UNIX®
SYSTEM V
Release 4

System Files and Devices Reference Manual

for
Motorola Processors

MOTOROLA

UNIX®
SYSTEM V
Release 4

System Files and Devices Reference Manual

for
Motorola Processors

MOTOROLA

Published by PTR Prentice-Hall, Inc.
A Simon & Schuster Company
Englewood Cliffs, New Jersey 07632

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-035874-6

Table of Contents

Introduction

File Formats(4) and Special Files(7)

Introduction

Reference Manuals

Description Manual pages provide technical reference information about the interfaces and execution behavior of each UNIX SYSTEM V Release 4 component.

Organization The *type* of component being described is indicated by the numerical section suffix. Within each section there may be subsections indicated by a single letter. Related sections are organized into reference manuals and alphabetized by name. The following table shows the contents of the reference manuals and their section suffixes.

Title and Contents	Sections
Commands Reference Manual Volumes 1 and 2	
General-purpose user commands	1
Basic networking commands	1C
Form and Menu Language Interpreter (FMLI)	1F
System maintenance commands	1M
Enhanced networking commands	1N
Miscellaneous reference information related to commands.	5
System Calls and Library Functions Reference Manual	
System calls	2
BSD system compatibility library	3
Standard C library	3C
Executable and linking format library	3E

Continued on next page

Reference Manuals, Continued

Contents	Sections
System Calls and Library Functions Reference Manual (continued)	
General-purpose library	3G
Math library	3M
Networking library	3N
Standard I/O library	3S
Specialized library	3X
Miscellaneous reference information related to programming.	5
System Files and Devices Reference Manual	
System file formats	4
Special files (devices)	7
Device Driver Interface/Driver - Kernel Interface Reference Manual	
Driver Data Definitions	D1
Driver Entry Point Routines	D2
Kernel Utility Routines	D3
Kernel Data Structures	D4
Kernel Defines	D5
Master Permuted Index	
Permuted index of all manual pages	All

Retitled Reference Manuals

Background Four reference manuals for this release have been restructured and/or retitled to more accurately describe their contents. The following table shows these changes.

Previous Titles	Current Titles	Current Sections
User's Reference Manual/ System Administrator's Reference Manual (Commands a - l) (Commands m - z)	*Commands Reference Manual (Volume 1, a - l) (Volume 2, m - z)*	1, 1C, 1F, 1M, 1N, 5
Programmer's Reference Manual: Operating System API Part 1: Programming Commands and System Calls Part 2: Functions	*System Calls and Library Functions Reference Manual*	2, 3, 3C, 3E, 3G, 3M, 3N, 3S, 3X, 5
System Files and Devices Reference Manual	*System Files and Devices Reference Manual* (section 5 removed)	4, 7
Permuted Index	*Master Permuted Index*	All

Manual Page Format

Main headings used

All UNIX manual pages have a common format. The following main headings are used:

Heading	Section Contents
NAME	Name of the component and brief statement of its purpose
SYNOPSIS	Syntax of the component
DESCRIPTION	General discussion of functionality
EXAMPLE	Example(s) of usage
FILES	File names built into the component
SEE ALSO	Cross-references to related components

Note: Not all manual pages use all headings.

Typographical Conventions

Style and conventions used

The following typographical and formatting conventions are used.

Convention	Indicates ...
`Constant width`	a literal that should be entered just as it appears
Italic	a substitutable argument
Square brackets around an argument [ ]	an optional argument
name or *file*	a file name
Ellipses ...	previous argument may be repeated
Argument beginning with - minus + plus = equal	a flag argument

Permuted Index

Definition

A permuted index is an alphabetical listing of all the keywords in the **NAME** line of a manual page.

Certain common words are not considered keywords and are not recognized. In the example below, the common words *of*, *to*, and *the* are not recognized.

Example

The **NAME** line of the `adjtime`(2) manual page appears below.

> **adjtime(2)**   **adjtime(2)**
>
> **NAME**
>
> `adjtime`- correct the time to allow synchronization of the system clock

The `adjtime`(2) entries from the permuted index are shown below. These entries appear in the a, c, and s sections of the permuted index respectively.

Remainder of NAME line	Keyword and NAME line	Manual Page
synchronization of the system/	adjtime correct the time to allow.	adjtime(2)
clock adjtime correct the time to	allow synchronization of the system . . .	adjtime(2)
allow synchronization of the system	clock adjtime correct the time to . . .	adjtime(2)
synchronization of the/ adjtime	correct the time to allow	adjtime(2)
adjtime correct the time to allow	synchronization of the system clock . . .	adjtime(2)
to allow synchronization of the	system clock / correct the time	adjtime(2)

Continued on next page

Permuted Index, Continued

How a permuted index is constructed

The center column lists each keyword followed by all or a portion of the **NAME** line, as space permits. The left column lists the remainder of the **NAME** line. The right column indicates the manual page being referenced.

Omitted words are indicated with a slash (/).

Identification of entries

Manual page entries are identified with their section suffixes shown in parentheses.

Example: `man`(1) and `man`(5)

Section suffixes eliminate confusion caused by duplication of names among the sections.

Master Permuted Index

Each reference manual has a permuted index for the manual pages contained in that book.

The *Master Permuted Index* covers all the manual pages of this documentation library.

Request for Comment

Description

A Request for Comment (RFC) is a document that describes some aspect of networking technology. The RFCs cited in the **SEE ALSO** section of these manual pages are available in hard copy for a small fee from:

Network Information System Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
415-859-6387 fax: 415-859-6028
email:`nisc@nisc.sri.com`

Online versions of RFCs

Online versions of the RFCs are available by `ftp` from `nic.ddn.mil`.To retrieve an on-line RFC, do the following:

Step	Action

1	Connect to the RFC host by entering:
	
	`ftp nic.ddn.mil`
	`user name: anonymous`
	`password: guest`

2	Retrieve the RFC by entering:
	`get rfc/rfc`*num*
	
	`where` *num* `is the number of the RFC`
	
	<u>Example</u>:
	`get rfc:rfc1171.txt`

3	End the `ftp` session by entering: `quit`

NAME

a.out - ELF (Executable and Linking Format) files

SYNOPSIS

```
#include <elf.h>
```

DESCRIPTION

The file name a.out is the default output file name from the link editor, ld(1). The link editor will make an a.out executable if there were no errors in linking. The output file of the assembler, as(1), also follows the format of the a.out file although its default file name is different.

Programs that manipulate ELF files may use the library that elf(3E) describes. An overview of the file format follows. For more complete information, see the references given below.

Linking View
ELF header
Program header table *optional*
Section 1
. . .
Section *n*
. . .
. . .
Section header table

Execution View
ELF header
Program header table
Segment 1
Segment 2
. . .
Section header table *optional*

An ELF header resides at the beginning and holds a "road map" describing the file's organization. Sections hold the bulk of object file information for the linking view: instructions, data, symbol table, relocation information, and so on. Segments hold the object file information for the program execution view. As shown, a segment may contain one or more sections.

A program header table, if present, tells the system how to create a process image. Files used to build a process image (execute a program) must have a program header table; relocatable files do not need one. A section header table contains information describing the file's sections. Every section has an entry in the table; each entry gives information such as the section name, the section size, and so on. Files used during linking must have a section header table; other object files may or may not have one.

Although the figure shows the program header table immediately after the ELF header, and the section header table following the sections, actual files may differ. Moreover, sections and segments have no specified order. Only the ELF header has a fixed position in the file.

When an a.out file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. The text segment is not writable by the program; if other processes are executing the same a.out file, the processes will share a single text segment.

The data segment starts at the next maximal page boundary past the last text address. (If the system supports more than one page size, the "maximal page" is the largest supported size.) When the process image is created, the part of the file holding the end of text and the beginning of data may appear twice. The duplicated chunk of text that appears at the beginning of data is never executed; it is duplicated so that the operating system may bring in pieces of the file in multiples of the actual page size without having to realign the beginning of the data section to a page boundary. Therefore, the first data address is the sum of the next maximal page boundary past the end of text plus the remainder of the last text address divided by the maximal page size. If the last text address is a multiple of the maximal page size, no duplication is necessary. The stack is automatically extended as required. The data segment is extended as requested by the `brk`(2) system call.

SEE ALSO

`as`(1), `cc`(1), `ld`(1), `brk`(2), `elf`(3E).

