. The driver utilizes the bottom half of the Virtual Protocol Machine (VPM) to perform the end-to-end protocol and transmission assurance for the synchronous line.

There are two modes of operation for synchronous terminals; application mode and line mode. In application mode, the user must be familiar with the operation of the Model 4540 terminal. Screen management functions are completely controlled by the user process; when formating a screen, the user must supply everything from the initial STX (Start-of-Text) character to the ETX (End-of-Text) character.

In line mode, the basic screen management functions are handled by the driver to make the synchronous terminal usable as a login terminal for most of the standard UNIX commands. (Commands that put the terminal in raw mode or write any control characters to the screen will probably not work as expected.) Writes to the terminal will be packaged in the necessary protocol so that only terminal operator input will be returned to the user process on a *read*(2). See *stty*(1) for details on setting these modes and other available options.

By convention, **/dev/st?** is the synchronous terminal control channel for communications line ?. Communication with the control channel is handled by the *stcntrl* and *stprint* commands (see *st*(1M)).

7

A user process will sleep when trying to open a terminal channel, until a terminal requests service. At that time, a channel will be assigned to that terminal, and it will remain allocated until the user process closes the terminal.

A user process will not sleep when trying to open a printer channel. Printer channel connections are established by *stprint* and remain in effect until the associated communications line drops.

In addition to the synchronous terminal equipment, appropriate synchronous VPM hardware is required.

FILES

/lib/stscr	synchronous terminal prototype script
/dev/un53.?	TN82/UN53 peripheral controller pair (3B20S only)
/dev/kmc?	KMC11-B microprocessor (DEC only)
/dev/vpb?	VPM bottom half (DEC only)
/dev/st?	synchronous terminal control channels
/dev/tty*	synchronous terminal user channels
/dev/sp*	synchronous printer user channels

In addition to the standard *ioctl* functions listed in *stermio*(7), the commands defined in **/usr/include/sys/stermio.h** are provided with the following interfaces:

```
ioctl(stcontrolfd, STPRINT, device)
char    device;
```

tells the driver that a printer is at the device address specified by *device* on the synchronous communications line associated with *stcontrolfd*. The return value is the minor device number associated with the printer. A −1 is returned if the association can't be made. (Too many printers are already associated or the communications line is not connected.)

ioctl(stcontrolfd, VPMSDEV, arg);

will assign VPM minor device number *arg* to the line associated with *stcontrolfd*.

ioctl(stcontrolfd, STSTART);

tells the driver to start up the line associated with *stcontrolfd*. If this is the first line started, buffer space will be allocated from physical memory for use by all lines.

ioctl(stcontrolfd, STHALT);

tells the driver to stop the line associated with *stcontrolfd*. If this is the last active line, the buffer space allocated on the first STSTART will be returned to the system.

ioctl(stfd, STWLINE);

returns the synchronous communications line number associated with the terminal, printer, or control channel file descriptor *stfd*.

SEE ALSO

st(1M), kmc(7), stermio(7), trace(7), un53(7), vpm(7).

7

NAME

stermio — general synchronous terminal interface

DESCRIPTION

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

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

A terminal associated with one of these files operates in half-duplex mode. Characters may be typed only when the terminal is in local mode.

When the user channel is in line mode, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character that is supplied by the driver at the end of each field from the terminal. No matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

When the user channel is in application mode, full blocks of data from the terminal may be requested in a read. As in line mode, any number may be requested without losing information. The program must know how to interpret the protocol and field separation characters to understand the data returned.

Certain characters have special functions on input when the user channel is in line mode. These functions and their default key assignments are summarized as follows:

INTR (PA1) generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).

QUIT (PA2) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.

EOF (PF12) may be used to generate an end-of-file from a terminal. When received, all characters changed in the terminal's buffer are queued to be passed to the program and the EOF is discarded. Thus, if there are no changes in the terminal's buffer, zero characters will be passed back, which is the standard end-of-file indication.

CTAB (@) will be translated to an ASCII HT character.

SEND (S/R, PF1 through PF11) may be used to send modified fields to the program. In line mode, each modified field is packaged as an input line with a trailing new-line character. In application mode, the entire transmission block is given to the program.

CLEAR (CLEAR) clears the screen. In line mode, the screen will be reformatted so that each line is a separate field. In application mode, reformatting the screen is the program's responsibility.

The character value for CTAB may be changed to suit individual tastes.

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

Several *ioctl*(2) system calls apply to synchronous terminal files. Several of these calls use the same structure defined in **/usr/include/sys/termio.h** as described in *termio*(7) and accept the same TCGETA, TCSETA, TCSETAW, and TCSETAF commands that are described there. When these calls are used, however, only the *c_iflag*, *c_oflag*, and *c_lflag* fields are used with these fields corresponding, respectively to the *imode*, *omode*, and *lmode* fields described in the following description of the *stermio* structure that is defined in **/usr/include/sys/stermio.h**. Within those fields, only the values described below, some of which overlap those described in *termio*(7), are used.

```
struct stermio {
        unsigned short  ttyid;    /* station and device id's */
        char            row;      /* cursor row position at last SEND */
        char            col;      /* cursor col position at last SEND */
        char            orow;     /* next output cursor row position */
        char            ocol;     /* next output cursor col position */
        char            tab;      /* translate to tab on input */
        char            aid;      /* function key identification code */
        char            ss1;      /* status and sense character 1 */
        char            ss2;      /* status and sense character 2 */
        unsigned short  imode;    /* input modes */
        unsigned short  lmode;    /* local modes */
        unsigned short  omode;    /* output modes */
};
```

The *ttyid* field contains the station selection character in the high order byte and the device selection character in the low order byte.

The *row* and *col* fields contain the row and column numbers of the screen position of the cursor when the last SEND key was hit. Rows are numbered from 1 through 24. Columns are numbered from 1 through 80.

The *orow* and *ocol* fields specify the next screen position that will be written.

The *tab* field contains the character that will be translated to an ASCII TAB character on input if line mode is enabled.

The *aid* field contains the function key identification code signifying the terminal key that caused the last buffer to be sent.

The *ss1* and *ss2* fields contain the last status and sense characters received from the terminal.

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

IUCLC	0001000	Map upper-case to lower-case on input.

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

The initial input control value is all bits clear.

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

OLCUC	0000002	Map lower case to upper on output.
TABDLY	0014000	Select horizontal-tab translation option:
TAB0	0	Don't modify tabs.
TAB3	0014000	Expand tabs to spaces.

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

Horizontal-tab type 0 specifies that tabs are not to be modified. Type 3 specifies that tabs are to be expanded into spaces.

The initial output control value is TAB3.

The *lmode* field of the argument structure is used by the line discipline to control terminal functions. The synchronous terminal line discipline provides the following:

XCASE	0000004	Canonical upper/lower presentation.
STFLUSH	0000400	Flush output on each *write*(2).
STWRAP	0001000	Wrap around long lines.
STAPPL	0002000	Use application mode.

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

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

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

If STAPPL is set, application mode is enabled. Read requests are satisfied directly from the terminal input buffer, and the user is responsible for handling all terminal protocol from the STX character through the ETX character on output.

If STAPPL is not set, line mode processing is enabled. This enables the input fields from the terminal to be broken into lines terminated with a new-line chracter and the actions to provide the IUCLC, TAB3, OLCUC, XCASE, STFLUSH, and STWRAP processing to be performed. For output, the screen is formatted so that each terminal line is a separate field. New-line characters cause the remainder of the current line to be cleared and the cursor to be positioned at the beginning of the next field. If data overflows the last line of the terminal, the cursor is repositioned to the beginning of the first field on the screen and output is halted until one of the SEND keys, the PF12 key, or the CLEAR key is hit to restart output. This allows the terminal operator to read a screen full of data before it is overwritten.

The initial value for terminal modes has the STAPPL and STWRAP modes enabled.

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

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

The commands using this form are:

STGET Get the parameters associated with the terminal and store in the *stermio* structure referenced by **arg**.

STSET Set the parameters associated with the terminal from the structure referenced by **arg**. Only the *imode*, *lmode*, *ocol*, *omode*, *orow*, and *tab* fields are affected. The change is immediate. A switch from application mode to line mode will cause the screen to be reformatted by the driver.

FILES

/dev/tty*

SEE ALSO

stty(1), ioctl(2), st(7),termio(7).

NAME

termio — general terminal interface

DESCRIPTION

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

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

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, the character # erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character @ kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The erase and kill characters may be changed.

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

INTR (Rubout or ASCII DEL) generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).

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

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

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

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

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

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

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

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

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

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

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

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

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

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

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

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

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

7

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

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

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

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

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

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

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

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

The initial input control value is all bits clear.

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

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

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

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

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

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

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

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

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

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

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

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

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

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
EXTA	0000016	External A
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.

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

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

7

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

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

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

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

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

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

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

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

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

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

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

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

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

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

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

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

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

The commands using this form are:

- TCGETA Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.
- TCSETA Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.
- TCSETAW Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.
- TCSETAF Wait for the output to drain, then flush the input queue and set the new parameters.

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

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

The commands using this form are:

- TCSBRK Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).
- TCXONC Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.
- TCFLSH If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

FILES

/dev/tty*

SEE ALSO

stty(1), ioctl(2).

NAME

tm — TM11/TU10 magnetic tape interface

DESCRIPTION

The files **mt0**, ..., **mt7** refer to the Digital Equipment Corporation TM11/TU10 magnetic tape control and transports at 800bpi. The files **mt0**, ..., **mt3** are designated normal-rewind on close, and the files **mt4**, ..., **mt7** are no-rewind on close. When opened for reading or writing, the tape is assumed to be positioned as desired. When a file is closed, a double end-of-file (double tape mark) is written if the file was opened for writing. If the file was normal-rewind, the tape is rewound. If it is no-rewind and the file was open for writing, the tape is positioned before the second EOF just written. If the file was no-rewind and opened read-only, the tape is positioned after the EOF following the data just read. Once opened, reading is restricted to between the position when opened and the next EOF or the last write. The EOF is returned as a zero-length read. By judiciously choosing *mt* files, it is possible to read and write multi-file tapes.

A standard tape consists of several 512 byte records terminated by an EOF. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time (although very inadvisable).

The **mt** files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the "raw" interface is appropriate. The associated files are named **rmt0**, ..., **rmt7** Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

FILES

/dev/mt?, /dev/rmt?

BUGS

If any non-data error is encountered, it refuses to do anything more until closed. The driver is limited to four transports.

NAME
tn4 — eight line asynchronous interface

DESCRIPTION
Each of the eight lines attached to a TN4 behaves as described in *termio*(7). The *c_cflag* items of **B200**, **EXTA**, and **EXTB** are not available.

FILES
/dev/tty*

SEE ALSO
termio(7), tn74(7).

NAME

tn74 − two line asynchronous interface

DESCRIPTION

Each of the two lines attached to a TN74 behaves as described in *termio*(7). The *c_cflag* items of **B200**, **EXTA**, and **EXTB** are not available.

FILES

/dev/tty*

SEE ALSO

termio(7), tn4(7).

NAME

tn83 — console/printer interface

DESCRIPTION

The TN83 is a specialized controller that provides the operator interface to the 3B20S. It supports the Emergency Action Interface. See *eai*(8).

The files **/dev/console** and **/dev/rop** refer to the system console and the receive-only printer. These special files implement a subset of those features described in *termio*(7). Among the differences are:

> Hardware options such as line speed are not selectable. The console runs at 9600 baud while the receive-only printer at 1200 baud.
>
> The ICANON option (raw mode) does not work.
>
> The START/STOP (control-s/control-q) characters only have a temporary affect. Use control-x/control-z instead.

Data read and/or written to/from **/dev/console** is automatically written to the receive-only printer. The command line **stty −echo < /dev/rop** will turn off this feature while **stty echo < /dev/rop** will turn it on.

System messages are normally printed to the console and the receive-only printer. These messages may be turned on or off by typing a control-o at the console.

FILES

/dev/console, /dev/rop

SEE ALSO

termio(7), eai(8).

7

NAME

tn85 — medium speed line printer controller

DESCRIPTION

The TN85 provides a parallel interface to either one or two medium speed line printers which can operate at up to 2000 lines per minute (132 columns per line, 96 character ASCII set). If two printers are connected to a TN85, then the combined throughput cannot exceed 2000 lines per minute total. For example, it can handle two 1000 line per minute printers or one 2000 line per minute printer.

FILES

/dev/lp*

7

NAME

trace — event-tracing driver

DESCRIPTION

Trace is a special file that allows event records generated within the UNIX kernel to be passed to a user program so that the activity of a driver or other system routines can be monitored for debugging purposes.

An event record is generated from within a kernel driver or system routine by invoking the *trsave* function:

```
trsave (dev, chno, buf, cnt)
char    dev, chno, *buf, cnt;
```

Dev is a minor device number of the trace driver; *chno* is an integer between 0 and 15 inclusive that identifies the data stream (channel) to which the record belongs; *buf* is a buffer containing the data for an event; and *cnt* is the number of bytes in *buf*. Calls to *trsave* will result in data being placed on a queue, provided that some user program has opened the trace minor device *dev* and has enabled channel *chno*. Event records consisting of a time stamp (4 bytes), the channel number (1 byte), the count (1 byte), and the event data (*cnt* bytes) are stored on a queue until a system-defined maximum (TRQMAX) is reached; an event record is discarded if there is not sufficient room on the queue for the entire record. The queue is emptied by a user program reading the trace driver. Each *read* returns an integral number of event records; the read count must, therefore, be at least equal to *cnt* plus six.

The *trace* driver supports *open, close, read,* and *ioctl* system calls. The *ioctl* system call is invoked as follows:

```
#include <sys/vpm.h>
int fildes, cmd, arg;
ioctl (fildes, cmd, arg);
```

The values for the *cmd* argument are:

VPMSETC—Enable trace channels. This command enables each channel indicated by a 1 in the bit mask found in *arg*. The low-order bit (bit 0) corresponds to channel zero, the next bit (bit 1) corresponds to channel 1, etc.

VPMGETC—Get enabled channels. This command returns in *arg* a bit mask containing a 1 for each channel that is currently enabled.

VPMCLRC—Disable channels. This command disables the channels indicated by a 1 in the bit mask found in *arg*.

SEE ALSO

vpmsave(1M), vpm(7).

NAME

ts — TS11 magnetic tape interface

DESCRIPTION

The files **mt0**, ..., **mt15** refer to the Digital Equipment Corporation TS11 magnetic tape control and transports at 1600bpi. The files **mt0**, ..., **mt3**, **mt8**, ..., **mt11** are designated normal-rewind on close, and the files **mt4**, ..., **mt7**, **mt12**, ..., **mt15** are no-rewind on close. When opened for reading or writing, the tape is assumed to be positioned as desired. When a file is closed, a double end-of-file (double tape mark) is written if the file was opened for writing. If the file was normal-rewind, the tape is rewound. If it is no-rewind and the file was open for writing, the tape is positioned before the second EOF just written. If the file was no-rewind and opened read-only, the tape is positioned after the EOF following the data just read. Once opened, reading is restricted to between the position when opened and the next EOF or the last write. The EOF is returned as a zero-length read. By judiciously choosing **mt** files, it is possible to read and write multi-file tapes.