NAME

acct - per-process accounting file format

SYNOPSIS

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

DESCRIPTION

Files produced as a result of calling acct(2) have records in the form defined by sys/acct.h, whose contents are:

```
typedef  ushort comp_t;     /* "floating point" */
           /* 13-bit fraction, 3-bit exponent  */

struct  acct
{
        char   ac_flag;     /* Accounting flag */
        char   ac_stat;     /* Exit status */
        uid_t  ac_uid;      /* Accounting user ID */
        gid_t  ac_gid;      /* Accounting group ID */
        dev_t  ac_tty;      /* control typewriter */
        time_t ac_btime;    /* Beginning time */
        comp_t ac_utime;    /* acctng user time in clock ticks */
        comp_t ac_stime;    /* acctng system time in clock ticks */
        comp_t ac_etime;    /* acctng elapsed time in clock ticks */
        comp_t ac_mem;      /* memory usage in clicks */
        comp_t ac_io;       /* chars trnsfrd by read/write */
        comp_t ac_rw;       /* number of block reads/writes */
        char   ac_comm[8];  /* command name */
};

extern  struct acct       acctbuf;
extern  struct vnode      *acctp;  /* vnode of accounting file */

#define AFORK  01          /* has executed fork, but no exec */
#define ASU    02          /* used super-user privileges */
#define ACCTF  0300        /* record type: 00 = acct */
#define AEXPND 040         /*Expanded Record Type*/
```

In ac_flag, the AFORK flag is turned on by each fork and turned off by an exec. The ac_comm field is inherited from the parent process and is reset by any exec. Each time the system charges the process with a clock tick, it also adds to ac_mem the current process size, computed as follows:

(data size) + (text size) / (number of in-core processes using text)

The value of ac_mem / (ac_stime + ac_utime) can be viewed as an approximation to the mean process size, as modified by text sharing.

The structure tacct, which resides with the source files of the accounting commands, represents the total accounting format used by the various accounting commands:

```
/*
 *  total accounting (for acct period), also for day
 */

struct tacct {
       uid_t          ta_uid;      /* userid */
       char           ta_name[8];  /* login name */
       float          ta_cpu[2];   /* cum. cpu time, p/np (mins) */
       float          ta_kcore[2]; /* cum kcore-minutes, p/np */
       float          ta_con[2];   /* cum. connect time, p/np, mins */
       float          ta_du;       /* cum. disk usage */
       long           ta_pc;       /* count of processes */
       unsigned short ta_sc;       /* count of login sessions */
       unsigned short ta_dc;       /* count of disk samples */
       unsigned short ta_fee;      /* fee for special services */
};
```

SEE ALSO

acct(1M), acctcom(1), acct(2), exec(2), fork(2),

NOTES

The ac_mem value for a short-lived command gives little information about the actual size of the command, because ac_mem may be incremented while a different command (for example, the shell) is being executed by the process.

NAME

`admin` - installation defaults file

DESCRIPTION

`admin` is a generic name for an ASCII file that defines default installation actions by assigning values to installation parameters. For example, it allows administrators to define how to proceed when the package being installed already exits on the system.

`/var/sadm/install/admin/default` is the default `admin` file delivered with System V Release 4.0. The default file is not writable, so to assign values different from this file, create a new `admin` file. There are no naming restrictions for `admin` files. Name the file when installing a package with the `-a` option of `pkgadd`. If the `-a` option is not used, the default `admin` file is used.

Each entry in the `admin` file is a line that establishes the value of a parameter in the following form:

param=value

Eleven parameters can be defined in an `admin` file. A file is not required to assign values to all eleven parameters. If a value is not assigned, `pkgadd` asks the installer how to proceed.

The eleven parameters and their possible values are shown below except as noted. They may be specified in any order. Any of these parameters can be assigned the value `ask`, which means that if the situation occurs the installer is notified and asked to supply instructions at that time.

`basedir` Indicates the base directory where relocatable packages are to be installed. The value may contain `$PKGINST` to indicate a base directory that is to be a function of the package instance.

`mail` Defines a list of users to whom mail should be sent following installation of a package. If the list is empty, no mail is sent. If the parameter is not present in the `admin` file, the default value of `root` is used. The `ask` value cannot be used with this parameter.

`runlevel` Indicates resolution if the run level is not correct for the installation or removal of a package. Options are:

- `nocheck` Do not check for run level.
- `quit` Abort installation if run level is not met.

`conflict` Specifies what to do if an installation expects to overwrite a previously installed file, thus creating a conflict between packages. Options are:

- `nocheck` Do not check for conflict; files in conflict will be overwritten.
- `quit` Abort installation if conflict is detected.
- `nochange` Override installation of conflicting files; they will not be installed.

`setuid` Checks for executables which will have setuid or setgid bits enabled after installation. Options are:

- `nocheck` Do not check for setuid executables.
- `quit` Abort installation if setuid processes are detected.
- `nochange` Override installation of setuid processes; processes will be installed without setuid bits enabled.

`action` Determines if action scripts provided by package developers contain possible security impact. Options are:

- `nocheck` Ignore security impact of action scripts.
- `quit` Abort installation if action scripts may have a negative security impact.

`partial` Checks to see if a version of the package is already partially installed on the system. Options are:

- `nocheck` Do not check for a partially installed package.
- `quit` Abort installation if a partially installed package exists.

`instance` Determines how to handle installation if a previous version of the package (including a partially installed instance) already exists. Options are:

- `quit` Exit without installing if an instance of the package already exists (does not overwrite existing packages).
- `overwrite` Overwrite an existing package if only one instance exists. If there is more than one instance, but only one has the same architecture, it overwrites that instance. Otherwise, the installer is prompted with existing instances and asked which to overwrite.
- `unique` Do not overwrite an existing instance of a package. Instead, a new instance of the package is created. The new instance will be assigned the next available instance identifier.

`idepend` Controls resolution if other packages depend on the one to be installed. Options are:

- `nocheck` Do not check package dependencies.
- `quit` Abort installation if package dependencies are not met.

`list_files`
Controls whether files are listed during processing. Options are:

- `nocheck` Do not list files during processing. Any other value causes files to be listed.

`rdepend` Controls resolution if other packages depend on the one to be removed. Options are:

nocheck	Do not check package dependencies.
quit	Abort removal if package dependencies are not met.

space — Controls resolution if disk space requirements for package are not met. Options are:

nocheck	Do not check space requirements (installation fails if it runs out of space).
quit	Abort installation if space requirements are not met.

NOTES

The value ask should not be defined in an admin file that will be used for non-interactive installation (since by definition, there is no installer interaction). Doing so causes installation to fail when input is needed.

EXAMPLE

```
basedir=default
runlevel=quit
conflict=quit
setuid=quit
action=quit
partial=quit
instance=unique
idepend=quit
rdepend=quit
space=quit
```

NAME

aliases, addresses, forward - addresses and aliases for sendmail

SYNOPSIS

/usr/ucblib/aliases
/usr/ucblib/aliases.dir
/usr/ucblib/aliases.pag
~/.forward

DESCRIPTION

These files contain mail addresses or aliases, recognized by sendmail, for the local host:

/etc/passwd
: Mail addresses (usernames) of local users.

/usr/ucblib/aliases
: Aliases for the local host, in ASCII format. This file can be edited to add, update, or delete local mail aliases.

/usr/ucblib/aliases. { dir , pag}
: The aliasing information from /usr/ucblib/aliases, in binary, dbm format for use by sendmail. The program, newaliases, maintains these files.

~/.forward
: Addresses to which a user's mail is forwarded (see Automatic Forwarding, below).

In addition, the Network Information Service (NIS) aliases map *mail.aliases* contains addresses and aliases available for use across the network.

Addresses

As distributed, sendmail supports the following types of addresses:

Local Usernames

username

Each local *username* is listed in the local host's /etc/passwd file.

Local Filenames

pathname

Messages addressed to the absolute *pathname* of a file are appended to that file.

Commands

| *command*

If the first character of the address is a vertical bar, (|), sendmail pipes the message to the standard input of the *command* the bar precedes.

DARPA-standard Addresses

username@domain

If *domain* does not contain any '.' (dots), then it is interpreted as the name of a host in the current domain. Otherwise, the message is passed to a *mailhost* that determines how to get to the specified domain. Domains are divided into subdomains separated by dots, with the top-level domain on the right. Top-level domains include:

Commercial organizations.

Educational organizations.

Government organizations.

Military organizations.

For example, the full address of John Smith could be:

`js@jsmachine.Podunk-U.EDU`

if he uses the machine named `jsmachine` at Podunk University.

uucp Addresses

... [*host* !] *host* ! *username*

These are sometimes mistakenly referred to as "Usenet" addresses. `uucp` provides links to numerous sites throughout the world for the remote copying of files.

Other site-specific forms of addressing can be added by customizing the `sendmail` configuration file. See the `sendmail`(1M) for details. Standard addresses are recommended.

Aliases

Local Aliases

`/usr/ucblib/aliases` is formatted as a series of lines of the form

aliasname : *address*[, *address*]

aliasname is the name of the alias or alias group, and *address* is the address of a recipient in the group. Aliases can be nested. That is, an *address* can be the name of another alias group. Because of the way `sendmail` performs mapping from upper-case to lower-case, an *address* that is the name of another alias group must not contain any upper-case letters.

Lines beginning with white space are treated as continuation lines for the preceding alias. Lines beginning with # are comments.

Special Aliases

An alias of the form:

`owner- aliasname :` *address*

directs error-messages resulting from mail to *aliasname* to *address*, instead of back to the person who sent the message.

An alias of the form:

aliasname`: :include:`*pathname*

with colons as shown, adds the recipients listed in the file *pathname* to the *aliasname* alias. This allows a private list to be maintained separately from the aliases file.

NIS Domain Aliases

Normally, the aliases file on the master NIS server is used for the *mail.aliases* NIS map, which can be made available to every NIS client. Thus, the `/usr/ucblib/aliases*` files on the various hosts in a network will one day be obsolete. Domain-wide aliases should ultimately be resolved into usernames on specific hosts. For example, if the following were in the domain-wide alias file:

```
jsmith:js@jsmachine
```

then any NIS client could just mail to jsmith and not have to remember the machine and username for John Smith. If a NIS alias does not resolve to an address with a specific host, then the name of the NIS domain is used. There should be an alias of the domain name for a host in this case. For example, the alias:

```
jsmith:root
```

sends mail on a NIS client to root@podunk-u if the name of the NIS domain is podunk-u.

Automatic Forwarding

When an alias (or address) is resolved to the name of a user on the local host, sendmail checks for a .forward file, owned by the intended recipient, in that user's home directory, and with universal read access. This file can contain one or more addresses or aliases as described above, each of which is sent a copy of the user's mail.

Care must be taken to avoid creating addressing loops in the .forward file. When forwarding mail between machines, be sure that the destination machine does not return the mail to the sender through the operation of any NIS aliases. Otherwise, copies of the message may ''bounce.'' Usually, the solution is to change the NIS alias to direct mail to the proper destination.

A backslash before a username inhibits further aliasing. For instance, to invoke the vacation program, user js creates a .forward file that contains the line:

```
\js, "|/usr/ucb/vacation js"
```

so that one copy of the message is sent to the user, and another is piped into the vacation program.

FILES

/etc/passwd
/usr/ucblib/aliases
~/.forward

SEE ALSO

newaliases(1M), sendmail(1M), vacation(1), dbm(3X), uucp(1C).

NOTES

Because of restrictions in dbm a single alias cannot contain more than about 1000 characters. Nested aliases can be used to circumvent this limit.

NAME

alp - Algorithm Pool management module

DESCRIPTION

The *STREAMS* module alp maintains a pool of algorithms (in the form of *STREAMS*-compatible subroutines) that may be used for processing *STREAMS* data messages. Interfaces are defined allowing modules to request and initiate processing by any of the algorithms maintained in the pool. It is expected to help centralize and standardize the interfaces to algorithms that now represent a proliferation of similar-but-different *STREAMS* modules. Its major use is envisioned as a central registry of available codeset conversion algorithms or other types of common data-manipulating routines.

An *algorithm pool* is a registry (or *pool*) of available functions; in this case, routines for performing transformations on *STREAMS* data messages. Registered functions may keep information on attached users, which means that algorithms need not be "stateless", but may maintain extensive state information related to each connection. An algorithm from the pool is called by another in-kernel module with arguments that are a *STREAMS* data message and a unique identifier. If a message is passed back to the caller, it is the algorithm's output, otherwise the algorithm may store partially convertible input until enough input is received to give back output on a subsequent call.

This pool is one means for providing a consistent and flexible interface for *codeset conversion* within *STREAMS* modules, especially kbd, but it may also be used to provide other services that are commonly duplicated by several modules.

The alp module contains some subroutines dealing with its (minor) role as a module, a data definition for an algorithm list, connection and disconnection routines, and a search routine for finding registered items. The module interface incorporated into alp serves the purpose of providing an ioctl interface, so that users can find out what algorithms are registered [see alpq(1)].

The programmer of a function for use with alp provides a simple *module* with a simple specified interface. The module must have an initialization routine (*xxx*init) which is called at system startup time to register itself with alp, an open routine, and an interface routine (which actually implements the algorithm).

The registry method of dynamically building the list of available functions obviates the need for recompiling modules or otherwise updating a list or reconfiguring other parts of the system to accommodate additions or deletions. To install a new function module, one merely links it with the kernel in whatever manner is standard for that system; there is no need for updating or re-configuring any other parts of the kernel (including alp itself). The remainder of this discussion concerns the in-kernel operation and use of the module.

Calling Sequence

An algorithm is called from the pool by first requesting a connection via the alp connection interface. The alp module returns the function address of an interface routine, and fills in a unique identifier (id) for the connection. The returned function address is NULL on failure (and id is undefined). This is a sample of making a connection to a function managed by alp:

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

```
     unsigned char *name;      /* algorithm name */
     caddr_t id;               /* unique id */
     mblk_t *(*func)();        /* ptr to func ret'ng ptr to mblk_t */
/*
 *   mblk_t *(*alp_con(unsigned char *, caddr_t))(mblk_t *, caddr_t);
 */
     ...

     if (func = alp_con(name, (caddr_t) &id))
           regular processing;
     else
           error processing;
```

Once the connection has been made, the interface routine can be called directly by the connecting module to process messages:

```
     mblk_t *inp, *outp;
     mblk_t *(*func)();
     ...
     outp = (*func)(mp, id);
     mp = NULL;    /* mp cannot be re-used! */
     if (outp)
           regular processing;
```

If the interface routine processed the entire message, then `outp` is a valid pointer to the algorithm's output message. If, however, the routine needs more information, or is buffering something, `outp` will be a null pointer. In either case, the original message (`mp`) may *not* be subsequently accessed by the caller. The interface routine takes charge of the message `mp`, and may free it or otherwise dispose of it (it may even return the same message). The caller may pass a null message pointer to an interface routine to cause a flush of any data being held by the routine; this is useful for end-of-file conditions to insure that all data has been passed through. (Interface routines must thus recognize a null message pointer and deal with it.)

Synchronization between input and output messages is not guaranteed for all items in the pool. If one message of input does not produce one message of output, this fact should be documented for that particular module. Many multibyte codeset conversion algorithms, to cite one instance, buffer partial sequences, so that if a multibyte character happens to be spread across more than one message, it may take two or more output messages to complete translation; in this case, it is only possible to synchronize when input message boundaries coincide with character boundaries.

Building an Algorithm for the Pool

As mentioned, the modules managed by `alp` are implemented as simple modules—*not STREAMS* modules—each with an initialization routine, an open routine, and a user-interface routine. The initialization routine is called when the system is booted and prior to nearly everything else that happens at boot-time. The routine takes no arguments and its sole purpose is to register the algorithm with the `alp` module, so that it may subsequently accessed. Any other required initialization may also be performed at that time. A generic initialization routine for a module called `GEN`, with prefix `gen` is as follows:

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

static mblk_t *genfunc(); /* interface routine */
caddr_t genopen();
static struct algo genlogo = {
      0,           /* in-core */
      (queue_t *)0,     /* read queue */
      (queue_t *)0,     /* write queue */
      genfunc,     /* interface routine */
      genopen,     /* open/close routine */
      (unsigned char *)"name",
      (unsigned char *)"explanation",
      (struct algo *)0
};
/*
 *    int alp_register(struct algo *);
 */

geninit()
{
      int rval;   /* return value from registrar */

      rval = alp_register(&genlogo);
      if (rval) cmn_err(CE_WARN, "warning message");
}
```

The registration routine, `alp_register` takes one argument and returns zero if successful. The argument is a pointer to the structure `algo` which has members (1) a pointer to the algorithm's entry point (in this case, the function `genfunc`), (2) a pointer to its name, and (3) a pointer to a character string containing a brief explanation. The name should be limited to under 16 bytes, and the explanation to under 60 bytes, as shown in the following example. Neither the name nor the explanation need include a newline.

It is possible for a single module to contain several different, related algorithms, which can each be registered separately by a single *init* routine.

A module's open routine is called by `alp_con` when a connection is first requested by a user (that is, a module that wishes to use it). The open routine takes two arguments. The first argument is an integer; if it is non-zero, the request is an ''open'' request, and the second argument is unused. The function should allocate a unique identifier and return it as a generic address pointer. If the first argument is zero, the request is a ''close'' request, and the second argument is the unique identifier that was returned by a previous open request, indicating which of (potentially several) connections is to be closed. The routine does any necessary clean-up and closes the connection; thereafter, any normal interface requests on that identifier will fail. This use of unique identifiers allows these modules to keep state information relating to each open connection; no format is imposed upon the unique identifier, so it may contain any arbitrary type of information, equivalent in size to a core address; `alp` and most callers will treat it as being of type `caddr_t`, in a manner similar to the private data held by each instantiation of a *STREAMS* module.

A skeleton for the `gen` module's open routine is:

```
caddr_t
genopen(arg, id)
      int arg;
      caddr_t id;
{
      if ( arg ) {
            open processing;
            return( unique-id );
      }
      close processing for id;
      return(0);
}
```

Once a connection has been made, users may proceed as in the example in the previous section. When the connection is to be closed (for example, the connecting module is being popped), a call is made to `alp_discon`, passing the unique id and the name:

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

      caddr_t id;
      char *name;
      mblk_t *mp;
/*
 *    mblk_t *alp_discon(unsigned char *, caddr_t);
 */
      ...
      mp = alp_discon(name, id);
      if (mp)
            process "left-over" data;
```

If the disconnect request returns a valid message pointer (`mp`) then there was unprocessed or partially processed data left in an internal buffer, and it should be dealt with by the caller (for example, by flushing it or sending it to the neighboring module).