A standard tape consists of several 512 byte records terminated by an EOF. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time (although very inadvisable).

The **mt** files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the "raw" interface is appropriate. The associated files are named **rmt0**, ..., **rmt15**. Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

FILES

/dev/mt*, /dev/rmt*

BUGS

If any non-data error is encountered, it refuses to do anything more until closed. Note that during a rewind or space-forward operation, control is not returned until the operation has completed. The driver is limited to one transport.

NAME

tty — controlling terminal interface

DESCRIPTION

The file **/dev/tty** is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

FILES

/dev/tty
/dev/tty*

SEE ALSO

dz(7), tn4(7), tn74(7).

NAME

tu78 — TU78 magnetic tape interface

DESCRIPTION

The files **mt0, ..., mt15** refer to the Digital Equipment Corporation TU78 magnetic tape control and transports. The files **mt0, ..., mt7** are 1600bpi, and the files **mt8, ..., mt15** are 6250bpi. The files **mt0, ..., mt3, mt8, ..., mt11** are designated normal-rewind on close, and the files **mt4, ..., mt7, mt12, ..., mt15** are no-rewind on close. When opened for reading or writing, the tape is assumed to be positioned as desired. When a file is closed, a double end-of-file (double tape mark) is written if the file was opened for writing. If the file was normal-rewind, the tape is rewound. If it is no-rewind and the file was open for writing, the tape is positioned before the second EOF just written. If the file was no-rewind and opened read-only, the tape is positioned after the EOF following the data just read. Once opened, reading is restricted to between the position when opened and the next EOF or the last write. The EOF is returned as a zero-length read. By judiciously choosing **mt** files, it is possible to read and write multi-file tapes.

A standard tape consists of several 512 byte records terminated by an EOF. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time (although very inadvisable).

The **mt** files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the "raw" interface is appropriate. The associated files are named **rmt0, ..., rmt15**. Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

FILES

/dev/mt*, /dev/rmt*

BUGS

If any non-data error is encountered, it refuses to do anything more until closed. The driver is limited to four transports.

SEE ALSO

gt(7), ht(7).

NAME

un32 — magnetic tape interface

DESCRIPTION

The files **tp?** and **tp?n** refer to the UN32 magnetic tape controllers and associated transports. Only 1600bpi is available. The files **tp?** are designated normal-rewind on close, and the files **tp?n** are no-rewind on close. When opened for reading or writing, the tape is assumed to be positioned as desired. When a file is closed, a double end-of-file (double tape mark) is written if the file was opened for writing. If the file was normal-rewind, the tape is rewound. If it is no-rewind and the file was open for writing, the tape is positioned before the second EOF just written. If the file was no-rewind and opened read-only, the tape is positioned after the EOF following the data just read. Once opened, reading is restricted to between the position when opened and the next EOF or the last write. The EOF is returned as a zero-length read. By judiciously choosing **tp** files, it is possible to read and write multi-file tapes.

A standard tape consists of several 512 byte records terminated by an EOF. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time (although very inadvisable).

The **tp** files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the "raw" interface is appropriate. The associated files are named **rtp?** and **rtp?n**. Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. In raw tape I/O, the buffer must begin on a word boundary and the count cannot be greater than 2048 bytes. Seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

FILES

/dev/tp*, /dev/rtp*

NAME

un52 − magnetic tape interface

DESCRIPTION

The files **tp?**, **tp?h**, **tp?n** and **tp?hn** refer to the UN52 magnetic tape controllers and associated transports. Only 1600bpi is available on the UN52, while 1600/6250bpi densities are available on the UN52B. The files are designated **tp?** for 1600bpi rewind-on-close, **tp?n** for 1600bpi no-rewind on close, **tp?h** for 6250bpi rewind-on-close, and **tp?hn** for 6250bpi no-rewind on close.

The tape is assumed to be positioned as desired when opened. If the file was opened for writing a double file mark is written on the tape when closed; if the file was normal-rewind, the tape is rewound, otherwise the tape is positioned before the second file mark. If the file was opened read-only, and if the file was normal-rewind, the tape is rewound, otherwise the tape is positioned after the file mark following the data just read. Once opened, reading is restricted to between the position when opened and the next file mark or the last write. A file mark is returned as a zero-length read.

A standard tape consists of several 512 byte records terminated by a file mark. To the extent possible, the system makes it possible to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time (although inadvisable).

The **tp** files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, or when large records are to be read or written, the "raw" interface is appropriate. The associated files are named **rtp?**, **rtp?n**, **rtp?h** and **rtp?hn**. Each *read* or *write* call reads or writes the next record on the tape. During a write, the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. If the record was larger than the buffer size, the extra data is discarded. The buffer must begin on a word boundary, and the count must be an even number and cannot be greater than 6144 bytes. Seeks are ignored.

FILES

/dev/tp*, /dev/rtp*

BUGS

Once a file mark or any drive error is encountered, it refuses to do anything more until closed.

NAME

un53 — UN53/TN82 synchronous device interface

DESCRIPTION

The files **un53*** refer to the UN53 synchronous communications devices. Each physical UN53/TN82 pair provides for either one high speed (up to 50KB) or three low speed (up to 9.6KB) synchronous communications line(s).

The device interface permits simultaneous communication on multiple logical devices on a single UN53/TN82 pair.

FILES

/dev/un53* logical communications lines.

7

NAME

vp — Versatec printer

DESCRIPTION

Vp provides the interface to the Versatec electro-static line printer. Both printing and plotting capabilities are implemented.

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

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

The *commands* are:

LPRGETV Return the state of the printer.

LPRSETV Set the state of the printer to *arg*.

FILES

/dev/vp

SEE ALSO

vpr(1), lp(7).

7

NAME

vpm — Virtual Protocol Machine

DESCRIPTION

This entry describes the VPM protocol driver and gives an introduction to the Virtual Protocol Machine (VPM).

VPM is a software package for implementing link-level protocols on Programmable Communication Devices (PCDs) in a high-level language. This is accomplished by a compiler that runs on UNIX and translates a high-level language description of a protocol into an intermediate language that is executed by the PCD.

The VPM software consists of the following components:

1. A compiler (*vpmc*(1M)) for the protocol description language; it runs on UNIX.
2. A program that controls the overall operation of the PCD and executes the protocol script.
3. A protocol driver.
4. *vpmstart*: a UNIX command that copies a load module into the PCD and starts it.
5. *vpmset*(1M): a UNIX command that logically connects VPM minor devices with PCD synchronous lines.
6. *vpmsave*(1M): a UNIX command that writes unformatted trace data to its standard output.
7. *vpmfmt*: a UNIX command that formats the output of *vpmsave*(1M).

Operation of the VPM Protocol Driver

7

The VPM protocol driver provides a simple user interface to a synchronous line controlled by a link-level protocol executing in a PCD. It supports the following UNIX system calls: *open*, *read*, *write*, *close*, and *ioctl*. If higher levels of protocol are required, the VPM protocol driver may be modified or replaced.

The VPM protocol driver communicates with the level 2 protocol executing in the PCD using the Common Synchronous Interface (CSI). CSI is a device-independent interface between a level 3 protocol executing as part of the UNIX operating system and a level 2 protocol executing in a PCD. The interface consists of procedure calls implementing a number of commands and reports.

Before a VPM protocol driver minor device can be used, it must be logically connected to a synchronous line of some PCD. This connection can be made by means of *ioctl* commands (see below). The command *vpmset*(1M) uses these *ioctl* commands to make these connections.

This driver allows UNIX user processes to transfer data to or from a remote terminal or computer system through VPM. Flow control and error recovery are provided by the level 2 protocol executed by the PCD.

The VPM protocol driver *open* for reading-and-writing is exclusive; opens for reading-only or writing-only are not exclusive. The *open* sends a command to the PCD which causes it to start executing the protocol. The protocol driver then supplies one or more 512-byte receive buffers to the PCD.

The *read* returns either the number of bytes requested or the number remaining in the current receive buffer, whichever is less; any remaining bytes in the current receive buffer are used to satisfy subsequent reads. The data from each user *write* is copied into one or more 512-byte buffers and passed to the PCD for transmission.

The *close* arranges for the return of buffers and for a general cleanup when the last transmit buffer has been returned by the interpreter. It also stops the execution of the protocol.

The VPM protocol driver *ioctl* system call has the form:

ioctl (fildes, cmd, arg)

Possible values for the *cmd* argument are:

VPMCMD—Send a command to the protocol script. The first four bytes of the array pointed to by *arg* are passed to the PCD, which saves them and passes them to the protocol script when it requests them via a *getcmd* primitive. Only the most recent command is kept by the PCD.

VPMERRSET—Set the maximum values for the error counters to the values in the array pointed to by *arg*. The array is assumed to contain eight (short) values. When a VPMERRS *ioctl* is executed the VPM protocol driver returns the current values of the error counters and sets them to the values defined by the VPMERRSET command. The error counters are then decremented when errors occur until they reach zero. If VPMERRSET is not done all reset values will default to zero. If a reset value is zero the corresponding error counter will be ignored.

VPMERRS—Get and then reset the error counters. The error counters are copied to the array pointed to by *arg*, which must be large enough to contain eight (short) counters. The error counters are then set to the values specified by a previous VPMERRSET, if any, zero otherwise.

VPMRPT—Get the latest script report. When the protocol script executes a *rtnrpt* primitive, a four-byte report is passed from the protocol script to the VPM protocol driver which saves it for later use. Only the most recent script report is kept by the driver. If there is a script report that has not previously been passed to a user via this *ioctl* command, that report is copied to the array pointed to by *arg* and a 1 is passed as the return value. If no script report is available, a zero is passed as the return value.

7

VPMSDEV—Connect a protocol driver minor device to a synchronous line of a PCD. *Arg* is an *int* containing the major and minor device numbers of the PCD. When using a DEC PCD the synchronous line number is encoded in bits 5-7 of the major/minor device number. To invoke this *ioctl* command, the file status flag O_NDELAY must be set.

VPMDETACH—Disconnect the protocol driver minor device and the PCD synchronous line. To invoke this *ioctl* command, the file status flag O_NDELAY must be set.

VPMOPTS—Set the protocol options. The previous options are cleared and the options represented by *arg* are set. The protocol options may be retrieved by the protocol script using the *getopt* primitive (see *vpmc*(1M)). When running *cslapb.r* as the level 2 protocol script, an octal 01 in *arg* will indicate to the script that it should use the B address (see *vpmset*(1M)). To invoke this *ioctl* command, the file status flag O_NDELAY must be set.

VPMPCDOPTS—Set the PCD options. If the PCD requires options, the previous options are cleared and the option represented by *arg* are set. To invoke this *ioctl* command, the file status flag O_NDELAY must be set. The following constants, which may be combined with the OR operator to form *arg*, apply only to the HDLC versions of the VPM interpreter as implemented on the DEC KMC and KMS:

HWLOOP—This option causes the interpreter to set maintenance loopback mode on the synchronous line interface associated with a particular line. This option is used by *vpmtest*(1M).

ADRSWTCH—This option causes the interpreter to invert (complement) the address bit (bit 2 of byte 0) of each transmitted frame. This allows the BX.25 level 2 protocol script *cslapb.r* to operate in a loopback mode. This option is used by *vpmtest*(1M).

X25MODE—If this option is set, the interpreter places the first three bytes of the data portion of each received frame into the octet portion of the buffer descriptor instead of into the data buffer. Similarly, the first three bytes of the data portion of each transmitted frame are taken from the octet portion of the buffer descriptor. This mode is used by the PDP11 version of the BX.25 level 3 driver.

VPMSTAT—This puts into the integer variable pointed to by *arg* the CSI index associated with the protocol minor device.

The VPM Event Trace

The VPM protocol driver and CSI routines generate a number of event records to allow the activity of the protocol driver, the interface routines and the protocol script to be monitored for debugging purposes. If a program such as *vpmsave*(1M) has opened minor device 0 of the trace driver and has enabled the appropriate channels on that device, these event records are queued for reading; otherwise the event records are discarded by the trace driver. Event records associated with CSI interface index *n* *modulo* 16 are put on the read queue for minor device 0 of the trace driver with a channel number of *n*. Calls to the system functions *vpmopen*, *vpmread*, *vpmwrite*, and *vpmclose* generate event records identified respectively by *open*, *read*, *write*, and *close* when the output of *vpmsave* is formatted and printed by *vpmfmt*. Calls to the *vpmc*(1M) primitive *trace*(*arg1*, *arg2*) cause the PCD to pass *arg1* and *arg2* along with the current value of the script location counter to the VPM driver, which generates an event record identified by **trace**.

When the script terminates for any reason, the driver is notified and generates an event record identified by **INTterm**. This record also contains the CSI minor device number, the script location counter, and a termination code; the code indicates the reason for termination as follows:

0 Normal termination; the PCD received a HALT command from the driver.
1 Undefined virtual-machine operation code.
2 Script program counter out of bounds.
3 Interpreter stack overflow or underflow.
4 Jump address not even.
5 UNIBUS error.
6 Transmit buffer has an odd address; or the driver tried to give the PCD too many transmit buffers; or a *get* or *rtnxbuf* was executed while no transmit buffer was open, i.e., no *getxbuf* was executed prior to the *get* or *rtnxbuf*.
7 Receive buffer has an odd address; or the driver tried to give the PCD too many receive buffers; or a *put* or *rtnrbuf* was executed while no receive buffer was open, i.e., no *getrbuf* was executed prior

to the *get* or *rtnxbuf*.

8 The script executed an *exit* primitive.
9 A *crc16* was executed without a preceding *crcloc* execution.
10 The PCD detected loss of the modem-ready signal at the modem interface.
11 Transmit-buffer sequence-number error (internal error).
12 Command error: an invalid command or an improper sequence of commands was received from the driver.
13 Not used.
14 Invalid transmit state (internal error).
15 Invalid receive state (internal error).
16 Not used.
17 *Xmtctl* or *setctl* attempted while transmitter was still busy.
18 Not used.
19 Same as error code 6.
20 Same as error code 7.
21 Script too large.
22 Used for debugging the PCD.
23 The driver's OK-check has timed out.
24 The array specified as an argument to a *getcmd* primative is too close to end of user's data space in the PCD.
25 PCD driver unable to accept command.
26 The PCD's OK-check has timed out.
27 No such line number on PCD.

SEE ALSO

vpmc(1M), vpmsave(1M), vpmset(1M), trace(7), x25(7).

NAME

x25 — BX.25 network interface

DESCRIPTION

The X25 driver provides multiplexed channels over one or more synchronous communications lines using the Bell System standard BX.25 Level 3 protocol. The current implementation supports permanent virtual circuits (PVCs) only; the call set-up features needed to support virtual calls have not yet been implemented. There is a separate and independent Level 3 interface for each communications line. Point-to-point connections between hosts are supported as well as connections via an X.25 network.