The `ioctl` and Query Interfaces

A kernel-level query interface is provided in addition to the query interface supported by the `alpq` command. The routine `alp_query` takes a single argument, a pointer to a *name*. If the name matches a registered function, `alp_query` returns a pointer to the function's *explanation* string, otherwise it returns a null pointer. A calling example is:

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

      unsigned char *name, *expl;
```

```
/*
 *    unsigned char *alp_query(unsigned char *);
 */
      ...
      if (expl = alp_query(name))
            regular processing;
      else
            error processing;
```

The `ioctl` interface provides calls for querying registered functions (for which the *explanation* discussed above is necessary); this is supported by the `alpq` command, which may be used whenever user-level programs need the associated information.

Uses

The `alp` module can be used to replace various kernel-resident codeset conversion functions in international or multi-language environments. The KBD subsystem (which supplies codeset conversion and keyboard mapping) supports the use of `alp` algorithms as processing elements.

Since state information may be maintained, functions may also implement processing on larger or more structured data elements, such as transaction records and network packets. Currently, *STREAMS* CPU priority is assumed by `alp` or should be set individually by interface and open routines.

FUTURE DIRECTIONS

It should also provide a service interface, so that the algorithms registered there might be used directly by programs running at user-level.

SEE ALSO

alpq(1), kbd(7).

EXAMPLES

```
/* Copyright (c) 1989, 1990 AT&T.  All Rights Reserved. */
#ident     "@(#)dely.c     1.0 AT&T USO PACIFIC 1990/03"

/*
 * This is a SAMPLE module that registers with ALP and performs
 * a one-message delay.
 */
#include <sys/types.h>
#include <sys/stream.h>
#include <sys/stropts.h>
#include <sys/kmem.h>
#include <sys/alp.h>

static mblk_t *dely();
caddr_t delyopen();

/*
 * Our state structure.  Keeps its own address and a pointer.
 */
struct dstruct {
      caddr_t d_unique;
```

```
	mblk_t *d_mp;
};

/*
 * The name is "Dely".  It has an open routine "delyopen"
 * and an interface "dely".
 */
static struct algo delyalgo =
{
	0, (queue_t *) 0, (queue_t *) 0, dely, delyopen,
	(unsigned char *) "Dely",
	(unsigned char *) "One Message Delay Buffer",
	(struct algo *) 0
};

/*
 * This is the sysinit routine, called when the system is
 * being brought up.  It registers "Dely" with ALP.
 */
delyinit()
{
	if (alp_register(&delyalgo))	/* then register with ALP */
		printf("DELY: register failed\n");
}

/*
 * This is the interface routine itself.
 * Holds onto "mp" and returns whatever it had before.
 */
static mblk_t *
dely(mp, id)
	mblk_t *mp;
	caddr_t id;
{
	register mblk_t *rp;
	register struct dstruct *d;

	d = (struct dstruct *) id; /* clarify the situation */
	rp = d->d_mp;
	d->d_mp = mp;
	return(rp);		/* return the previous message */
}

/*
 * The open (and close) routine.
 * Use kmem_zalloc() to get a private
 * structure for saving state info.
 */
caddr_t
delyopen(arg, id)
```

```
        int arg;          /* 1 = open, 0 = close */
        caddr_t id;       /* ignored on open; is unique id on close */
{
        register struct dstruct *d;
        register mblk_t *rp;

        if (! arg) {      /* close processing */
                d = (struct dstruct *) id;
                d->d_unique = (caddr_t) -1;
                rp = d->d_mp;
                kmem_free(d, sizeof(struct dstruct));
                return((caddr_t) rp);
        }
        /* otherwise, open processing */
        d = (struct dstruct *) kmem_zalloc(sizeof(struct dstruct),
                KM_NOSLEEP);
        d->d_unique = (caddr_t) &d;
        return((caddr_t) d);
}
```

NAME

ar - archive file format

SYNOPSIS

```
#include <ar.h>
```

DESCRIPTION

The archive command ar is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor ld.

Each archive begins with the archive magic string.

```
#define  ARMAG   "!<arch>\n"   /* magic string */
#define  SARMAG  8             /* length of magic string */
```

Following the archive magic string are the archive file members. Each file member is preceded by a file member header which is of the following format:

```
#define  ARFMAG    "`\n"

struct  ar_hdr           /* file member header */
{
    char    ar_name[16];/* '/' terminated file member name */
    char    ar_date[12];/* file member date */
    char    ar_uid[6];   /* file member user identification */
    char    ar_gid[6];   /* file member group identification */
    char    ar_mode[8];  /* file member mode (octal) */
    char    ar_size[10];/* file member size */
    char    ar_fmag[2];  /* header trailer string */
};
```

All information in the file member headers is in printable ASCII. The numeric information contained in the headers is stored as decimal numbers (except for *ar_mode* which is in octal). Thus, if the archive contains printable files, the archive itself is printable.

If the file member name fits, the *ar_name* field contains the name directly, and is terminated by a slash (/) and padded with blanks on the right. If the member's name does not fit, *ar_name* contains a slash (/) followed by a decimal representation of the name's offset in the archive string table described below.

The *ar_date* field is the modification date of the file at the time of its insertion into the archive. Common format archives can be moved from system to system as long as the portable archive command ar is used.

Each archive file member begins on an even byte boundary; a newline is inserted between files if necessary. Nevertheless, the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

Each archive that contains object files [see a.out(4)] includes an archive symbol table. This symbol table is used by the link editor ld to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and/or updated by `ar`.

The archive symbol table has a zero length name (that is, `ar_name[0]` is `'/'`), `ar_name[1]=='  '`, and so on). All "words" in this symbol table have four bytes, using the machine-independent encoding shown below. (All machines use the encoding described here for the symbol table, even if the machine's "natural" byte order is different.)

	0	1	2	3
`0x01020304`	01	02	03	04

The contents of this file are as follows:

1. The number of symbols. Length: 4 bytes.
2. The array of offsets into the archive file. Length: 4 bytes * "the number of symbols".
3. The name string table. Length: *ar_size* - 4 bytes * ("the number of symbols" + 1).

As an example, the following symbol table defines 4 symbols. The archive member at file offset 114 defines `name` and `object`. The archive member at file offset 426 defines `function` and a second version of `name`.

Offset	+0	+1	+2	+3	
0	4				4 offset entries
4	114				`name`
8	114				`object`
12	426				`function`
16	426				`name`
20	n	a	m	e	
24	\0	o	b	j	
28	e	c	t	\0	
32	f	u	n	c	
36	t	i	o	n	
40	\0	n	a	m	
44	e	\0			

The number of symbols and the array of offsets are managed with `sgetl` and `sputl`. The string table contains exactly as many null terminated strings as there are elements in the offsets array. Each offset from the array is associated with the corresponding name from the string table (in order). The names in the string table are all the defined global symbols found in the common object files in the archive. Each offset is the location of the archive header for the associated symbol.

If some archive member's name is more than 15 bytes long, a special archive member contains a table of file names, each followed by a slash and a new-line. This string table member, if present, will precede all "normal" archive members. The special archive symbol table is not a "normal" member, and must be first if it exists. The *ar_name* entry of the string table's member header holds a zero length name `ar_name[0]=='/'`, followed by one trailing slash (`ar_name[1]=='/'`), followed by blanks (ar_name[2]==' ', and so on). Offsets into the string table begin at zero. Example *ar_name* values for short and long file names appear below.

Offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
0	f	i	l	e	_	n	a	m	e	_
10	s	a	m	p	l	e	/	\n	l	o
20	n	g	e	r	f	i	l	e	n	a
30	m	e	x	a	m	p	l	e	/	\n

Member Name	*ar_name*	Note
short-name	short-name/	Not in string table
file_name_sample	/0	Offset 0 in string table
longerfilenamexample	/18	Offset 18 in string table

SEE ALSO

ar(1), ld(1), strip(1), sputl(3X), a.out(4)

NOTES

strip will remove all archive symbol entries from the header. The archive symbol entries must be restored via the -ts options of the ar command before the archive can be used with the link editor ld.

NAME

archives - device header file

DESCRIPTION