The X25 uses the *Common Synchronous Interface* (CSI) to access communications lines controlled by various kinds of programmable communications devices (PCDs). Level 2 of BX.25, the link level, is implemented by software or firmware in the PCD. On DEC machines, the PCD is a DEC KMC11-B microprocessor, using the UNIX Virtual Protocol Machine (VPM) software package. On the 3B20S, the PCD is a TN75B or TN82 peripheral controller.

The special files **/dev/x25/s***?* refer to the minor devices of the X25 driver. Each such minor device, also referred to as a *slot*, can be connected by means of a *network control* device (see *nc*(7)) to an arbitrary logical channel (1-4095) on a specified X25 interface. Provided the other end of the logical channel has been connected in an analogous fashion, each slot so connected is the terminus of a *permanent virtual circuit*, which is a full-duplex connection between a set of user processes on the local host and another set of user processes on a remote host. A logical channel is a connection which may be multiplexed with other channels over a physical link to a remote host or an X.25 network. Each X25 interface (also referred to as a *link*) must be connected via the network-control device to a particular synchronous line.

7

The X.25 driver includes the BX.25 link backup facility. This facility provides for automatic changeover to a backup synchronous line which may be configured for any X25 interface. A changeover could occur for several reasons: if there is a failure on the link, i.e. physical severing of the link (Level 1) or a failure at Level 2; the link is noisy and produces too many errors; a changeover to the backup link is requested via the *nc* device; or the remote end of the link initiates a changeover. Level 3 will be unaware of the changeover and any lost packets will be recovered by the Level 3 recovery procedures. The procedures for configuring backup links and requesting a changeover to the backup link are described in the manual entries for x25pvc(1M) and nc(7).

A user process accesses a BX.25 minor device (slot) using *open*, *close*, *read*, *write*, and *ioctl* system calls.

There are several internal flags that are maintained by the X25 driver for each slot. The values of these flags can be read and in some cases modified by means of the *ioctl* system call (see below).

An *open* and return the error ENXIO if the specified slot is not configured, ENODEV if the slot is not currently connected to a logical channel on some link, or EL3HLT if the link to which the slot is connected is not currently active. The user may request the normal *open* options O_RDONLY, O_WRONLY, and O_RDWR. The user may also request that reads with no data available should not sleep by using the O_NDELAY *open* flag, or that the *open* is to be exclusive by using the O_EXCL *open* flag. If an exclusive *open* is requested and the slot is already in use, the error EACCES will be returned. A successful *open* will clear the *isreset* status bit (see the discussion of *ioctl* below).

An *open* may or may not block until the far end is also open, depending on the session-establishment protocol requested. The choice of session-establishment protocol is made by means of the network-control device at the time the permanent virtual circuit is installed. There are three possibilities: the first mode, referred to as the "no-protocol" session mode, is for the *open* to return immediately. This puts the burden on the user program to determine whether the far end is actually open. The reset session mode, intended only for compatibility with certain non-UNIX implementations of BX.25, uses a RESET in-order packet to indicate to the far end that the slot has been opened and a RESET out-of-order packet to indicate to the far end that the slot has been closed. In the current implementation, the RESET in-order and RESET out-of-order packets are recognized when they are received, but are not transmitted (so-called "passive" mode). To avoid data loss with this mode, the application on the non-UNIX side must wait until it receives data from the UNIX side (allow the UNIX side to perform the first write) before it sends any data to UNIX. The third mode, which is the one recommended for most applications, uses BX.25 session-layer *Connect/Accept* qualified data messages to indicate that a slot has been opened and session-layer *Disconnect* qualified data messages to indicate that a *close* has occurred. In the last two modes, an *open* will block until the indication that the far end is open has been received, unless the O_NDELAY *open* option was specified, in which case the status of the far end of the PVC must be obtained by using *ioctl* (see below).

Regardless of the session-establishment protocol in effect, data which is received while a slot is not open will be acknowledged and silently discarded. However, if the session-layer *open/close* protocol is selected, no data can be transmitted until both ends of the PVC have been opened.

The data specified by each *write* is transmitted as a single BX.25 message, possibly multi-packet. The user has the option of waiting for acknowledgement of the last packet of each message before the *write* returns; this feature is called *delivery confirmation* (see the discussion of *ioctl* below). Even if the O_NDELAY mode was requested, the user process will be put to sleep if the amount of data in the transmit queue for the slot exceeds some high-water mark; the process will be given a wake-up when the transmit queue has been drained to the low-water mark. If the slot is not open for writing and a *write* system call is issued, the error EBADF is returned.

A user reads in record mode, which means that each *read* will return data from a single message only. If the slot is not open for reading and a *read* system call is issued, the error EBADF is returned. If the count specified on the read request is not large enough to accommodate the entire message, the remainder of the message will be returned on subsequent *reads*. The message-continued flag (*messcont*) will be set when a partial read occurs; this flag will be cleared when the last byte of the message is finally read. If no data is available, the user process will be put to sleep, unless the O_NDELAY option was specified on the *open* or the equivalent mode was set via an *fcntl* system call. If O_NDELAY was specified and no data is available, the *read* will return zero bytes. If a partial message is available and O_NDELAY has been set, the *read* will return zero until the end of the message has been received or the count can be filled. However, if the channel is flow-controlled, the read will return the partial message even if O_NDELAY is set. If O_NDELAY is not set, the *read* will sleep until the entire message has been received or the count has been satisfied.

7

Zero-length messages will be sent and received as such (but see *BUGS* below).

If the *faropen* flag (described below) is not set, a *write* will return immediately with a count of zero. A *read* will return a zero-length record (indicating end-of-file) if *faropen* is not set and the receive queue is empty. If the end of a message is on the input queue, a *read* will *not* return a zero indicating end-of-file, regardless of the state of the slot. Note that this means that the flags returned by *ioctl*, indicating that the far end has closed or a RESET has occurred, may be set before they take effect on user reads.

Except as just noted, once a RESET has occurred (indicating possible data loss), all reads and writes will fail, returning the error EL3RST, until the *isreset* flag is cleared by an *ioctl* or a successful *open*.

If the state of the channel is halted, all calls to the BX.25 subsystem for that channel will fail with error EL3HLT. This occurs when a link dies or a severe error causes the protocol to be stopped on the channel or link. If the channel is not in the halted state but level 2 has lost synchronization, the error EL2NSYNC is returned.

Signals will cause reads and writes to return the amount actually read or written, unless it was zero bytes, in which case the error EINTR will be returned. However, if O_NDELAY was specified, the amount actually read or written will be returned, whether zero or greater. If a *write* is interrupted by a signal, the data already packetized will be transmitted as a BX.25 message; that is, a subsequent *write* will always begin a new message. The return value from the *write* will indicate the number of bytes actually queued for transmission.

When the last user closes, any unread data in the receive queue will be discarded. Data in the transmit queue will *not* be discarded, but will be transmitted normally. When the transmit queue has been drained, the session take-down protocol, if any, is then followed (either to send a session-layer *Disconnect* message or to mark the channel as being out of use, so that incoming packets can be discarded).

Several options and actions can be requested via the *ioctl* system call, which takes three arguments: *fildes*, *request*, and *arg*. To use this *ioctl* system call, the line

```
#include        <sys/x25user.h>
```

must be included in the user program. Possible values for the *request* argument are:

X25SET Set the flags and options for the slot or channel.

X25GET Return the status information for the channel and its associated link. The structure pointed to by *arg* will receive the values described below.

The structure pointed to by *arg* for X25SET is:

```
struct x25sctl {
        ushort
                c_delconf: 1, /* delivery confirmation */
                c_isreset : 1, /* channel reset not cleared */
                c_ndelay : 1; /* no delay requested */
};
```

Only the *delconf* and *isreset* flags can be changed by *ioctl*.

The flag bits are further explained as follows:

c_delconf
While this bit is set, each *write* system call will block until the last packet of the corresponding message has been acknowledged; another writer, if one exists, will also be blocked until the previous writer's message is acknowledged. If this bit is not set, a user *write* can return immediately after the message has been completely packetized and queued for transmission, possibly allowing several out-going messages to be unacknowledged at the same time.

c_isreset
This flag if set indicates that a RESET has occurred. The user may clear this flag by setting the corresponding flag bit to 1 in the value passed by X25SET. (The user may not set this flag). If the value for this flag passed by X25SET is 0, the internal value of this flag will *not* be changed. This flag is always cleared by a successful *open*.

c_fndelay
This flag if set indicates that the O_NDELAY flag has been set. If this flag is not set on an *open* system call, it can be set or cleared on the *fcntl* system call. The value of this bit may not be changed by using the *ioctl* system call.

The structure pointed to by *arg* for X25GET is:

```
struct x25gctl {
        struct t_state  c_t_state;   /* tab state structure */
        struct l_state  c_l_state;   /* link state structure */
        struct s_flags  c_s_flags;   /* slot flags structure */
        unsigned char c_xmtq;        /* length of lev 3 xmtq */
        unsigned char c_recvq;       /* length of lev 3 recvq */
        unsigned char c_rststate;    /* restart state of link */
        struct l_line   c_primline;  /* primary line info */
        struct l_line   c_bckline;   /* backup line info */
};
```

The structure returned contains information about the channel and its associated link. Macros are provided to obtain some of the more useful information. These macros are especially useful in determining the conditions of the channel and link when a system call to a BX.25 minor device returns an error indicating an abnormal situation. To use the macros, a user must issue an *ioctl* using the file descriptor of the BX.25 minor device. Each macro has an argument *arg* which is a pointer to an *x25gctl* structure. The following macros are provided:

X25LRDY	true if the link has completed restart procedures
X25LHLT	true if the link is halted
X25L2FUL	true if the level 2 queue of the link is full
X25ABNHLTP	true if the link is halted because of an abnormal condition on the primary device
X25NRMHLTP	true if the link is halted because of a normal halt command on the primary device
X25L2ERRP	the value of the error code returned by the primary device when it last halted
X25L2RDYP	true if level 2 is synchronized on the primary device
X25L2FCP	the number of times the level 2 retry counter reached its maximum

7

X25LATCHP	true if the link is attached to a primary device
X25CHLT	true if the state of the channel is halted
X25CRDY	true if the channel is in the data transfer state
X25FAROPN	true if the far end of the channel is open
X25ISRESET	true if the channel has just completed reset procedures
X25MSGCON	true if the message was not completely consumed by last user read
X25L3XQFUL	true if the level 3 transmit queue is full
X25L3RQFUL	true if the level 3 receive queue is full

In order to access information not obtained using the macros, the user may look at the structures declared in */usr/include/sys/x25u.h* and */usr/include/sys/x25.h* to determine how to access the desired values directly.

The following is an example of a situation where an *ioctl* call would be useful. Assume that only a primary device has been attached to the link associated with the channel being used. If the error EL3HLT is returned when a system call is issued, an *ioctl* call would then be issued. The EL3HLT error return indicates that the link is in the halted state and the macros can provide more useful information. The macro UABNHLTP(*arg*) will return true if the link halted abnormally. The macro UL2ERRP(*arg*) will then return the value of the error code returned by the primary device. The manual entry *vpm*(7) describes what the error codes mean and the file */usr/include/sys/csierrs.h* contains defines for these errors. If the link halted normally, this indicates that someone entered a command to halt the link.

7

SEE ALSO

x25pvc(1M), fcntl(2), ioctl(2), open(2), read(2), write(2), nc(7), vpm(7).
Operations Systems Network Protocol Specification: BX.25 Issue 2.

BUGS

The multiplicity of options for the *open/close* protocol reflects a lack of standardization and a certain amount of confusion. However, in the near future, the session layer will be implemented and will handle this problem so that the user will not have to select an option or have to worry about *open/close* synchronization.

It is not clear that the treatment of the O_NDELAY flag is correct; this is an area that is particularly likely to change. In particular, the read partial message/return zero dilemma for *read* O_NDELAY calls is puzzling. One would like to return zero until the entire message has been received, but a long, multi-packet message could deadlock such a scheme. Thus, the "read something if flow-controlled" method was used.

At present, there is no way to tell whether a return value of zero for a no-delay *read* is due to a zero-length message or to the lack of anything to read.

It would be dangerous to assume that zero-length messages will be propagated correctly through an X25 network or that they will be treated in a compatible manner by other implementations of BX.25.

There is no way to send an INTERRUPT packet. An INTERRUPT packet received from the far end will be confirmed and discarded.

NAME

intro — introduction to system maintenance procedures

DESCRIPTION

This section outlines certain procedures that will be of interest to those charged with the task of system maintenance. Included are discussions on such topics as boot procedures, recovery from crashes, file backups, etc.

BUGS

No manual can take the place of good, solid experience.

NAME

3B20boot — 3B20S bootstrap procedures

DESCRIPTION

Lboot is a program that is read in from the boot section of the disk in response to the BOOT command on the EAI page at the console. See *eai*(8) for a description of the console operations. Other options specified on the EAI page control the functions performed by *lboot*.

If the SEC−DISK flag is clear on the EAI page when the BOOT command is issued, moving head disk 0 is used as the boot device. If the SEC−DISK flag is set, moving head disk 1 will be used as the boot device. The BACKUP−ROOT flag controls which one of two file systems on the chosen disk will be used. If the flag is clear, the primary root file system on the disk will be used. If the flag is set, the backup root file system will be used.

The PROMPT−UNIX flag is used to specify the name of the program to be booted. If the PROMPT−UNIX flag is clear, `/unix` on the chosen file system will be booted. If the PROMPT−UNIX flag is set, *lboot* will ask for the name of the program to be booted. (NOTE: To respond to *lboot*, you must be out of the EAI command area. Use the NORM DISP or CMD−MSG key to get out of the command area.) If the name given is a directory on the chosen file system, *lboot* will respond with a listing of the files present in the directory. If the name given is a normal, executable file in *a.out* format, it will be loaded into memory and control will be transferred to it.

Any standard `/unix` file will look at the MIN−CONFIG and INH−CACHE flags when it begins execution. If the MIN−CONFIG flag is clear, all of the peripheral devices will be brought into service. See *don*(1M) for the normal device configuration mechanism. If the MIN−CONFIG flag is set, `/unix` will only bring the boot device, the system console, and a tape drive into service and only the first megabyte of main memory will be used. If the INH−CACHE flag is set, `/unix` will leave the cache memory disabled. If the INH−CACHE flag is clear, the cache will be enabled.

SEE ALSO

don(1M), newboot(1M), dsk(7), eai(8), prm(8).

8

DIAGNOSTICS

Self-explanatory messages about bad directory entries and bad file formats. The following code words are used in success PRMs:

Code	*Meaning*
dfcn	DFC number
dskn	Moving head disk number

The success PRM's issued during the boot process are:

PRM	*Meaning*
`E100 0000` *dfcn* *dskn*	DFC is in service
`E100 BBBB 3B3B 3B3B`	Specified program loaded into memory
`E200 0000 0000 0ADD`	UNIX is checking memory
`E200 0000 0000 0001`	UNIX is initializing I/O devices
`E200 0000 0000 0002`	UNIX is ready to mount ROOTDEV
`EC00 3B3B 3B3B 3B3B`	UNIX is ready to run user processes

Failure PRM's can be found in *prm*(8).