```
/* Magic numbers */

#define CMN_ASC      0x070701     /* Cpio Magic Number for -c header */
#define CMN_BIN      070707       /* Cpio Magic Number for Binary header */
#define CMN_BBS      0143561      /* Cpio Magic Number for Byte-Swap header */
#define CMN_CRC      0x070702     /* Cpio Magic Number for CRC header */
#define CMS_ASC      "070701"     /* Cpio Magic String for -c header */
#define CMS_CHR      "070707"     /* Cpio Magic String for odc header */
#define CMS_CRC      "070702"     /* Cpio Magic String for CRC header */
#define CMS_LEN      6            /* Cpio Magic String length */

/* Various header and field lengths */

#define CHRSZ        76       /* -H odc size minus filename field */
#define ASCSZ        110      /* -c and CRC hdr size minus filename field */
#define TARSZ        512      /* TAR hdr size */

#define HNAMLEN      256      /* max filename length for binary and odc hdrs */
#define EXPNLEN      1024     /* max filename length for -c and CRC headers */
#define HTIMLEN      2        /* length of modification time field */
#define HSIZLEN      2        /* length of file size field */

/* cpio binary header definition */

struct hdr_cpio {
    short    h_magic,                 /* magic number field */
             h_dev;                   /* file system of file */
    ushort   h_ino,                   /* inode of file */
             h_mode,                  /* modes of file */
             h_uid,                   /* uid of file */
             h_gid;                   /* gid of file */
    short    h_nlink,                 /* number of links to file */
             h_rdev,                  /* maj/min numbers for special files */
             h_mtime[HTIMLEN],        /* modification time of file */
             h_namesize,              /* length of filename */
             h_filesize[HSIZLEN];     /* size of file */
    char     h_name[HNAMLEN];         /* filename */
} ;

/* cpio -H odc header format */

struct c_hdr {
    char     c_magic[CMS_LEN],
             c_dev[6],
             c_ino[6],
             c_mode[6],
```

```
              c_uid[6],
              c_gid[6],
              c_nlink[6],
              c_rdev[6],
              c_mtime[11],
              c_namesz[6],
              c_filesz[11],
              c_name[HNAMLEN];
} ;

/* -c and CRC header format */

struct Exp_cpio_hdr {
    char      E_magic[CMS_LEN],
              E_ino[8],
              E_mode[8],
              E_uid[8],
              E_gid[8],
              E_nlink[8],
              E_mtime[8],
              E_filesize[8],
              E_maj[8],
              E_min[8],
              E_rmaj[8],
              E_rmin[8],
              E_namesize[8],
              E_chksum[8],
              E_name[EXPNLEN];
} ;

/* Tar header structure and format */

#define TBLOCK       512 /* length of tar header and data blocks */
#define TNAMLEN      100 /* maximum length for tar file names */
#define TMODLEN      8   /* length of mode field */
#define TUIDLEN      8   /* length of uid field */
#define TGIDLEN      8   /* length of gid field */
#define TSIZLEN      12  /* length of size field */
#define TTIMLEN      12  /* length of modification time field */
#define TCRCLEN      8   /* length of header checksum field */

/* tar header definition */

union tblock {
    char dummy[TBLOCK];
    struct header {
        char t_name[TNAMLEN];            /* name of file */
        char t_mode[TMODLEN];            /* mode of file */
        char t_uid[TUIDLEN];             /* uid of file */
        char t_gid[TGIDLEN];             /* gid of file */
```

```
        char t_size[TSIZLEN];           /* size of file in bytes */
        char t_mtime[TTIMLEN];          /* modification time of file */
        char t_chksum[TCRCLEN];         /* checksum of header */
        char t_typeflag;                /* flag to indicate type of file */
        char t_linkname[TNAMLEN];       /* file this file is linked with */
        char t_magic[6];                /* magic string always "ustar" */
        char t_version[2];              /* version strings always "00" */
        char t_uname[32];               /* owner of file in ASCII */
        char t_gname[32];               /* group of file in ASCII */
        char t_devmajor[8];             /* major number for special files */
        char t_devminor[8];             /* minor number for special files */
        char t_prefix[155];             /* pathname prefix */
    } tbuf;
};

/* volcopy tape label format and structure */

#define VMAGLEN 8
#define VVOLLEN 6
#define VFILLEN 464

struct volcopy_label {
    char    v_magic[VMAGLEN],
            v_volume[VVOLLEN],
            v_reels,
            v_reel;
    long    v_time,
                v_length,
            v_dens,
            v_reelblks,             /* u370 added field */
            v_blksize,              /* u370 added field */
            v_nblocks;              /* u370 added field */
    char    v_fill[VFILLEN];
    long    v_offset;               /* used with -e and -reel options */
    int     v_type;                 /* does tape have nblocks field? */
} ;
```

NAME

ARP - Address Resolution Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <net/if_arp.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

d = open ("/dev/arp", O_RDWR);
```

DESCRIPTION

ARP is a protocol used to map dynamically between Internet Protocol (IP) and 10Mb/s Ethernet addresses. It is used by all the 10Mb/s Ethernet datalink providers (interface drivers). It is not specific to the Internet Protocol or to the 10Mb/s Ethernet, but this implementation currently supports only that combination. The STREAMS device `/dev/arp` is not a Transport Level Interface (TLI) transport provider and may not be used with the TLI interface.

ARP caches IP-to-Ethernet address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message that requires the mapping and broadcasts a message on the associated network requesting the address mapping. If a response is provided, the new mapping is cached and any pending message is transmitted. ARP will queue at most one packet while waiting for a mapping request to be responded to; only the most recently transmitted packet is kept.

To facilitate communications with systems which do not use ARP, `ioctl` requests are provided to enter and delete entries in the IP-to-Ethernet tables.

USAGE

```
#include <sys/sockio.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/if_arp.h>
struct arpreq arpreq;
ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```

Each `ioctl` request takes the same structure as an argument. `SIOCSARP` sets an ARP entry, `SIOCGARP` gets an ARP entry, and `SIOCDARP` deletes an ARP entry. These `ioctl` requests may be applied to any Internet family socket descriptor *s*, or to a descriptor for the ARP device, but only by the privileged user. The `arpreq` structure contains:

```
/*
* ARP ioctl request
*/
struct arpreq {
      struct sockaddr arp_pa;     /* protocol address */
      struct sockaddr arp_ha;     /* hardware address */
      int             arp_flags;  /* flags */
};
/*  arp_flags field values */
```

```
#define ATF_COM          0x2  /* completed entry (arp_ha valid) */
#define ATF_PERM         0x4  /* permanent entry */
#define ATF_PUBL         0x8  /* publish (respond for other host) */
#define ATF_USETRAILERS  0x10 /* send trailer packets to host */
```

The address family for the arp_pa sockaddr must be AF_INET; for the arp_ha sockaddr it must be AF_UNSPEC. The only flag bits that may be written are ATF_PERM, ATF_PUBL and ATF_USETRAILERS. ATF_PERM makes the entry permanent if the ioctl request succeeds. The peculiar nature of the ARP tables may cause the ioctl request to fail if too many permanent IP addresses hash to the same slot. ATF_PUBL specifies that the ARP code should respond to ARP requests for the indicated host coming from other machines. This allows a host to act as an ARP server, which may be useful in convincing an ARP-only machine to talk to a non-ARP machine.

ARP is also used to negotiate the use of trailer IP encapsulations; trailers are an alternate encapsulation used to allow efficient packet alignment for large packets despite variable-sized headers. Hosts that wish to receive trailer encapsulations so indicate by sending gratuitous ARP translation replies along with replies to IP requests; they are also sent in reply to IP translation replies. The negotiation is thus fully symmetrical, in that either or both hosts may request trailers. The ATF_USETRAILERS flag is used to record the receipt of such a reply, and enables the transmission of trailer packets to that host.

ARP watches passively for hosts impersonating the local host (that is, a host which responds to an ARP mapping request for the local host's address).

SEE ALSO

arp(1M), ifconfig(1M), if(3N), inet(7)

Plummer, Dave, ''*An Ethernet Address Resolution Protocol -or- Converting Network Protocol Addresses to 48.bit Ethernet Addresses for Transmission on Ethernet Hardware*,'' RFC 826, Network Information Center, SRI International, Menlo Park, Calif., November 1982

Leffler, Sam, and Michael Karels, ''*Trailer Encapsulations*,'' RFC 893, Network Information Center, SRI International, Menlo Park, Calif., April 1984

NAME

asyhdlc - Asynchronous HDLC protocol module

SYNOPSIS

asyhdlc

DESCRIPTION

The asyhdlc module is pushed on a tty stream attached to an asynchronous serial line so that PPP may use that line to transmit and receive IP datagrams.

A PPP HDLC packet lacks a CRC checksum and uses a ''transparent code'' for data transmission. asyhdlc performs the following functions on PPP datagrams:

generates and validates the CRC checksum

encodes and decodes packet data to achieve data transparency - charater stuffing

generates and strips framing patterns delimiting packet start and end

See ppp(7) for additional information about the PPP implementation.

SEE ALSO

ppp(7)
RFC 1171

NAME

binarsys - remote system information for the ckbinarsys command

DESCRIPTION

binarsys contains lines of the form:

remote_system_name:*val*

where *val* is either Y or N. This line indicates whether that particular remote system can properly deal with messages having binary content. The absence of an entry for a particular system or absence of the binarsys file altogether will imply No.

Blank lines or lines beginning with # are considered comments and ignored. Should a line of Default=y be encountered, the default condition for missing entries described in the previous paragraph is reversed to be Yes. Another line of Default=n will restore the default condition to No.

mail is distributed with the binarsys file containing only a Default=y line.

FILES

/etc/mail/binarsys

SEE ALSO

ckbinarsys(1M), mail(1), mailsurr(4).

NAME

bootparams - boot parameter data base

SYNOPSIS

/etc/bootparams

DESCRIPTION

The bootparams file contains the list of client entries that diskless clients use for booting. For each diskless client, the entry should contain the following information:

name of client
a list of keys, names of servers, and pathnames

The first item of each entry is the name of the diskless client. The subsequent item is a list of keys, names of servers, and pathnames.

Items are separated by TAB characters.

EXAMPLE

This is an example of a /etc/bootparams entry:

```
myclient   root=myserver:/nfsroot/myclient\
      swap=myserver:/nfsswap/myclient\
      dump=myserver:/nfsdump/myclient
```

FILES

/etc/bootparams

SEE ALSO

bootparamd(1M)

NAME

cdrom - CDROM device support

DESRIPTION

CDROM disk drives perform like hard disk drives except for the following:

Read only
: CDROM disks are read-only devices. Any attempt to write to a CDROM disk results in an error (EROFS).

2048 Byte Blocks
: CDROM drives are accessed in multiples of 2048 bytes. All raw transfers must be aligned on 2048-byte boundaries and have a transfer byte count that is a multiple of 2048 bytes. If either of these conditions is not met, the I/O results in an error (EIO).

Slicing
: If a CDROM disk has a valid Motorola Volume ID, the Volume Table of Contents (VTOC) reads from the disk. If the CDROM disk does not have a valid volume ID, the VTOC consists of two slices: slice zero and slice seven. Slice zero is the first slice on a boot disk which always contains root. Slice seven represents the whole disk, whether it contains root or not.

Door Locking
: When no process currently has the CDROM drive open and it is being opened for the first time, the media-eject button on the drive becomes disabled until the last close, if the CDROM drive has a locking door.

Presence of Media
: If there is no CDROM in the drive, an open attempt results in an error (ENXIO).

IOCTL COMMANDS

CDROMs support several ioctl(2) functions on the character or raw devices. These functions permit control beyond the normal open(2), close(2), read(2), and write(2) system calls. All ioctl(2) operations take the form ioctl (*fildes, command, *arg)*. Any attempt to utilize ioctl(2) functions not listed below cause an EINVAL error to be returned.

The operations supported by CDROMs are listed below in alphabetical order.

DKGETCFG
: Get parameters associated with the disk and store them in the dkconfig structure referenced by *arg*. The disk is not accessed by this command.

DKGETINFO
: Get parameters associated with the disk and store them in the dkblk0 structure referenced by *arg*. The disk is not accessed by this command.

DKGETSLC
: Get the VTOC information for a disk and return the information in a structure of type struct motorola_vtoc (defined in sys/vtoc.h) referenced by *arg*. While the number of supported slices is determined by the number of slices defined in the ddefs file, all disks are expected to support 16 slices. The disk is not accessed by this command.

DKINQUIRY

Return the SCSI INQUIRY data for the device; it is only valid for SCSI CDROMs. This `ioctl` can be done on any device that the calling process has open. The SCSI INQUIRY data for the device is copied into the `struct inquiry` structure pointed to by *arg*. The `struct inquiry` structure is defined in `sys/dk.h`.

DKREADCAP

Return the SCSI READ CAPACITY data for the device; it is only valid for SCSI CDROMs. This `ioctl` can be done on any disk or CDROM device that the calling process has open. The SCSI READ CAPACITY data for the device is copied into the `struct readcap` structure pointed to by *arg*. The `struct readcap` structure is defined in `sys/dk.h`. Note that the SCSI READ CAPACITY command returns the number of the last logical block on the media. This `ioctl` adds one to that number so that it represents the actual capacity of the device (logical block numbers start at zero).

DKTRAY_OPEN

Cause the CDROM door to open after processing the last close (when no process has the drive open). The *arg* parameter is not used.

V_GETSSZ

Return the physical sector size of the CDROM. The *arg* parameter specifies a structure of type `io_arg` (defined in `sys/vtoc.h`). The `sectst` and `datasz` members of the `io_arg` structure are ignored. The `memaddr` member of the structure points to the address of an integer which contains the sector size after a successful operation.

V_PDREAD

Read the Physical Description Area of the disk. The *arg* parameter specifies a structure of type `io_arg` (defined in `sys/vtoc.h`). The `sectst` and `datasz` members of the `io_arg` structure are ignored. The `memaddr` member of the `io_arg` structure points to the address of a structure of type `pdsector` (defined in `sys/vtoc.h`) which contain the requested data upon successful completion.

V_PDWRITE

Write the Physical Description Area of the disk. This command always returns `EROFS`. The *arg* parameter specifies a structure of type `pdinfo` (defined in `sys/vtoc.h`).

V_PREAD

Read physical sectors. This interface assumes that sectors are 512 bytes in length so the driver is responsible for mapping the requested block(s) to the correct portion of the correct sector on the CDROM regardless of the actual physical sector size. The *arg* parameter specifies a structure of type `io_arg` (defined in `sys/vtoc.h`). The `sectst` member of the `io_arg` structure contains the starting sector number and the `datasz` member contains the number of sectors. The `memaddr` member of the `io_arg` structure points to the address of a sufficiently large area which contains the requested data upon successful completion.

V_PWRITE
Write physical sectors. This command always returns EROFS. The *arg* parameter specifies a structure of type io_arg (defined in sys/vtoc.h).

V_RVTOC
Read the VTOC from the disk. The *arg* parameter specifies a structure of type io_arg (defined in sys/vtoc.h). The sectst and datasz members of the io_arg structure are ignored. The memaddr member of the io_arg structure points to the address of a structure of type vtoc (defined in sys/vtoc.h) which contains the requested data upon successful completion.

V_WVTOC
Write the VTOC to the disk. This command always returns EROFS. The *arg* parameter specifies a structure of type vtoc (defined in sys/vtoc.h).

SEE ALSO

disk(7), floppy(7), intro(7)

NAME

clone - open any major/minor device pair on a STREAMS driver

DESCRIPTION

clone is a STREAMS software driver that finds and opens an unused major/minor device on another STREAMS driver. The major device number passed to clone during open corresponds to the clone driver and the minor device number corresponds to the target driver. Each open results in a separate stream to a previously unused major/minor device.

The clone driver consists solely of an open function. This open function performs all of the necessary work so that subsequent system calls [including close(2)] require no further involvement of clone.

clone will generate an ENXIO error, without opening the device, if the major/minor device number provided does not correspond to a valid major/minor device, or if the driver indicated is not a STREAMS driver.

SEE ALSO

log(7).

NOTES

Multiple opens of the same major/minor device cannot be done through the clone interface. Executing stat(2) on the file system node for a cloned device yields a different result from executing fstat(2) using a file descriptor obtained from opening the node.

NAME

compver - compatible versions file

DESCRIPTION

compver is an ASCII file used to specify previous versions of the associated package which are upward compatible. It is created by a package developer.

Each line of the file specifies a previous version of the associated package with which the current version is backward compatible.

Since some packages may require installation of a specific version of another software package, compatibility information is extremely crucial. Consider, for example, a package called "A" which requires version "1.0" of application "B" as a prerequisite for installation. If the customer installing "A" has a newer version of "B" (1.3), the compver file for "B" must indicate that "1.3" is compatible with version "1.0" in order for the customer to install package "A."

NOTES

The comparison of the version string disregards white space and tabs. It is performed on a word-by-word basis. Thus `1.3    Enhanced` and `1.3 Enhanced` would be considered the same.

EXAMPLE

A sample compver file is shown below.