BUGS

Lboot never uses the cache memory. *Lboot* isn't smart enough to know what *a.out* files can be used as bootable programs. If an *a.out* is specified that is not a bootable program, *lboot* will load it in and branch to it. What happens after that is unpredictable.

NAME

3B20ops – 3B20S console operations

DESCRIPTION

The daily procedures involved in running UNIX on the 3B20S system are described here.

Disk Boot

See *3B20boot*(8) for a complete description of how to boot the machine from disk. The # prompt indicates that the system has come up through **init** S (see *init*(1M)) and that the shell is ready to accept commands.

This is the appropriate time to do file system backups, and *fsck*(1M) should be executed. One must never operate the system with a defective file system.

After successful completion of *fsck*, the operator can bring the system to multi-user operation by executing **init 2**.

Bringing the System Down

The shutdown procedure is designed to gracefully turn off all processes and bring the system back to single user state with all buffers flushed. To do this the operator can execute *shutdown*(1M) or the following sequence of commands:

```
killall
sync
telinit 6
fsck (optional)
```

The system may then be halted using the Emergency Action Interface (see *eai*(8)).

System Dumps

After a system crash the procedure outlined in *eai*(8) should be followed to dump the contents of memory to the disk. Then the dump can be analyzed using *crash*(1M) and the **/dev/dump***?* file (see *dsk*(7)). To save the dump for later examination, the dump may be copied to a file using the command:

dd if=/dev/dump*U* of=savefile

where *U* is the unit (drive) number of the disk containing the dump.

SEE ALSO

crash(1M), date(1), filesave(1M), fsck(1M), init(1M), shutdown(1M), system(4), 3B20boot(8), eai(8).

8

NAME

70boot — 11/70 bootstrap procedures

DESCRIPTION

To bootstrap programs from a wide range of storage media, the PDP-11/70 has a dedicated diagnostic bootstrap loader called the M9301-YC. The M9301-YC contains two 256 word ROMs (17 765 000 to 17 765 776 and 17 773 000 to 17 773 776) which contain hardware verification diagnostic routines and bootstrap loader routines.

The diagnostic portion tests the basic CPU to verify correct operation. The branches, registers, all addressing modes, and most of the instructions are checked. If requested, memory management and the UNIBUS map are turned on. Then memory is tested from virtual address 001 000 to 157 776 with the cache disabled. Next the cache is enabled and tested.

The physical memory tested is determined by the console switches. Console switches <15:12> are used to set physical address bits <19:16>. If console switches <15:12> are zero, memory management and the UNIBUS map will not be enabled, so that physical memory 0 to 157 776 will be used. If console switches <15:12> are non-zero, then memory management, the UNIBUS map, and 22-bit mapping will be enabled. Table I describes the physical address ranges for each switch setting. In all cases, virtual addresses 160 000 to 177 776 are mapped to the peripheral page, physical addresses 17 600 000 to 17 777 776. Note that physical memory above 512K words is not accessible by this program even though the physical memory maximum is 1920K words.

The bootstrap portion of the M9301-YC attempts to BOOT from the device and drive number specified in the console switches. Console switches <7:3> select the device and console switches <2:0> select the drive number. Table II describes the devices selected for each switch setting. If console switches <7:0> are zero, the program will read a set of switches on the M9301-YC, set by field service, to determine a default boot device and drive number. These switches appear at location 17 773 024, however bits <8:4> select the device and bits <3:1> select the drive number.

Having selected a boot device, the program will read a block of data into memory starting at virtual address 0, and then jump to virtual address 0. Table III describes the details of booting for each device. Note that the physical address selection is the same as described above for the diagnostic portion. Excluding the RX11/RX01 floppy disk, bootstrap programs must fit in one block of 256 words, even though this program may read in more.

To start operation of the bootstrap loader, halt the CPU by depressing the HALT switch, set the Address Display select switch to Console Physical, set the Console Switch Register to 165 000, and depress the Load Address switch. Then reset the console switches to 0 and set switches <15:12> for the desired physical memory (normally 0) and switches <7:0> for the desired device (normally 0 for the default boot). Put the HALT switch in the ENABLE position and depress the START switch. The diagnostic portion will then run followed by the boot from the selected media. This takes approximately three seconds.

Any error during the diagnostic portion will cause the CPU to halt. Table IV lists the addresses and error indications. Only cache errors are recoverable in that by pressing the CONTINUE switch the program will disable the cache by forcing misses and proceed to the bootstrap section. If there is an error in reading the boot block, the program will do a RESET instruction and jump back to the memory test section (test 24) and then attempt to boot again.

SEE ALSO

romboot(8), unixboot(8).

Table I — Physical Memory Selection

Console switches <15:12>	Physical addresses
00	00 000 000 - 00 157 776
01	00 200 000 - 00 357 776
02	00 400 000 - 00 557 776
03	00 600 000 - 00 757 776
04	01 000 000 - 01 157 776
05	01 200 000 - 01 357 776
06	01 400 000 - 01 557 776
07	01 600 000 - 01 757 776
10	02 000 000 - 02 157 776
11	02 200 000 - 02 357 776
12	02 400 000 - 02 557 776
13	02 600 000 - 02 757 776
14	03 000 000 - 03 157 776
15	03 200 000 - 03 357 776
16	03 400 000 - 03 557 776
17	03 600 000 - 03 757 776

Table II — Device selection

Console switches <7:3>	Device	
00	illegal	
01	TM11/TU10	Magnetic tape
02	TC11/TU56	DECtape
03	RK11/RK05	Disk pack
04	RP11/RP03	Disk pack
05	reserved	
06	RH70/TU16	Magnetic tape
07	RH70/RP04	Disk pack
10	RH70/RS04	Fixed head disk
11	RX11/RX01	Diskette
12-37	illegal	

8

Table III — Boot procedures

TU10:	Select drive, wait until online, set to 800 bpi, rewind, space forward 1 record, read 1 record (maximum of 256 words).
TU56:	Select drive, rewind, read 512 words.
RK05 or RP03:	Select drive, start at block 0, read 512 words.
TU16:	Select drive on first TM02, wait until online, set to 800 bpi, PDP format, rewind, space forward 1 record, read 1 record (maximum of 512 words).
RP04:	Select drive, read-in preset, set to 16-bits/word, ECC inhibit, start at block 0, read 512 words.
RS04:	Select drive, start at block 0, read 512 words.
RX01:	Select drive 0 or 1, start at track 1, sector 1 (IBM standard), read 64 words.

8

Table IV — Error halts

Address displayed	Test	Subsystem under test
17 765 004	1	Branch
17 765 020	2	Branch
17 765 036	3	Branch
17 765 052	4	Branch
17 765 066	5	Branch
17 765 076	6	Branch
17 765 134	7	Register data path
17 765 146	10	Branch
17 765 166	11	CPU instruction
17 765 204	12	CPU instruction
17 765 214	13	CPU instruction
17 765 222	14	CPU instruction
17 765 236	14	CPU instruction
17 765 260	15	CPU instruction
17 765 270	16	Branch
17 765 312	16	CPU instruction
17 765 346	17	CPU instruction
17 765 360	20	CPU instruction
17 765 374	20	CPU instruction
17 765 450	21	Kernel PAR
17 765 474	22	Kernel PDR
17 765 510	23	JSR
17 765 520	23	JSR
17 765 530	23	RTS
17 765 542	23	RTI
17 765 550	23	JMP
17 765 742	25	Main memory data compare error
17 765 760	25	Main memory data compare error
17 776 000	25	Main memory parity error; no recovery possible from this error
17 773 644	26	Cache memory data compare error
17 773 654	26	Cache memory no hit, recoverable
17 773 736	27	Cache memory data compare error
17 773 746	27	Cache memory no hit, recoverable
17 773 764	25/26	Cache memory parity error, recoverable

8

NAME

750ops — VAX-11/750 console operations

DESCRIPTION

The procedures described here include the major operational sequences involved in running UNIX on the VAX-11/750 system. Note that these procedures are different from those used on the VAX-11/780. For the VAX-11/780, see *780ops*(8). The following notation is used:

1. Special characters are enclosed in <> (e.g., **<ctl>** represents the "control" key, and **<cr>** stands for the "carriage return" key).
2. Items within **{}**s are mandatory substitutions.
3. Items within **[]**s are optional.

DAILY PROCEDURES

Disk Boot

There is no floppy disk drive and controller supplied with the console subsystem of the VAX-11/750, nor an LSI-11 microprocessor as in the VAX-11/780. Instead a TU58 tape cartridge is provided in the CPU cabinet. It may be used to boot the system (i.e., the bootstrap procedure can be stored in this cartridge).

When the system is first turned on, the console prompt >>> is printed. If UNIX has been shut down, but not halted (see *Bringing the System Down*), the operator must type **<ctl>p** to get into console mode. This also halts the CPU.

With the system halted, any of the console commands may be executed as described below under *Console Operation*.

To boot the stand-alone shell (*sash*) from the default disk drive, the operator types **B<cr>**. The following is an example of this operation as seen on the console, picking up after the **<ctl>p**:

```
>>>B<cr>
       %%

$$
```

There is a four-position switch in the front panel that selects the boot device. The boot command will boot from the device selected by this switch. Alternately, the boot command may have an argument that selects the boot device superseding the switch-selected one.

```
>>>B [ddcu] <cr>
```

where *dd* is a device code from:

Code	Device
DL	RL01/02
DB	RP04/05/06/07, RM03, RM&80
DD	DECTAPE II Cartridge (TU58)
DM	RK06/07

c specifies the I/O channel adapter and *u* is one digit identifying the device drive number. For example,

```
>>>B DDA0 <cr>
```

will boot from the TU58.

The **$$** prompt indicates that the stand-alone shell (*sash*) is ready to accept commands. If it is desired to run stand-alone *fsck*(1M) (or other stand-alone functions), this is the time to do it. The commands have the form **/stand/***program* where *program* can be any name from a limited list of

UNIX commands found in the directory **/stand**. To perform a file system consistency check, type:

$$ /stand/fsck /dev/rp0

To bring up UNIX, the operator must type **unix<cr>**. The system will come up through **init S** (see *init*(1M)).

This is the appropriate time to do file system backups, and *fsck*(1M) should be executed if it was not executed in the stand-alone section of the boot. One must never operate UNIX with a defective file system.

After successful completion of *fsck*(1M), the operator can bring the system to multi-user operation by executing **init 2**.

Bringing the System Down

The shutdown procedure is designed to gracefully turn off all processes and bring the system back to single user state with all buffers flushed. To do this the operator should execute *shutdown*(1M). If *shutdown* is not successful, use the following sequence of commands:

```
killall
sync
telinit 6
fsck (optional)
```

The system may then be halted by typing **<ctl>p.**

System Dumps

After a system crash, the following procedure should be used to get a system dump on tape.

1. Mount a tape with write ring and bring it on-line.
2. Halt the system and enter console mode with **<ctl>p**.
3. Issue the following command sequence, each command followed by <cr>:

   E/G 0 (*Keep typing E's until all registers have been examined: R0 thru R15*)

   S 400 (*Start execution at 400, i.e., dump to tape*)
4. Before returning to UNIX, execute the stand-alone *fsck*(1M).

INSTALLATION BOOT PROCEDURE

Tape Boot

The user must type in the following program to read the first record on tape drive 0. Type <cr> at the end of each input line:

```
>>>H
>>>I

>>>D/G E 20000
>>>D/P 20000 F5508FD0
>>>D + D05000FF
>>>D + F308008F
>>>D + A0B45100
>>>D + D0421002
>>>D + 0000008F
>>>D + 8FD06180
>>>D + 80000100
>>>D + CFDE04A1
>>>D + CA53003A
>>>D + FFFE008F
>>>D + 8FC853FF
>>>D + 00000200
```

8

```
>>>D + 6053B053
>>>D + 8FB01B10
>>>D + 25CF0200
>>>D + 1DCFB400
>>>D + 018FB000
>>>D + 0014CFC0
>>>D + F46053B0
>>>D + BF002BCF
>>>D + 808FB300
>>>D + 1302A000
>>>D + 000005F8
>>>D + 029AC004
>>>D + 00080000
>>>D + 00000000
>>>D + 00000000
>>>D + 00000000
>>>D + 028C0000
>>>D + 000E0000
>>>D + 00010000
>>>D + 00000000
>>>S 20000          (Start tape load)

        00020055 06

>>>S 0              (Execute boot program loaded from tape)
```

From this point the loader initiates a question and answer sequence to control the remainder of the load process.

CONSOLE OPERATION

The following is condensed from Chapter 6 of the *VAX Hardware Handbook*, DEC, 1980−81.

The following are the standard console commands. The most abbreviated form is shown in parentheses.

<ctl>P — Puts the machine in Console I/O mode and halts the processor. A halt message is printed.

<ctl>U — Deletes the current input line.

<del> — Deletes the previous character.

(E)XAMINE {address}
: Displays 8-digit hexadecimal address and its contents.

(D)EPOSIT {address} {data}
: Enters data to address. (Refer to *VAX Hardware Handbook* for EXAMINE and DEPOSIT qualifiers.)

(I)NITIALIZE — Initializes CPU.

(H)ALT — The HALT command is implemented on the VAX-11/750 for the sake of consistency with the VAX-11/780. It does not actually halt the CPU since the CPU must already be halted to respond to the command (see **<ctl>P** above). It does reset the console defaults.

(S)TART {address}
: Initializes CPU, enters address to PC, issues CONTINUE to CPU, and puts console into Program I/O mode.

(C)ONTINUE — Allows the user to restart a halted program without altering the state of the machine.

FILES

/etc/shutdown
/stand/*

SEE ALSO

filesave(1M), fsck(1M), init(1M), shutdown(1M), tapeboot(8).

8

NAME

780ops — VAX-11/780 console operations

DESCRIPTION

The procedures described here include the major operational sequences involved in running UNIX on the VAX-11/780 system. Note that these procedures are different from those used on the VAX-11/750. For the VAX-11/750, see *750ops*(8). The following notation is used:

1. Special characters are enclosed in <> (e.g., **<ctl>** represents the "control" key, and **<cr>** stands for the "carriage return" key).
2. Items within { }s are mandatory substitutions.

DAILY PROCEDURES

Disk Boot

This procedure can be used only on a system with a floppy disk updated for use with UNIX. If the floppy disk has not been so updated, the sequences shown below under *UNIX Floppy Update* must be performed.

When the system is first turned on, the console prompt >>> is printed. If UNIX has been shut down, but not halted (see *Bringing the System Down*), the operator must type **<ctl>p** to get into console mode. After the prompt, type **H<cr>** to halt the system.

With the system halted, any of the console commands may be executed as described below under *Console Operation*.

To boot the stand-alone shell (*sash*) from the default disk drive, the operator types **B<cr>**. Alternatively, the boot command may have an argument that selects the boot device superceding the default. For example,

```
B RP0
```

will boot from disk drive 0, and

```
B RP1
```

will boot from drive 1.

The following is an example of this operation, starting after the **<ctl>p** command:

```
>>>H<cr>
          HALTED AT nnnnnnnn
>>>B<cr>
          CPU HALTED
          INIT SEQ DONE
          HALT INST EXECUTED
          HALTED AT nnnnnnnn
          LOAD DONE, nnnnnnnn BYTES LOADED
$$
```

The **$$** prompt indicates that the stand-alone shell (*sash*) is ready to accept commands. If it is desired to run stand-alone *fsck*(1M) (or other stand-alone functions), this is the time to do it. The commands have the form **/stand/***program* where *program* can be any name from a limited list of UNIX commands found in the directory **/stand**. To perform a file system consistency check, type:

```
$$ /stand/fsck /dev/rp0
```

To bring up UNIX, the operator must type **unix<cr>**. The system will come up through **init S** (see *init*(1M)).

This is the appropriate time to do file system backups, and *fsck*(1M) should be executed if it was not executed in the stand-alone section of the boot. One must never operate UNIX with a defective file system.

After successful completion of *fsck*(1M), the operator can bring the system to multi-user operation by executing **init 2**.

Bringing the System Down

The shutdown procedure is designed to gracefully turn off all processes and bring the system back to single user state with all buffers flushed. To do this the operator should execute *shutdown*(1M). If *shutdown* is not successful, use the following sequence of commands:

```
killall
sync
telinit 6
fsck (optional)
```

The system may then be halted by typing the **<ctl>p** and **H<cr>** sequence.

System Dumps

After a system crash, the following procedure should be used to get a system dump on tape.

1. Mount a tape with write ring and bring it on-line.
2. Enter console mode with **<ctl>p**.
3. After the >>> prompt, halt the system with **H<cr>**.
4. Issue the following command sequence, each command followed by **<cr>**:

E R0/N:F	(*Examine R0 thru R15*)
E SP	(*Get the stack pointer for the next command*)
E/V @/N:3F	(*Examine virtual memory beginning at the address from the previous instruction, and continuing for the next 63 locations; i.e., examine the stack*)
S 400	(*Start execution at 400, i.e., dump to tape*)

5. Before returning to UNIX, execute the stand-alone *fsck*(1M).

To read the dump tape into a file for examination by *crash*(1M) or any other debugging program, use *dd*(1). For example, the following will read the dump and create a file named core in the current directory:

```
dd if=/dev/rmt0 of=core bs=16b
```

System Faults

On occasion, the UNIBUS or its devices fail in such a manner as to flood the console with error messages and suspend operations on UNIBUS devices. It may be possible under these conditions to bring the system down gracefully from an internal point-of-view, by inhibiting UNIBUS interrupts and running a normal shutdown. The following sequence can be executed:

<ctl>p	
>>>H	
>>>E 20006004	(*Look at UBA control register*)
>>>D * 1	(*Clear the UBA*)
>>>C	(*Return to UNIX*)

You should now be able to login as **root** and run a normal shutdown sequence. Reboot the system by normal means, ensuring *fsck*(1M) is performed.

INSTALLATION BOOT PROCEDURES

Tape Boot

The floppy disk delivered with the VAX-11/780 does not have tape-boot capability. The user must type in a program to read the first record on tape drive 0. If tape drive 0 is a TE16-type tape drive, use the first program provided below. For a TU78 tape drive, use the second program. Type <cr> at the end of each input line:

TE16 Tape Drive

```
>>>H
>>>U
>>>I

        INIT SEQ DONE

>>>D 20000 20008FD0
>>>D + D0502001
>>>D + 3204A001
>>>D + C003C08F
>>>D + A0D40424
>>>D + 8FD00C
>>>D + C0800000
>>>D + 8F320800
>>>D + 10A0FE00
>>>D + C007D0
>>>D + C039D004
>>>D + 400
>>>S 20000          (Start tape load)

        HALT INST EXECUTED
        HALTED AT 0002002F

>>>S 0              (Execute boot program loaded from tape)
```

TU78 Tape Drive

```
>>>H
>>>U
>>>I

        INIT SEQ DONE

>>>D 20000 20008FD0
>>>D + D4502001
>>>D + 8FD00CA0
>>>D + 80000000
>>>D + 320800C0
>>>D + A0FE008F
>>>D + C004D010
>>>D + 39320404
>>>D + 000400C0
>>>S 20000          (Start tape load)

        HALT INST EXECUTED
        HALTED AT 00020029

>>>S 0              (Execute boot program loaded from tape)
```

From this point the loader initiates a question and answer sequence to control the remainder of the load process.

Disk Boot

The floppy disk delivered with the VAX-11/780 does not have UNIX disk-boot capability. The user must type in the following program to read the first block on disk drive 0. Type <cr> at the end of each line.

>>> H	
>>> LINK	(*Save the following sequence on the floppy*)
	(*The prompt should change to* <<<)
<<< H	*Halt processor*
<<< U	*Unjam the SBI*
<<< I	*Initialize the processor*
<<< D/I 11 20003800	*Register initialization*
<<< D R0 0	*Device type code*
<<< D R1 8	*NEXUS number of MBA in hex*
<<< D R2 0	*drive number*
<<< D R3 0	*drive number*
<<< D R4 0	
<<< D R5 8	*Software boot flags*
<<< D FP 0	*Set "no machine check expected"*
<<< S 20003000	*Start rom program*
<<< WAIT DONE	
<<< E SP	*Show address of working memory +0x200*
<<< L VMB.EXE/S:@	*Load primary bootstrap*
<<< S @	*and start it*
<<< <ctl>C	(*Exit* LINK *mode*)
>>>	

You are now ready to boot UNIX. Each time it is necessary to boot (or reboot) UNIX, simply follow the sequence:

>>> P<cr>	(*Execute the commands saved in floppy link file; the console should echo each command in the file.*)
$$ unix<cr>	(*Load and execute* **/unix**)

UNIX Floppy Update

To update the console floppy for UNIX operation, one must have brought UNIX up by one of the initial-load procedures described above. The following sequence can then be executed.

```
# cd /stand/conflp
# sh update
```

Update prints commentary during the update operation indicating the files that are being replaced or added. Finally, a new table of contents is printed and the available space is indicated.

CONSOLE OPERATION

The following is condensed from Chapter 14 of the *VAX-11/780 Hardware Handbook*, DEC, 1980.

The following are the standard console commands. The most abbreviated form is shown in parentheses.

<ctl>P — Causes console to exit Program I/O mode (talking to the VAX-11/780 program). This does *not* halt the VAX CPU.

<ctl>U — Deletes the current input line.

<del> — Deletes the previous character.

<ctl>C — Interrupts printout.

(HE)LP — Prints "help" file of which this is a part.

(E)XAMINE {address}
: Displays 8-digit hexadecimal address and its contents. See "help" file for qualifiers.

(D)EPOSIT {address} {data}
Enters data to address.

(I)NITIALIZE Initializes CPU.

(U)NJAM Unjams the SBI.

(SH)OW Displays console and CPU state.

(H)ALT Halts execution of VAX CPU instructions.

(S)TART {address}
Initializes CPU, enters address to PC, issues CONTINUE to CPU, and puts console into Program I/O mode.

(C)ONTINUE Starts execution of VAX CPU instructions.

(SE)T (T)ERMINAL (P)ROGRAM
Puts console into Program I/O mode.

@{file} Causes the named floppy file to be printed and executed.

WARNINGS

Only **<ctl>p** can be executed from Program I/O mode. It *does not* stop the VAX CPU from running. Only **HALT** can be executed while the VAX CPU is running and not in Program I/O mode; therefore, the sequence to stop the VAX-11/780 while running UNIX (Program I/O mode) is:

<ctl>p
>>>H<cr>

FILES

/etc/shutdown
/stand/*

SEE ALSO

filesave(1M), fsck(1M), init(1M), shutdown(1M), tapeboot(8).

8

NAME

crash — what to do when the system crashes

DESCRIPTION

This entry gives at least a few clues about how to proceed if the system crashes. It can't pretend to be complete.

How to bring it back up. If the reason for the crash is not evident (see below for guidance on "evident") you may want to try to dump the system if you feel up to debugging. At the moment a dump can be taken only on magtape. With a tape mounted and ready, stop the machine, load address 44(8) (on the PDP-11), 400(16) (on the VAX-11/780; see *780ops*(8)), and start. This should write a copy of all of core on the tape with an EOF mark. Be sure the ring is in, the tape is ready, and the tape is clean and new.

In restarting after a crash, always bring up the system single-user, as specified in *unixboot*(8) as modified for your particular installation. Then perform an *fsck*(1M) on all file systems which could have been in use at the time of the crash. If any serious file system problems are found, they should be repaired. When you are satisfied with the health of your disks, check and set the date if necessary, then come up multi-user.

To even boot UNIX at all, three files (and the directories leading to them) must be intact. First, the initialization program **/etc/init** must be present and executable. If it is not, the CPU will loop in user mode at location 6(8) (PDP-11), 13(16) (VAX-11/780). For *init* to work correctly, **/dev/console** and **/bin/sh** must be present. If either does not exist, the symptom is best described as thrashing. *Init* will go into a *fork/exec* loop trying to create a shell with proper standard input and output.

If you cannot get the system to boot, a runnable system must be obtained from a backup medium. The root file system may then be doctored as a mounted file system as described below. If there are any problems with the root file system, it is probably prudent to go to a backup system to avoid working on a mounted file system.

Repairing disks. The first rule to keep in mind is that an addled disk should be treated gently; it shouldn't be mounted unless necessary, and if it is very valuable yet in quite bad shape, perhaps it should be copied before trying surgery on it. This is an area where experience and informed courage count for much.

Fsck(1M) is adept at diagnosing and repairing file system problems. It first identifies all of the files that contain bad (out of range) blocks or blocks that appear in more than one file. Any such files are then identified by name and *fsck* requests permission to remove them from the file system. Files with bad blocks should be removed. In the case of duplicate blocks, all of the files except the most recently modified should be removed. The contents of the survivor should be checked after the file system is repaired to ensure that it contains the proper data. (Note that running *fsck* with the **−n** option will cause it to report all problems without attempting any repair.)

Fsck will also report on incorrect link counts and will request permission to adjust any that are erroneous. In addition, it will reconnect any files or directories that are allocated but have no file system references to a "lost+found" directory. Finally, if the free list is bad (out of range, missing, or duplicate blocks) *fsck* will, with the operators concurrence, construct a new one.

Why did it crash? UNIX types a message on the console typewriter when it voluntarily crashes. Here is the current list of such messages, with enough information to provide a hope at least of the remedy. The message has the form "panic: ...", possibly accompanied by other information. Left unstated in all cases is the possibility that hardware or software error produced the message in some unexpected way.

blkdev
: The *getblk* routine was called with a nonexistent major device as argument. Definitely hardware or software error.

devtab
: Null device table entry for the major device used as argument to *getblk*. Definitely hardware or software error.

iinit
: An I/O error reading the super-block for the root file system during initialization.

no fs
: A device has disappeared from the mounted-device table. Definitely hardware or software error.

no imt
: Like "no fs", but produced elsewhere.

no clock
: During initialization, neither the line nor programmable clock was found to exist.

I/O error in swap
: An unrecoverable I/O error during a swap. Really shouldn't be a panic, but it is hard to fix.

out of swap space
: A program needs to be swapped out, and there is no more swap space. It has to be increased. This really shouldn't be a panic, but there is no easy fix.

trap
: An unexpected trap has occurred within the system. This is accompanied by three numbers: a "ka6", which is the contents of the segmentation register for the area in which the system's stack is kept; "aps", which is the location where the hardware stored the program status word during the trap; and a "trap type" which encodes which trap occurred. The trap types are:

PDP-11:

0	bus error
1	illegal instruction
2	BPT/trace
3	IOT
4	power fail
5	EMT
6	recursive system call (TRAP instruction)
7	11/70 cache parity, or programmed interrupt
8	floating point trap
9	segmentation violation

VAX-11/780:

0	reserved addressing fault
1	illegal instruction
2	BPT instruction trap
3	XFC instruction trap

8

4 reserved operand fault
5 recursive system call (CHMK instruction)
6 floating point trap
7 software level 1 (reschedule) trap
8 segmentation violation
9 protection fault
10 trace trap
11 compatibility mode fault

In some of these cases it is possible for octal 40 to be added into the trap type; this indicates that the processor was in user mode when the trap occurred. If you wish to examine the stack after such a trap, either dump the system, or use the console switches to examine core; the required address mapping is described below.

Interpreting dumps. All file system problems should be taken care of before attempting to look at dumps. The dump should be read into the file **/usr/tmp/core**; *cp*(1) will do. At this point, you should execute *ps −el −c /usr/tmp/core* and *who* to print the process table and the users who were on at the time of the crash.

Additional information for the PDP-11. You should dump (*adb*(1)) the first 30 bytes of **/usr/tmp/core**. Starting at location 4, the registers R0, R1, R2, R3, R4, R5, SP and KDSA6 (KISA6 for 11/40s) are stored. If the dump had to be restarted, R0 will not be correct. Next, take the value of KA6 (location 22(8) in the dump) multiplied by 100(8) and dump 2000(8) bytes starting from there. This is the per-process data associated with the process running at the time of the crash. Relabel the addresses 140000 to 141776. R5 is C's frame or display pointer. Stored at (R5) is the old R5 pointing to the previous stack frame. At (R5)+2 is the saved PC of the calling procedure. Trace this calling chain until you obtain an R5 value of 141756, which is where the user's R5 is stored. If the chain is broken, you have to look for a plausible R5, PC pair and continue from there. Each PC should be looked up in the system's name list using *adb*(1) and its : command, to get a reverse calling order. In most cases this procedure will give an idea of what is wrong. A more complete discussion of system debugging is impossible.

SEE ALSO

adb(1), fsck(1M), 780ops(8), unixboot(8).

NAME

crash − what to do when the system crashes

DESCRIPTION

This entry gives at least a few clues about how to proceed if the system crashes. It can't pretend to be complete.

How to bring it back up. If UNIX voluntarily crashed, it will take a memory dump to disk and attempt to reboot itself. If the system appears to be "hung" for unknown reasons, a memory dump should be taken. See *3B20ops*(8) for this procedure.

After a crash, it is imperative that the file systems be checked for consistency. Perform an *fsck*(1M) on all file systems that were in use at the time of the crash. If any serious file system problems are found, they should be repaired. When you are satisfied with the health of your disks, check and set the date if necessary, then come up multi-user.

If it will not boot. There are many reasons why UNIX might not boot, including: hardware problems, an improperly configured system, a corrupted boot section on disk, or a corrupted root file system. Most boot failures will cause a processor recovery message (PRM) to be displayed on the system console. See *prm*(8) for a list of failure messages. If **/dev/console** or **/bin/sh** cannot be accessed, the system will just appear to "hang" without any failure message. If *lboot* cannot be loaded into memory by *Microboot* (indicated by PRM's starting with **F0**), suspect hardware problems or a corrupted boot section on disk. If UNIX runs and then "hangs" or "panics", suspect an improperly configured system or a corrupted root file system. As a general strategy, try the following in order: boot an older system version and/or minimally configured, boot from the back-up root file system, boot from another disk pack or the secondary disk drive, have the hardware checked out.