```
1.3
1.0
```

SEE ALSO

depend(4)

NAME

connld - line discipline for unique stream connections

DESCRIPTION

connld is a STREAMS-based module that provides unique connections between server and client processes. It can only be pushed [see streamio(7)] onto one end of a STREAMS-based pipe that may subsequently be attached to a name in the file system name space. After the pipe end is attached, a new pipe is created internally when an originating process attempts to open(2) or creat(2) the file system name. A file descriptor for one end of the new pipe is packaged into a message identical to that for the ioctl I_SENDFD [see streamio(7)] and is transmitted along the stream to the server process on the other end. The originating process is blocked until the server responds.

The server responds to the I_SENDFD request by accepting the file descriptor through the I_RECVFD ioctl message. When this happens, the file descriptor associated with the other end of the new pipe is transmitted to the originating process as the file descriptor returned from open(2) or creat(2).

If the server does not respond to the I_SENDFD request, the stream that the connld module is pushed on becomes uni-directional because the server will not be able to retrieve any data off the stream until the I_RECVFD request is issued. If the server process exits before issuing the I_RECVFD request, the open(2) or the creat(2) system calls will fail and return -1 to the originating process.

When the connld module is pushed onto a pipe, messages going back and forth through the pipe are ignored by connld.

On success, an open of connld returns 0. On failure, errno is set to the following values:

EINVAL	A stream onto which connld is being pushed is not a pipe or the pipe does not have a write queue pointer pointing to a stream head read queue.
EINVAL	The other end of the pipe onto which connld is being pushed is linked under a multiplexor.
EPIPE	connld is being pushed onto a pipe end whose other end is no longer there.
ENOMEM	An internal pipe could not be created.
ENXIO	An M_HANGUP message is at the stream head of the pipe onto which connld is being pushed.
EAGAIN	Internal data structures could not be allocated.
ENFILE	A file table entry could not be allocated.

SEE ALSO

streamio(7).

NAME

cons1x7 - hardware specific console driver for the MVME1X7 family

DESCRIPTION

This STREAMS-based driver provides console I/O when the system is running on an MVME1X7 CPU board. This driver is accessable only through the standard console device special files /dev/console (/dev/contty00), /dev/contty (/dev/contty01), /dev/contty02, /dev/contty03, and /dev/conctl.

The device special files eventually access the STREAMS-based console driver which, when used in conjunction with the STREAMS line discipline module ldterm, supports the termios(2) and termio(7) processing.

The configurable parameter C1X7_TXFIFO_MAX has a default of 8 and is located in the driver master.d file. This parameter describes the maximum number of bytes which should be written to the CD2400 transmit FIFO each time the FIFO is filled. Values 1 through 15 inclusive are valid. Increasing this parameter decreases the number of interrupts taken as a result of any of the serial data lines on the MVME1X7. The characters may be placed in the FIFO at an interrupt priority and may slow the response time of the system if large amounts of data are being sent through the onboard serial lines. If an invalid value is chosen for this parameter, it is reset to the default value and a warning message is printed to the system console.

In addition to the IOCTLs supported in termio(7), three other IOCTLs are supported. See the **USAGE** section for IOCTL details.

USAGE

STREAM Message Processing

In addition to the IOCTLs listed in termio(7), the following IOCTLs are supported. The definitions for the IOCTLs are in the file /usr/include/sys/cd2400.h.

M_IOCTL

MSETHWHAND causes the driver to enable out-of-band flow control using CTS(Clear to Send). This causes character transmission to begin only after CTS is active(low). If a console port is in aysnchronous mode, then when CTS goes inactive(high) after transmission has started, the channel stops transmitting after the current characters in the transmit hold register and shift register are transmitted. When in synchronous mode and CTS goes inactive, then the channel stops transmission after the current frame. Transmission restarts after CTS goes active. Also, MSETHWHAND sets a receive FIFO threshold of 10 characters. Automatic hardware flow control(DTR/DSR) activates when the FIFO threshold is reached.

MCLEARWHAND causes the driver to clear the flow controls set by MSETHWHAND. The hardware then returns to the no flow control state.

MGETHWHAND causes the driver to return the current status of CTS and DTR/DSR hardware flow control. The driver returns a data structure of type HWhandshake. HWhandshake is defined in the file /usr/include/sys/cd2400.h. HWhandshake.stat will equal HDFLOW_ENABLED if flow control is on and HDFLOW_DISABLED if it is off.

An example of code to implement each IOCTL is listed below:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termio.h>
#include <termios.h>
#include <sys/cd2400.h>
#include <stropts.h>

struct strioctl command ;

int sethwhandshake(int fd)
{
      int err=0 ;

      command.ic_cmd = MSETHWHAND ;
      command.ic_len = 0 ;
      command.ic_dp = NULL ;

      if ( ioctl(fd, I_STR, &command) < 0 ) {
         printf("ioctl error sending command to console driver") ;
         err = -1 ;
      }

      return(err) ;
}

int clearhwhandshake(int fd)
{
      int err=0 ;

      command.ic_cmd = MCLEARHWHAND ;
      command.ic_len = 0 ;
      command.ic_dp = NULL ;

      if ( ioctl(fd, I_STR, &command) < 0 ) {
         printf("ioctl error sending command to console driver") ;
         err = -1 ;
      }

      return(err) ;
}


int gethwhandshake(int fd)
{
      int err=0 ;
```

```
        HWhandshake shake;

        command.ic_cmd = MGETHWHAND ;
        command.ic_len = sizeof(shake) ;
        command.ic_dp = (char *) &shake ;

        if ( ioctl(fd, I_STR, &command) < 0 ) {
           printf("ioctl error sending command to driver") ;
           err = -1 ;
        }

        if (shake.stat == HDFLOW_DISABLED )
           printf("Hardware handshake is DISABLED") ;

        if (shake.stat == HDFLOW_ENABLED )
           printf("Hardware handshake is ENABLED") ;

        return(err) ;
}
```

FILES

/dev/console
/dev/contty
/dev/contty??
/dev/conctl
/usr/include/sys/cd2400.h
/usr/include/sys/cons1x7.h

SEE ALSO

dcon(1A), mvmecpu(1M), termios(2), console(7), iuart(7), ldterm(7), termio(7).

NAME

console - STREAMS-based console interface

DESCRIPTION

/dev/console and /dev/contty00 are synonyms for the system console and refer to an asynchronous serial data line originating from the system board.

For security reasons, the permissions on /dev/console are set to 620, restricting writer access by group and other. This will cause applications writing to /dev/console to fail. If you have such an application, change the permissions on /dev/console as follows:

```
/bin/chmod 666 /dev/console
```

/dev/contty and /dev/contty01 refer to a second asynchronous serial data line originating from the system board. /dev/contty02 and /dev/contty03 refer to a third and fourth serial data line originating from the system board. These serial data lines are only available on the MVME187 and MVME167 CPU boards.

/dev/conctl is the console control port.

These device special files access the STREAMS-based console driver which, when used in conjunction with the STREAMS line discipline module ldterm, supports the termios(2) and termio(7) processing.

FILES

/dev/console
/dev/contty
/dev/contty??
/dev/conctl

SEE ALSO

crash(1M), dcon(1M), mvmecpu(1M), termios(2), cons1x7(7), iuart(7), ldterm(7), termio(7).

NAME

`copyright` - copyright information file

DESCRIPTION

`copyright` is an ASCII file used to provide a copyright notice for a package. The text may be in any format. The full file contents (including comment lines) is displayed on the terminal at the time of package installation.

NAME

core - core image file

DESCRIPTION

The UNIX system writes out a core image of a process when it is terminated due to the receipt of some signals. The core image is called core and is written in the process's working directory (provided it can be; normal access controls apply). A process with an effective user ID different from the real user ID will not produce a core image.

The core file contains all the process information pertinent to debugging: contents of hardware registers, process status and process data. The format of a core file is object file specific.

For ELF executable programs [see a.out(4)], the core file generated is also an ELF file, containing ELF program and file headers. The e_type field in the file header has type ET_CORE. The program header contains an entry for every loadable and writeable segment that was part of the process address space, including shared library segments. The contents of the segments themselves are also part of the core image.

The program header of an ELF core file also contains a NOTE segment. This segment may contain the following entries. Each has entry name "CORE" and presents the contents of a system structure:

prstatus_t
: The entry containing this structure has a NOTE type of 1. This structure contains things of interest to a debugger from the operating system's u-area, such as the general registers, signal dispositions, state, reason for stopping, process ID and so forth. The structure is defined in sys/procfs.h.

prpsinfo_t
: The entry containing this structure has a NOTE type of 3. It contains information of interest to the ps(1) command, such as process status, cpu usage, "nice" value, controlling terminal, user ID, process ID, the name of the executable and so forth. The structure is defined in sys/procfs.h.

For 68k only COFF executable programs produce core files consisting of two parts: the first section is a copy of the system's per-user data for the process, including the general registers. The format of this section is defined in the header files sys/user.h and sys/reg.h. The remainder of a COFF core image represents the actual contents of the process data space.

For 88k only COFF executable programs produce core files in the following format (data structures are defined in sys/ptrace.h):

a struct ptrace_user containing the current status of the process

one struct pt_mem_desc for each shared memory segment attached to the process

one struct pt_mem_desc for each shared library data segment attached to the process the process's data segment

the process's stack segment

the contents of the shared memory and shared library data segments referred to by the pt_mem_desc entries

The size of the core file created by a process may be controlled by the user [see getrlimit(2)].