Repairing disks. The first rule to keep in mind is that an addled disk should be treated gently; it shouldn't be mounted unless necessary, and if it is very valuable yet in quite bad shape, perhaps it should be copied before trying surgery on it. This is an area where experience and informed courage count for much.

8

Fsck(1M) is adept at diagnosing and repairing file system problems. It first identifies all of the files that contain bad (out of range) blocks or blocks that appear in more than one file. Any such files are then identified by name and *fsck* requests permission to remove them from the file system. Files with bad blocks should be removed. In the case of duplicate blocks, all of the files except the most recently modified should be removed. The contents of the survivor should be checked after the file system is repaired to ensure that it contains the proper data. (Note that running *fsck* with the **−n** option will cause it to report all problems without attempting any repair.)

Fsck will also report on incorrect link counts and will request permission to adjust any that are erroneous. In addition, it will reconnect any files or directories that are allocated but have no file system references to a "lost+found" directory. Finally, if the free list is bad (out of range, missing, or duplicate blocks) *fsck* will, with the operators concurrence, construct a new one.

Why did it crash? All messages printed by UNIX are saved in a circular buffer contained in memory starting at the symbol **putbuf**. These messages can be looked at by examining the memory dump using *crash*(1M).

UNIX prints a message of the form "panic: ..." when it voluntarily crashes. Here is an incomplete list of such messages.

cannot mount root
: An I/O error occurred while trying to mount the root file system. Most likely caused by an improperly configured system.

cannot allocate system buffers
: Too many "buffers" have been configured into the system.

cannot allocate character buffers
: Too many "clists" have been configured into the system.

i/o error in swap
: A hardware error occurred while swapping a process.

trap
: An unexpected hardware trap has occurred. This message is accompanied by the physical page addresses of the UAREA of the last running process, the interrupt stack pointer, the program counter, the processor status word, and a short message describing the type of trap:

  Protection violation — an attempt to access memory in a way that is not permitted, e.g. writing a read-only segment.

  Segmentation violation — an attempt to access memory not within the kernel's address space.

  Addressing Alignment Error — an attempt to access a data object at an improper boundary, e.g. a word at an odd address.

Other "panics" are possible but in almost all cases indicate hardware problems or that UNIX has been tampered with.

SEE ALSO

crash(1M), fsck(1M), 3B20ops(8), 3B20boot(8), prm(8).

8

NAME

diskboot — disk bootstrap programs

DESCRIPTION

There are several programs available to accomplish bootstraps off of a variety of disks. These programs reside in the directory **/stand**.

The program must be located in block 0 of the disk pack. The space available for the program is thus only one block (256 words) which severely constrains the amount of error handling. Block 0 is unused by the UNIX file system, so this does not affect normal file system operation. To boot, the program must be read into memory starting at address 0 and started at address 0. This may be accomplished by standard DEC ROM bootstraps, special ROM bootstraps, or manual procedures.

After initial load, the program relocates itself to high core as specified when assembled (typically 24K words, maximum of 28K). Next, memory below the program is cleared and the prompt # is typed on the console. A one digit field specifying the disk drive is expected. For example, 2 would correspond to drive 2, starting at cylinder 0. The last word in the boot block contains a cylinder offset, initially zero, which may be changed to access another section of the disk pack. No error checking is done on this field; invalid data will cause unpredictable results. Also, there is no error checking on disk reads.

After the file system select, the program prompts with =. The user must then enter the UNIX path name of the desired file. The # character will erase the last character typed, the @ character will kill the entire line, and A through Z is translated to **a** through **z**. Also, carriage return (CR) is mapped into line-feed (LF) on input, and LF is output as CR-LF. The upper-case to lower-case conversion is used to handle upper-case-only terminals such as the TELETYPE® Model 33 or the DEC LA30. Therefore, a file name with upper case characters cannot be booted using this procedure.

After the name has been completely entered by typing CR or LF, the program searches the file system specified for the path name. Note, the path name may be any valid UNIX file system path name. If the file does not exist, or if the file is a directory or special file, the bootstrap starts over and prompts with #. Otherwise, the file is read into memory starting at address 0. If address 0 contains 000 407, a UNIX **a.out** program is assumed and the first 8 words are stripped off by relocating the loaded program toward address 0. Finally, a jump to address 0 is done by executing **jsr pc,*$0**.

FILES

/usr/src/stand source directory

SEE ALSO

a.out(4), fs(4), tapeboot(8), unixboot(8).

8

NAME
eai — 3B20S emergency action interface

DESCRIPTION
The functions of the 3B20S Emergency Action Interface (EAI) on the system console are described below.

Function Keys
Four special function keys, labeled EA DISP, NORM DISP, CMD/MSG and ALM RLS are on the keyboard of the system console:

EA DISP — This key starts the emergency action mode and causes the EAI display, consisting of status indicators and a menu of commands, to appear on the top half of the screen. Status indicators are updated every two seconds or as changes occur. If UNIX is running, then the bottom half of the screen may be used as a login terminal and will scroll without affecting the EAI display. If the EAI display is already present, then depressing this key will cause the screen to be updated.

NORM DISP — This key ends the EAI mode, erases the EAI display and leaves the screen blank. The full screen is now available as a UNIX login terminal.

CMD/MSG — This key toggles the cursor between the command entry area and the UNIX portion (bottom half message section) of the screen. EAI commands can be entered only when the cursor is in the command entry position (next to CMD:). This key is effective only when the screen is in EAI mode.

ALM RLS — This key currently performs no function.

Status Indicators
MTTY — A single digit incremented once every two seconds that indicates the ability of the Maintenance TTY Peripheral Controller (MTTYPC) to update the EAI display.

3BCC — A series of five indicators describing the current state of the 3B20S processor as seen by the EAI.

- ACT — The 3B20S processor is on-line (it has I/O access).
- RUN — The 3B20S is processing instructions (not stopped or halted).
- FONL — The 3B20S is forced to be the on-line processor (I/O is allowed) and the Diagnostic Processor (DP) cannot gain I/O access.
- FOFL — The 3B20S is forced to be the off-line processor (I/O is inhibited) and cannot gain I/O access.
- RCVRY — 3B20S microcode has signaled the start of processor recovery.

DPCC — A series of five indicators describing the current state of the DP as seen by DP microcode. These are the same as the indicators for 3BCC above, with the role of the 3B20S and the DP interchanged.

SCCS — currently unused.

EAI — A single indicator with three possible states describing the state of the link between the EAI and the MTTYPC.

8

ASW All Seems Well.

ERR The EAI can communicate with the MTTYPC but there are problems.

OOS The EAI is unable to communicate with the MTTYPC.

DPI Same as EAI, but indicates the state of the link between the DP and the MTTYPC.

TIMEOUT This appears only if the EAI has not received a low priority Processor Recovery Message (PRM) within a seventy second time period. This is an indication of lack of sanity of the 3B20S processor.

3BPRM Processor Recovery Message (PRM) from the 3B20S processor.

DPPRM PRM from the Diagnostic Processor.

Commands

Commands can be entered only when the cursor is positioned in the top left-hand corner of the screen next to **CMD:**. A command is entered by keying in the number associated with the command by the EAI display menu. Commands may be terminated either by a carriage return or by an exclamation point (**!**). A character may be erased by a backspace or an underscore. A line may be killed with a dollar sign (**$**). When a line is entered, the EAI responds with **OK** for a successfully entered command, or **NG** for an invalid command.

Commands **60−65** cause immediate action when they are entered. Commands **60−62** refer to the Diagnostic Processor (DP), which will be supplied in the future as an option.

60 3B−FONL Forces the 3B20S processor on-line, allowing the 3B20S processor I/O access and inhibiting the DP I/O access. Any diagnostics that were running on the DP are aborted.

61 DP−FONL Forces the 3B20S processor off-line, inhibiting the 3B20S processor I/O access and allowing the DP I/O access. If UNIX was running on the 3B20S, then it is aborted. The DP executes IOP diagnostics, reads a diagnostic tape and then establishes an interface to the MTTYPC in order to accept diagnostic commands.

62 DP−INIT Initializes the DP.

63 CFT−INIT Currently not implemented.

64 PRM−DUMP Currently not implemented.

65 CLR−EAI Resets all functions on the EAI display and zeroes the 3BPRM and DPPRM fields. All SET/CLR functions are reset to CLR.

Commands **70−73** and **76−93** set or clear options to be used during and after the next initialization, disk boot, disk dump or load from tape. They cause no immediate action. Commands **74** and **75** affect only the EAI display and not the UNIX software. In each pair below, the even number sets the option and is displayed as **SET**, and the odd number clears the option and is displayed as **CLR**. The description below represents the option that is selected when the even command of the pair is entered. Unless explicitly noted otherwise, the corresponding odd command undoes this option.

70–71 SEC–DISK
: Causes moving head disk 1 on disk file controller 0 to be used as the boot device or the disk to be loaded by LDTAPE (see **98** below). Clearing this option causes moving head disk 0 on disk file controller 0 to be used.

72–73 INH–TIMER
: Inhibits automatic recovery when a sanity timeout occurs.

74–75 PRM–TRAP
: Freezes the next failing PRM on the EAI display.

76–77 PARAMETER
: Sets a parameter which is used to determine the action taken by INIT (see **95** below) or by automatic recovery after a failure. When the **76** command is entered, the user is prompted for a single character parameter value on the command entry line in the EAI display. After the character is entered, it will be displayed next to the word PARAMETER on the display. Possible values for the parameter are:

h causes the system to idle.

H causes the system to halt. The system will reboot if the sanity timer is not inhibited.

d causes the system to dump a memory image to disk and then reboot.

D causes the system to dump a memory image to disk and then idle.

r causes the system to reboot.

If the parameter is cleared or if it is set to a value not mentioned above, then the default action will be reboot the processor.

80–81 PROMPT–UNIX
: If set, this causes the disk bootstrap program to prompt the user for the name of the program to be booted. If clear, **/unix** will be chosen as the program to be booted. See *3B20boot*(8).

82–83 BACKUP–ROOT
: Causes the disk bootstrap program to find the program to be booted on the backup root file system. If clear, the normal root file system is used (see *dsk*(7)).

84–85 MIN–CONFIG
: Causes UNIX to bring only the boot device, the system console and a tape drive into service and only the first megabyte of main memory will be used.

86–87 INH–HDW–CHK
: Causes UNIX to disable refresh and correctable main store parity error detection.

88–89 INH–SFT–CHK
: Currently unused.

90–91 INH–ERR–INT
: Currently unused.

92−93 INH−CACHE
Disables the use of cache memory.

Commands **95−99** cause immediate action which is affected by options **70−73** and **76−93** above. If these commands fail, they will output PRMs in the 3BPRM field of the display. An explanation of failure PRMs is found in *prm*(8).

95 INIT	Causes different action depending on the parameter value set by command **76**.
96 BOOT	Causes a disk bootstrap. See *3B20boot*(8).
97 DUMP	Causes a memory image of the operating system to be dumped to disk followed by a disk bootstrap.
98 LDTAPE	Causes a disk to be loaded from tape. See *ldtape*(8).
99 HALT	Causes UNIX to idle.

SEE ALSO

dsk(7), 3B20boot(8), ldtape(8), prm(8).
UNIX System Operator's Guide.

8

NAME

ldtape — load disk from tape procedures

DESCRIPTION

Ldft is a program loaded from tape into memory and executed in response to the LDTAPE command on the EAI page of the console. (See *eai*(8) and Setting up UNIX in the *UNIX System Administrator's Guide* for further details on the use of the console and setting up UNIX.) *Ldft* is intended for use only to create a disk pack in a proper format when a new release of UNIX is installed.

To run *ldft*, mount the disk pack that is to be loaded on moving head disk drive 0 or 1, mount LDFT tape number 0 on tape drive unit 0, and issue the LDTAPE command on the EAI page at the console.

Ldft will look at the SEC−DISK flag and the PARAMETER field on the EAI when the LDTAPE command is issued. The disk to be loaded is specified by the SEC−DISK flag. Moving head disk 0 is used if the flag is clear. Moving head disk 1 is used if the flag is set.

If the PARAMETER field contains an f or an F, *ldft* will format the disk in the specified drive before continuing. (The format should be done unless it is known that the disk pack has already been formatted.)

Ldft will then rewind the tape and issue a success Processor Recovery Message (PRM) asking for LDFT tape number 1 to be mounted. (See the diagnostics section below.) Mount the next tape and issue another LDTAPE command. *Ldft* will read the tape copying data to disk as it is read. When the end of tape is found, *ldft* will rewind the tape. If another LDFT tape is expected, *ldft* will issue a success PRM requesting the next tape. When the last LDFT tape has been read, *ldft* will issue a success PRM similar to a request for the next tape with the tape number field containing BBBB. When this point is reached, the disk has been loaded and can be booted. See *3B20boot*(8) for boot procedures.

SEE ALSO

3B20boot(8), eai(8), prm(8).
Setting up UNIX in the *UNIX System Administrator's Guide*.

DIAGNOSTICS

The following code words are used in success PRM's from *ldft*:

Code	*Meaning*
cyls	50 cylinder disk section number
sect	Tape section number
tape	Tape reel number

The following success PRM's are generated by *ldft*:

PRM	*Meaning*
E100 7000 0000 0000	IOP, tape, and DFC in service
E100 7100 *tape* 0000	Request to mount tape
E100 7500 *sect* *tape*	Section header read successfully
EF00 0000 *cyls* 0000	Starting disk section format

Failure PRM's are listed in *prm*(8).

NAME

mk — how to remake the system and commands

DESCRIPTION

All source for UNIX is in a source tree distributed in the directory **/usr/src**. This includes source for the operating system, libraries, commands, miscellaneous files necessary to the running system, and procedures to create everything from this source.

The top level consists of the directories **cmd**, **lib**, **uts**, **head**, and **stand** as well as commands to remake each of these "directories". These commands are named *:mk*, which remakes everything, and *:mk***dir** where **dir** is the directory to be recreated. Each recreation command will make all or part of the piece; over which it has control. *:mk* will run each of these commands and thus recreate the whole system.

The **lib** directory contains libraries used when loading user programs. The largest and most important of these is the C library. All libraries are in sub-directories and are created by a makefile or runcom. A runcom is a Shell command procedure used specifically to remake a piece of the system. *:mklib* will rebuild the libraries that are given as arguments. The argument ***** will cause it to remake all libraries.

The **head** directory contains the header files, usually found in **/usr/include** on the running system. *:mkhead* will install those header files that are given as arguments. The argument ***** will cause it to install all header files.

The **uts** directory contains the source for the UNIX operating system. *:mkuts* (no arguments) invokes a series of makefiles that will recreate the operating system.

The **stand** directory contains stand-alone commands and boot programs. *:mkstand* will rebuild and install these programs.

The **cmd** directory contains files and directories. *:mkcmd* transforms source into a command based upon its suffix (**.l**, **.y**, **.c**, **.s**, **.sh**), or its makefile (see *make*(1)) or runcom. A directory is assumed to have a makefile or a runcom that will take care of creating everything associated with that directory and its sub-directories. Makefiles and runcoms are named *command***.mk** and *command***.rc** respectively.

:mkcmd will recreate commands based upon a makefile or runcom if one of them exists; alternatively commands are recreated in a standard way based on the suffix of the source file. All commands requiring more than one file of source are grouped in sub-directories, and must have a makefile or a runcom. C programs (**.c**) are compiled by the C compiler and loaded stripped with shared text. Assembly language programs (**.s**) are assembled with **/usr/include/sys.s** which contains the system call definitions. Yacc programs (**.y**) and lex programs (**.l**) are processed by *yacc*(1) and *lex*(1) respectively before C compilation. Shell programs (**.sh**) are copied to create the command. Each of these operations leaves a command in **./cmd** which is then installed by using **/etc/install**.

The arguments to *:mkcmd* are either command names, or subsystem names. The subsystems distributed with UNIX are: **acct**, **graf**, **rje**, **sccs**, and **text**. Prefacing the *:mkcmd* instruction with an assignment to the Shell variable **$ARGS** will cause the indicated components of the subsystem to be rebuilt.

The entire **sccs** subsystem can be rebuilt by:

```
/usr/src/:mkcmd  sccs
```

while the *delta* component of **sccs** can be rebuilt by:

ARGS="delta" /usr/src/:mkcmd sccs

The *log* command, which is a part of the **stat** package, which is itself a part of the **graf** package, can be rebuilt by:

ARGS="stat log" /usr/src/:mkcmd graf

The argument ***** will cause all commands and subsystems to be rebuilt.

Makefiles, both in **./cmd** and in sub-directories, have a standard format. In particular *:mkcmd* depends on there being entries for *install* and *clobber*. *Install* should cause everything over which the makefile has jurisdiction to be made and installed by **/etc/install.** *Clobber* should cause a complete cleanup of all unnecessary files resulting from the previous invocation.

Most of the runcoms in **./cmd** (as opposed to sub-directories) relate in particular to a need for separated instruction and data (I and D) space.

In the past, dependency on the C library routine *ctime*(3C) was also important. *Ctime* had to be modified for all systems located outside of the eastern time zone, and all commands that referenced it had to be recompiled. *Ctime* has been rewritten to check the environment (see *environ*(5)) for the time zone. This results in time zone conversions possible on a per-process basis. **/etc/profile** sets the initial environment for each user, and **/etc/rc** sets it for certain system daemons. These two programs are the only ones which must be modified outside of the eastern time zone.

An effort has been made to separate the creation of a command from source, and its installation on the running system. The command **/etc/install** is used by *:mkcmd* and most makefiles to install commands in the proper place on the running system. The use of install allows maximum flexibility in the administration of the system. Install makes very few assumptions about where a command is located, who owns it, and what modes are in effect. All assumptions may be overridden on invocation of the command, or more permanently by redefining a few variables in install. The object is to install a new version of a command in the same place, with the same attributes as the prior version.

In addition, the use of a separate command to perform installation allows for the creation of test systems in other than standard places, easy movement of commands to balance load, and independent maintenance of makefiles. The minimization of makefiles in most cases, and the site independence of the others should greatly reduce the necessary maintenance, and allow makefiles to be considered part of the standard source.

SEE ALSO

install(1M), make(1).

NAME

prm — 3B20S Processor Recovery Messages

DESCRIPTION

This manual page describes the Processor Recovery Messages (PRM's) produced by the 3B20S processor.

SEE ALSO

3B20boot(8), eai(8), ldtape(8).

DIAGNOSTICS

These code words appear in the following list of PRM's:

Code	*Meaning*
cc	DFC job completion code
dcod	3/6 code for disk device
dfca	High order 16 bits of DFC 2nd status word
dfcb	Low order 16 bits of DFC 2nd status word
dmpct	Number of tries necessary to get a dump
flag	Root or backup root filesystem flag
resp	IOP command completion response
sect	Tape section number
stat	Channel status
tape	Tape reel number
tcod	3/6 code for tape device

Non-error PRM's:

PRM	*Meaning*
E000 0000 0000 0000	MRF entered
E000 0100 0000 0000	MRF successfully exited to PINIT
E000 0200 0000 0000	MRF successfully exited to STOP & SWITCH
E000 0300 0000 0000	Successful exit from microboot
E000 0400 0000 0000	MRF exited to POWER CLEAR halt
E000 0500 0000 0000	Pump successfully exited to PINIT
E000 0600 0000 0000	Successful exit from tapeboot ucode
E000 0000 0070 *dmpct*	Completed dump

Failure PRM's:

PRM	*Meaning*
F000 0800 0000 0000	Microboot—lboot job—channel init failed
F000 0900 0000 0000	Microboot—vtoc job—dma setup failed
F000 0A00 0000 0000	Microboot—vtoc job—bic init failed
F000 0B00 0000 0000	Microboot—vtoc job—pic init failed
F000 0C00 0000 0000	Microboot—vtoc job—boot command failed
F000 0D00 0000 0000	Microboot—vtoc job—disk job timed out
F000 0E00 0000 0000	Microboot—vtoc job—disk job error
F000 1000 0000 0000	Microboot—ucode pump—channel init failed
F000 1100 0000 0000	Microboot—ucode pump—dma setup failed
F000 1200 0000 0000	Microboot—ucode pump—bic init failed
F000 1300 0000 0000	Microboot—ucode pump—pic init failed
F000 1400 0000 0000	Microboot—ucode pump—boot command failed
F000 1500 0000 0000	Microboot—ucode pump—disk job timed out
F000 1600 0000 0000	Microboot—ucode pump—disk job error
F000 1700 0000 0000	Microboot—ucode pump—error in ucode file
F000 1800 0000 0000	Microboot—lboot job—channel init failed
F000 1900 0000 0000	Microboot—lboot job—dma setup failed
F000 1A00 0000 0000	Microboot—lboot job—bic init failed

F000 1B00 0000 0000	Microboot—lboot job—pic init failed
F000 1C00 0000 0000	Microboot—lboot job—boot command failed
F000 1D00 0000 0000	Microboot—lboot job—disk job timed out
F000 1E00 0000 0000	Microboot—lboot job—disk job error
F000 2100 0000 0000	LDFT—channel or bic init failed
F000 2200 0000 0000	LDFT—pic init failed
F000 2300 0000 0000	LDFT—dma setup for tape failed
F000 2400 0000 0000	LDFT—quick sysgen for tape failed
F000 2500 0000 0000	LDFT—tape rewind failed
F000 2600 0000 0000	LDFT—tape read of ucode header failed
F000 2700 0000 0000	LDFT—bad ucode header
F000 2800 0000 0000	LDFT—tape read of ucode failed
F000 2900 0000 0000	LDFT—invalid ucode file
F000 2A00 0000 0000	LDFT—tape read of LDFT header failed
F000 2B00 0000 0000	LDFT—invalid LDFT header
F000 2C00 0000 0000	LDFT—tape read of LDFT failed
F000 2D00 0000 0000	LDFT—bic init for disk failed
F000 2E00 0000 0000	LDFT—pic init for disk failed
F100 0100 0000 *stat*	Can't initialize DMAC
F100 0200 0000 *stat*	Can't initialize DSCH
F100 0300 0000 *stat*	Can't enable DMA access to MAS
F100 1000 0000 *stat*	Can't write DMAC ram
F100 1100 0000 0000	Can't clear DBS
F100 1200 0000 *stat*	Can't initialize BIC
F100 1300 0000 *stat*	Can't enable BIC/PIC interface
F100 1400 0000 *stat*	Can't enable BIC interrupts
F100 1500 0000 *stat*	Can't sysgen the DFC
F100 1600 0000 *stat*	Can't bring the DFC into service
F100 2000 0000 *stat*	Can't bring the disk into service
F100 3100 0000 *stat*	Can't read DFC status
F100 3200 0000 *stat*	Can't read DFC status
F100 4000 0000 *stat*	Can't reset BIC interrupt
F100 4100 0000 *stat*	Can't send status command to DFC
F100 4200 0000 *stat*	Can't read 1st DFC status word
F100 4300 0000 *stat*	Can't read 2nd DFC status word
F100 44*cc* *dfca* *dfcb*	Bad DFC response
F100 4500 0000 *stat*	Can't enable BIC interrupt
F100 5000 0000 *stat*	Can't write DMAC ram
F100 5001 0000 *stat*	Can't clear DDBS
F100 5002 0000 *stat*	Can't send BIC command
F100 5003 0000 *stat*	Can't send BIC command
F100 5004 0000 *stat*	Can't enable IOP interrupts
F100 5005 0000 *stat*	Can't send IOP sysgen
F100 5006 0000 0000	IOP sysgen failed
F100 5010 0000 0000	IOP interrupt error
F100 5011 0000 *stat*	Can't sense DDBS status
F100 5012 0000 *stat*	DDBS status error
F100 5013 0000 *stat*	Can't reset BIC interrupt
F100 5020 0000 *stat*	Can't sense BIC status
F100 5021 0000 *stat*	Can't sense BIC status
F100 5022 0000 *stat*	BIC status error

8

F100 5023 0000 *stat*	Can't send PIC command
F100 5030 0000 *resp*	Tape read failed
F100 5040 0000 *resp*	Tape rewind failed
F100 5050 0000 0000	IOP response queue empty
F100 5060 0000 *resp*	PC community configure failed
F100 5061 0000 *resp*	Clear PC micro failed
F100 5062 0000 *resp*	Console failed to come into service
F100 5063 0000 *resp*	Console pump failed
F100 5064 0000 *resp*	Console exec failed
F100 5065 0000 *resp*	Console restore failed
F100 5066 0000 *resp*	Console connect failed
F100 5070 0000 *resp*	Console I/O failed
F100 5080 0000 *resp*	PC community failed to come into service
F100 5081 0000 *resp*	Clear PC micro failed
F100 5082 0000 *resp*	Tape PC failed to come into service
F100 7200 *tcod* 0000	Bad 3/6 code for tape from microboot
F100 7300 *dcod* 0000	Bad 3/6 code for disk from microboot
F100 7400 *sect* *tape*	Bad tape section header
F100 8000 0000 *flag*	VTOC has no entry for filesystem
F100 C000 0000 0000	File size too big
0000 0000 DDDD 0002	Dump can't initialize DSCH
0000 0000 DDDD 0003	Dump can't initialize DMA
0000 0000 DDDD 0005	Dump can't enable DMA interupts
0000 0000 DDDD 0006	Dump can't load DMAC ram
0000 0000 DDDD 0007	Dump can't clear dual bus selector
0000 0000 DDDD 0008	Dump can't initialize BIC
0000 0000 DDDD 0009	Dump can't enable BIC interface
0000 0000 DDDD 0010	Dump can't enable device interupts
0000 0000 DDDD 0011	Dump can't sysgen the DFC
0000 0000 DDDD 0012	Dump can't bring DFC into service
0000 0000 DDDD 0014	Dump can't bring disk into service
0000 0000 DDDD 0015	Dump can't reset BIC interupts
0000 0000 DDDD 0016	Dump can't get job states
0000 0000 DDDD 0017	Dump can't job completion word
0000 0000 DDDD 0018	Dump can't get job error word
0000 0000 DDDD 0019	Dump can't get job error word
0000 0000 DDDD 0020	Dump can't enable BIC interupts
0000 0000 DDDD 0021	Dump can't send job pending command
F200 DEAD DEAD DEAD	Panic in UNIX
F300 FFFF FFFF FFFF	UNIX can't execute /etc/init

8

NAME

rje — RJE (Remote Job Entry) to IBM

SYNOPSIS

/usr/rje/rjeinit
/usr/rje/rjehalt

DESCRIPTION

RJE is the communal name for a collection of programs and a file organization that allows a UNIX system, equipped with the appropriate hardware and associated Virtual Protocol Machine (VPM) software, to communicate with IBM's Job Entry Subsystems by mimicking an IBM 360 remote multileaving work station.

Implementation.

RJE is initiated by the command *rjeinit* and is terminated gracefully by the command *rjehalt*. While active, RJE runs in the background and requires no human supervision. It quietly transmits, to the IBM system, jobs that have been queued by the *send*(1C) command, and operator requests that have been entered by the *rjestat*(1C) command. It receives, from the IBM system, print and punch data sets and message output. It enters the data sets into the proper UNIX directory and notifies the appropriate user of their arrival. It scans the message output to maintain a record on each of its jobs. It also makes these messages available for public inspection, so that *rjestat*(1C), in particular, may extract responses.

Unless otherwise specified, all files and commands described below reside in directory **/usr/rje** (first exceptions: *send* and *rjestat*).

There are two sources of data to be transmitted by RJE from UNIX to an IBM System/370. In both cases, the data is organized as files in the **/usr/rje/squeue** directory. The first are files named **co*** which are created by the enquiry command *rjestat*(1C). The second source, containing the bulk of the data, are files named **rd*** or **sq*** which have been created by *send* and queued by the program *rjeqer*. On completion of processing *send* invokes *rjeqer*. *Rjeqer* and *rjestat* inform the program *rjexmit* that a file has been queued via the file **joblog**. Upon successful transmission of the data to the IBM machine, *rjexmit* removes the queued file. As files are transmitted and received, the program *rjedisp* writes an entry containing the date, time, file name, logname, and number of records in the file **acctlog**, if it exists. This file can be used for local logging or accounting information, but is not used elsewhere by RJE. The use of this information is up to the RJE administrator.

Each time *rjeinit* is invoked, the **joblog** file is truncated and recreated from the contents of the **/usr/rje/squeue** directory. During this time, *rjeinit* prevents simultaneous updating of the **joblog** file.

Output from the IBM system is classified as either a print data set, a punch data set, or message output. Print output is converted to an ASCII text file, with standard tabs. Form feeds are suppressed, but the last line of each page is distinguished by the presence of an extraneous trailing space. Punch output is converted to *pnch*(4) format. This classification and both conversions occur as the output is received. Files are moved or copied into the appropriate user's directory and assigned the name **prnt*** or **pnch***, respectively, or placed into user directories under user-specified names, or used as input to programs to be automatically executed, as specified by the user. This process is driven by the "usr=..." specification. RJE retains ownership of these files and permits read-only access to them. Message output is digested by RJE immediately and is not retained.

8

A record is maintained for each job that passes through RJE. Identifying information is extracted contextually from files transmitted to and received from the IBM system. This information is stored and used by the *rjedisp* program for IBM job acknowledgements and delivery of output files.

The IBM system automatically returns an acknowledgement message for each job it receives. Other status messages are returned in response to enquiries entered by users. All messages received by RJE are appended to the **resp** file. The **resp** file is automatically truncated when it reaches 70,000 bytes. Each enquiry is preceded and followed by an identification card image of the form "$UX<*process id*>". The IBM system will echo this back as an illegal command. The appearance of process ids in the response stream permits responses to be passed on to the proper users.

While it is active, RJE occupies at least the three process slots that are appropriated by *rjeinit*. These slots are used to run *rjexmit*, the transmitter, *rjerecv*, the receiver, and *rjedisp*, the dispatcher. These three processes are connected by pipes. The function of each is as follows:

rjexmit Cycles repetitively, looking for data to transmit to the IBM system. After transmission, *rjexmit* passes an event notice to *rjedisp*. If *rjexmit* encounters a **stop** file, (created by *rjehalt*), it exits normally. In the case of error termination, *rjexmit* reboots RJE by executing *rjeinit*.

rjerecv Cycles repetitively, looking for data returning from the IBM machine. Upon receipt of data, *rjerecv* notifies either *rjexmit* or *rjedisp* of the event (transfer information is sometimes passed to *rjexmit*). *Rjerecv* exits normally at the first appropriate moment when it encounters the file **stop**, or exits reluctantly when it encounters a run of errors.

rjedisp Follows up event notices by directing output files, updating records, and notifying users. *Rjedisp* references the system files **/etc/passwd** and **/etc/utmp** to correlate user names, numeric ids, and terminals. Termination of *rjerecv* causes *rjedisp* to exit also.

Rjeinit has the capability of *dialing* any remote IBM system with the proper hardware and software configuration.

8

Most RJE files and directories are protected from unauthorized tampering. The exception is the **spool** directory. It is used by *send*(1C) to create temporary files in the correct file system. *Rjeqer* and *rjestat*(1C), the user's interfaces to RJE, operate in *setuid* mode to contribute the necessary permission modes.

Administration.

Some minimal oversight of each RJE subsystem is required. The RJE mailbox should be inspected and cleaned out periodically. The **job** directory should also be checked. The only files placed there are output files whose destination file systems are out of space. Users should be given a short period of time (say, a day or two), and then these files should be removed.

The configuration table **/usr/rje/lines** is accessed by all components of RJE. Each line of the table (maximum of 8) defines an RJE connection. Its seven columns may be labeled *host*, *system*, *directory*, *prefix*, *device*, *peripherals* and *parameters*. These columns are described as follows:

host

The name of a remote IBM computer (e.g., **A B C**). This string can be up to 5 characters.

system
The name of a UNIX system. This name should be the same as the system name from *uname*(1).

directory
This is the directory name of the servicing RJE subsystem (e.g., **/usr/rje1**).

prefix
This is the string prefixed (redundantly) to several crucial files and programs in **directory** (e.g., **rje1**, **rje2**, **rje3**).

device
This is the name of the controlling VPM device, with **/dev/** excised.

peripherals
This field contains information on the logical devices (readers, printers, punches) used by RJE. Each subfield is separated by :, and is described as follows:

(1) Number of logical readers.
(2) Number of logical printers.
(3) Number of logical punches.

Note: the number of peripherals specified for an RJE subsystem **must** agree with the number of peripherals which have been described on the remote machine for that line.

parameters
This field contains information on the type of connection to make. Each subfield is separated by :. Any or all fields may be omitted; however, the fields are positional. All but trailing delimiters must be present. For example, in

1200:512:::9-555-1212

subfields 3 and 4 are missing, but the delimiters are present. Each subfield is defined as follows:

(1) **space**
This subfield specifies the amount of space (S) in blocks that RJE tries to maintain on file systems it touches. The default is 0 blocks. *Send* will not submit jobs and *rjeinit* issues a warning when less than $1.5S$ blocks are available; *rjerecv* stops accepting output from the host when the capacity falls to S blocks; RJE becomes dormant, until conditions improve. If the space on the file system specified by the user on the "usr=" card would be depleted to a point below S, the file will be put in the **job** subdirectory of the connection's home directory, rather than in the place that the user requested.

(2) **size**
This subfield specifies the size in blocks of the largest file that can be accepted from the host without truncation taking place. The default is no truncation.

(3) **badjobs**
This subfield specifies what to do with undeliverable returning jobs. If an output file is undeliverable for any reason other than file system space limitations (e.g., missing or invalid "usr=" card) and this subfield contains the letter **y**, the output will be retained in the **job** subdirectory of the

8

home directory, and login **rje** is notified. If this subfield contains an **n** or has any other value, undeliverable output will be discarded. The default is **n**.

(4) **console**

This subfield specifies the status of the interactive status terminal for this line. If the subfield contains an **i**, all console status facilities are inhibited (e.g., *rjestat*(1C) will not behave like a status terminal). In all cases, the normal non-interactive uses of *rjestat*(1C) will continue to function. The default is **y**.

(5) **dial-up**

This subfield contains a telephone number to be used to call a host machine. The telephone number may contain the digits 0 thru 9 and the character — which denotes a pause. If the telephone number is not present, no dialing is attempted and a leased line is assumed.

Sign-on is controlled by the existence of a **signon** file in the home directory. If this file is present, its contents are sent as a sign-on message to the host system. If this file does not exist, a blank card is sent. Sign-off is controlled in the same way, except that the **signoff** file is sent by *rjehalt* if it exists. If the **signoff** file does not exist, a "**/*signoff**" card is sent. These files should be ASCII text and no more than 80 characters.

Send(1C) and *rjestat*(1C) select an available connection by indexing on the **host** field of the configuration table. RJE programs index on the **prefix** field. A subordinate directory, **sque**, exists in **/usr/rje** for use by *rjedisp* and *shqer* programs. This directory holds those output files that have been designated as standard input to some executable file. This designation is done via the "usr=..." specification. *Rjedisp* places the output files here and updates the file **log** to specify the order of execution, arguments to be passed, etc. *Shqer* executes the appropriate files.

All RJE programs are shared text; therefore, if more than one RJE is to be run on a given UNIX system, simply link (via *ln*) RJE2 program names to RJE names in **/usr**.

8

SEE ALSO

cp(1), rjestat(1C), send(1C), pnch(4), un53(7), vpm(7), mk(8).
UNIX System User's Guide.
UNIX Remote Job Entry Administrator's Guide in the *UNIX System Administrator's Guide*.
Setting up UNIX in the *UNIX System Administrator's Guide*.

DIAGNOSTICS

Rjeinit provides brief error messages describing obstacles encountered while bringing up RJE. They can best be understood in the context of the RJE source code. The most frequently occurring one is "cannot open /dev/vpm?". This may occur if the VPM script has not been started, or if another process already has the VPM device open.

Once RJE has been started, users should assist in monitoring its performance, and should notify operations personnel of any perceived need for remedial action. *Rjestat*(1C) will aid in diagnosing the current state of RJE. It can detect, with some reliability, when the far end of the communications line has gone dead, and will report in this case that the host computer is not responding to RJE.

NAME

romboot — special ROM bootstrap loaders

DESCRIPTION

To bootstrap programs from various storage media, standard DEC ROM bootstrap loaders are often used. However, such standard loaders may not be compatible with UNIX bootstrap programs or may not exist on a particular system. Thus, special bootstrap loaders were designed that may be cut into a programmable ROM (M792 read-only-memory) or manually toggled into memory.

Each program is position-independent, that is, it may be located anywhere in memory. Normally, it is loaded into high core to avoid being overwritten. Each reads one block from drive 0 into memory starting at address 0 and then jumps to address 0. To minimize the size, each assumes that a system INIT was generated prior to execution. Also, the address of one of the device registers is used to set the byte count register or word count register. In each case, this will read in at least 256 words, which is the maximum size of bootstrap programs.

On disk devices, block 0 is read; on tape devices, one block from the current position. Thus, the tape should be positioned at the load point (endzone if DECtape) prior to booting. Also, the standard DEC bootstrap loader for magnetic tape may be emulated by positioning the tape at the load point and executing the bootstrap loader twice.

By convention, on PDP 11/45 systems, address 773 000 is the start of a tape bootstrap loader, and 773 020 the start of a disk bootstrap loader. The actual loaders used depend on the particular hardware configuration.

SEE ALSO

70boot(8), unixboot(8).

CODE

TC11 — DECtape

```
    012700          mov     $tcba,r0
    177346
    010040          mov     r0,-(r0)        /use tc addr for wc
    012740          mov     $3,-(r0)        /read bn forward
    000003
    105710    1:    tstb    (r0)            /wait for ready
    002376          bge     1b
    112710          movb    $5,(r0)         /read forward
    000005
    105710    1:    tstb    (r0)            /wait for ready
    002376          bge     1b
    005007          clr     pc              /transfer to zero
```

TU10 — Magnetic Tape

```
    012700          mov     $mtcma,r0
    172526
    010040          mov     r0,-(r0)        /use mt addr for bc
    012740          mov     $60003,-(r0)    /read, 800 bpi, 9 track
    060003
    105710    1:    tstb    (r0)            /wait for ready
    002376          bge     1b
    005007          clr     pc              /transfer to zero
```

TU16 — Magnetic Tape
```
    012700          mov     $mtwc,r0
    172442
    012760          mov     $1300,30(r0)   /set 800 bpi, PDP format
    001300
    000030
    010010          mov     r0,(r0)        /use mt addr for wc
    012740          mov     $71,-(r0)      /read
    000071
    105710    1:    tstb    (r0)           /wait for ready
    002376          bge     1b
    005007          clr     pc             /transfer to zero
```
RK05 — Disk Pack
```
    012700          mov     $rkda,r0
    177412
    005040          clr     -(r0)
    010040          mov     r0,-(r0)       /use rk addr for wc
    012740          mov     $5,-(r0)       /read
    000005
    105710    1:    tstb    (r0)           /wait for ready
    002376          bge     1b
    005007          clr     pc             /transfer to zero
```
RP03 — Disk Pack
```
    012700          mov     $rpmr,r0
    176726
    005040          clr     -(r0)
    005040          clr     -(r0)
    005040          clr     -(r0)
    010040          mov     r0,-(r0)       /use rp addr for wc
    012740          mov     $5,-(r0)       /read
    000005
    105710    1:    tstb    (r0)           /wait for ready
    002376          bge     1b
    005007          clr     pc             /transfer to zero
```
RP04 — Disk Pack
```
    012700          mov     $rpcs1,r0
    176700
    012720          mov     $21,(r0)+      /read-in preset
    000021
    012760          mov     $10000,30(r0)  /set to 16-bits/word
    010000
    000030
    010010          mov     r0,(r0)        /use rp addr for wc
    012740          mov     $71,-(r0)      /read
    000071
    105710    1:    tstb    (r0)           /wait for ready
    002376          bge     1b
    005007          clr     pc             /transfer to zero
```

8

NAME

tapeboot — magnetic tape bootstrap program

DESCRIPTION

Tapeboot handles the problem of booting a PDP-11/45 or PDP-11/70 from a TU10 or TU16 tape transport. In both cases, the tape density is 800 bpi. The complete program fits in one 512 byte block, but is duplicated so that one copy resides in block 0 and another in block 1. Thus, both the standard DEC ROM bootstrap loaders and the special ROM loaders will work. For example, to create a boot tape, execute:

cat /stand/tapeboot *program-to-boot* >/dev/mt0

To boot from magnetic tape, read the first record of the tape into memory starting at address 0 and then jump to address 0, using a special ROM or some manual procedure (toggle in the program). The bootstrap program relocates itself to high core as specified when assembled (typically 24K words, maximum of 28K). It then determines whether to use the TU10 code or the TU16 code. The TU10 is used if the TM11 command register (772 522) exists and the function (bits <3:1>) is non-zero, otherwise the TU16 is used. It then types on the console UNIX **tape boot loader**, rewinds the tape, reads two blocks to skip past itself on the tape, clears memory, and reads the rest of the tape, to the tape mark, into memory starting at address 0. If address 0 contains 000 407, a UNIX **a.out** program is assumed and the first 8 words are stripped off by relocating the loaded program toward address 0. Finally, a jump to address 0 is done by executing **jsr pc,*$0**.

If there is an error while reading the tape, the bootstrap program will type **tape error** and attempt to read the record again.

FILES

/stand/tapeboot	TU10/TU16 magtape bootstrap
/usr/src/stand	source directory

SEE ALSO

unixboot(8).

8

NAME

trouble — trouble reporting system

DESCRIPTION

The first line of the **/usr/lib/trouble/trsh** file must have the correct company code for your site; a local modification is necessary. The *trouble* command will not run until this change is made.

The official company codes are as follows:

at	AT&T
bl	Bell Labs
cb	Cincinnati Bell
cd	C&P of Washington
cm	C&P of Maryland
cv	C&P of Virginia
cw	C&P of West Virginia
lb	Illinois Bell
ll	AT&T Long Lines
mb	Michigan Bell
ms	Mountain States Telephone
nb	Indiana Bell
ne	New England Telephone
nj	New Jersey Bell
nv	Nevada Bell
nw	Northwestern Bell
ny	New York Telephone
ob	Ohio Bell
pa	Bell of Pennsylvania
pn	Pacific Northwest Bell
pt	Pacific Telephone & Telegraph
sb	Southern Bell
sc	South Central Bell
sn	Southern New England Telephone
sw	Southwestern Bell
we	Western Electric
wt	Wisconsin Telephone

8

All trouble reports are archived in **/usr/lib/trouble/tr.a**; this file should be checked weekly to ensure that it does not get too large. If it gets too large, it should be moved to **/usr/lib/trouble/otr.a**; after a week or so, the old archive can be thrown away.

The **trouble** login is intended to be used only for administering the *trouble* system. If *uucp* cannot deliver a trouble report, *mail*(1) will be returned to the **trouble** login. Any trouble reports not delivered may be retransmitted by using the *trxmit* command with the trouble report numbers as arguments. (Hence, the reason for saving the archive for a while.)

The **/usr/lib/trouble/names** file can be expanded to include the names of additional people at your site.

The per-line format of this file is as follows:

letter-IDs(3-6) location phone name(with appropriate blanks)

The above fields are separated by blanks and/or tabs. When the *letter-ID* is identified, the *name*, *location* and *phone* will be taken from this file (provided they have legal formats). Note that the *name* field is the only one that can have blanks.

FILES

/usr/lib/trouble/tr.a	archived trouble reports
/usr/lib/trouble/instruct	instructions
/usr/lib/trouble/trsh	trouble report shell
/usr/lib/trouble/trxmit	re-transmission shell
/usr/lib/trouble/names	letter ID data base

SEE ALSO

trouble(1), uucp(1C).

8

NAME

unixboot – UNIX startup and boot procedures

DESCRIPTION

How to start UNIX. UNIX is started by placing it in core at location zero and transferring to zero. Since the system is not reenterable, it is necessary to read it in from disk or tape. See *diskboot*(8) or *tapeboot*(8).

The switches. On systems with console switches, the switches are examined 60 times per second, and the contents of the address specified by the switches are displayed in the display register. If the switch address is even, the address is interpreted in kernel (system) space; if odd, the rounded-down address is interpreted in the current user space.

Init. The operating system invokes *init*(1M) as process number 1. It comes up conventionally in single-user mode.

FILES

/unix UNIX code

SEE ALSO

init(1M), 70boot(8), diskboot(8), romboot(8), tapeboot(8).