SEE ALSO

crash(1M), tbx(1), getrlimit(2), setuid(2), elf(3E), a.out(4), signal(5).

NAME

`depend` - software dependencies files

DESCRIPTION

`depend` is an ASCII file used to specify information concerning software dependencies for a particular package. The file is created by a software developer.

Each entry in the `depend` file describes a single software package. The instance of the package is described after the entry line by giving the package architecture and/or version. The format of each entry and subsequent instance definition is:

type pkg name
    (arch)version
    (arch)version
    ...

The fields are:

type — Defines the dependency type. Must be one of the following characters:

- P — Indicates a prerequisite for installation, for example, the referenced package or versions must be installed.
- I — Implies that the existence of the indicated package or version is incompatible.
- R — Indicates a reverse dependency. Instead of defining the package's own dependencies, this designates that another package depends on this one. This type should be used only when an old package does not have a `depend` file but it relies on the newer package nonetheless. Therefore, the present package should not be removed if the designated old package is still on the system since, if it is removed, the old package will no longer work.

pkg — Indicates the package abbreviation.

name — Specifies the full package name.

(arch)version — Specifies a particular instance of the software. A version name cannot begin with a left parenthesis. The instance specifications, both *arch* and *version*, are completely optional but must each begin on a new line that begins with white space. A null version set equates to any version of the indicated package.

EXAMPLE

Here is a sample depend file:

```
I msvr M68K Messaging Server
P ctc Cartridge Tape Utilities
P dfm Directory and File Management Utilities
P ed Editing Utilities
P ipc Inter-Process Communication Utilities
P lp Line Printer Spooling Utilities
P shell Shell Programming Utilities
P sys System Header Files
           Release 3.0
P sysadm System Administration Utilities
P term Terminal Filters Utilities
P terminfo Terminal Information Utilities
P usrenv User Environment Utilities
P uucp Basic Networking Utilities
P x25 X.25 Network Interface
           Issue 1 Version 1
           Issue 1 Version 2
P windowing AT&T Windowing Utilities
           (M68k)Version 1
R cms M68k Call Management System
```

NAME

device-map - script for makedev

DESCRIPTION

The /etc/device-map file controls the assignment of generic device names for system administration and generic use.

The /etc/device-map file contains two kinds of lines: comment lines and assignment lines.

1. Any line starting with the # character is assumed to be a comment.
2. An assignment line consists of two fields separated by white space (tab or space characters). The first field specifies the generic device type (for example, ctape, disk, ninetrack). The second field contains the controller-specific name of the device that will be assigned that generic name (for example, /dev/rmt/m328_c0d0).

The generic device number is assigned automatically, based on the position of the assignment line relative to other generic assignment of that type.

If the controller-specific device does not exist or is of the incorrect type, the assignment line is ignored. Processing continues on other legal assignment lines.

A partial example of an /etc/device-map file is presented below:

```
          .
          .
#   Cartridge tapes devices
ctape /dev/rmt/m328_c0d0
ctape /dev/rmt/m328_c0d4
          .
          .
```

SEE ALSO

makedev(1M)

NAME

dfstab - file containing commands for sharing resources

DESCRIPTION

dfstab resides in directory /etc/dfs and contains commands for sharing resources across a network. dfstab gives a system administrator a uniform method of controlling the automatic sharing of local resources.

Each line of the dfstab file consists of a share(1M) command. The dfstab file can be read by the shell directly to share all resources, or system administrators can prepare their own shell scripts to execute particular lines from dfstab.

The contents of dfstab are executed automatically when the system enters run level 3.

SEE ALSO

share(1M), shareall(1M)

NAME

dir (generic) - format of directories

DESCRIPTION

Directory format is entirely *FSType*-specific. See dir_*FSType*(4) for information.

SEE ALSO

dir_s5(4), dir_ufs(4).

NAME

dir (s5) - format of s5 directories

SYNOPSIS

```
#include <sys/types.h>
#include <sys/fs/s5dir.h>
```

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the mode word of its i-node entry [see the s5-specific inode(4)]. The structure of a directory entry as given in the include file is:

```
#ifndef  DIRSIZ
#define  DIRSIZ  14
#endif
struct direct
{
     o_ino_t     d_ino;      /* s5 inode type */
     char        d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are . for the entry itself and .. for the parent directory. The meaning of .. is modified for the root directory of the master file system; there is no parent, so .. has the same meaning as . has.

SEE ALSO

s5_specific inode(4)

NAME

dir (ufs) - format of ufs directories

SYNOPSIS

```
#include <sys/param.h>
#include <sys/types.h>
#include <sys/fs/ufs_fsdir.h>
```

DESCRIPTION

A directory consists of some number of blocks of DIRBLKSIZ bytes, where DIRBLKSIZ is chosen such that it can be transferred to disk in a single atomic operation (for example, 512 bytes on most machines).

Each DIRBLKSIZ-byte block contains some number of directory entry structures, which are of variable length. Each directory entry has a struct direct at the front of it, containing its inode number, the length of the entry, and the length of the name contained in the entry. These are followed by the name padded to a 4 byte boundary with null bytes. All names are guaranteed null-terminated. The maximum length of a name in a directory is MAXNAMLEN.

```
#define DIRBLKSIZ    DEV_BSIZE
#define MAXNAMLEN    256
struct     direct {
    u_long    d_ino;                    /* inode number of entry */
    u_short   d_reclen;                 /* length of this record */
    u_short   d_namlen;                 /* length of string in d_name */
    char      d_name[MAXNAMLEN + 1];    /* name must be no longer than this */
};
```

SEE ALSO

ufs-specific fs(4)

NAME

dirent - file system independent directory entry

SYNOPSIS

```
#include <dirent.h>
```

DESCRIPTION

Different file system types may have different directory entries. The dirent structure defines a file system independent directory entry, which contains information common to directory entries in different file system types. A set of these structures is returned by the getdents(2) system call.

The dirent structure is defined below.

```
struct  dirent {
        ino_t           d_ino;
        off_t           d_off;
        unsigned short  d_reclen;
        char            d_name[1];
};
```

The d_ino is a number which is unique for each file in the file system. The field d_off is the offset of that directory entry in the actual file system directory. The field d_name is the beginning of the character array giving the name of the directory entry. This name is null terminated and may have at most MAXNAMLEN characters. This results in file system independent directory entries being variable length entities. The value of d_reclen is the record length of this entry. This length is defined to be the number of bytes between the current entry and the next one, so that the next structure will be suitably aligned.

SEE ALSO

getdents(2)

NAME

disk - disk support

DESCRIPTION

All Motorola disks support dynamic slice sizing. The Volume Table of Contents (VTOC) contains the slicing information for the disk. Up to 16 slices may be specified. Therefore, you do not have to configure the size and slicing of a disk into the driver. You can attach any size disk without changing any configuration information.

The raw device nodes /dev/rdsk/prefix_* allow the transfer of a specified number of bytes in multiples of sector size between the hard disk drive and a location in the user's address space. The typical number of bytes in a sector is 512.

Disk devices may be removable or non-removable (fixed).

IOCTL COMMANDS

Disk drivers support several ioctl(2) functions on the character or raw devices. These functions permit control beyond the normal open(2), close(2), read(2), and write(2) system calls. All ioctl(2) operations take the form ioctl (*fildes, command, *arg*). Any attempt to utilize ioctl(2) functions not listed below causes an EINVAL error to be returned.

The operations supported by disks are listed below in alphabetical order.

DKFIXBADSPOT
: Lock out a bad spot on the disk based on the information in the dkbadlst structure referenced by *arg*. The dkbadlst structure is defined in sys/dk.h.

DKFORMAT
: Format a disk. The dkfmt structure is defined in sys/dk.h.

DKGETCFG
: Get parameters associated with the disk and store them in the dkconfig structure referenced by *arg*. The dkconfig structure is defined in sys/dk.h. The disk is not accessed by this command.

DKGETINFO
: Get parameters associated with the disk and store them in the dkblk0 structure referenced by *arg*. The dkblk0 structure is defined in sys/dk.h. The disk is not accessed by this command.

DKGETSLC
: Get the VTOC information for a disk and return the information in a structure of type struct motorola_vtoc (defined in sys/vtoc.h) referenced by *arg*. While the number of supported slices is determined by the number of slices defined in the ddefs file, all disks are expected to support 16 slices. The disk is not accessed by this command.

DKSETCFG
: Get parameters associated with the disk and store them in the dkconfig structure referenced by *arg*. The disk is not accessed by this command.

DKSETINFO
: Set parameters associated with the disk based on the values in the dkblk0 structure referenced by *arg*. The disk is not accessed by this command.

DKSETSLC
: Set the VTOC information for a disk and return the information in a structure of type `struct motorola_vtoc` (defined in `sys/vtoc.h`) referenced by *arg*. The disk is not accessed by this command.

DKINQUIRY
: Return the SCSI INQUIRY data for the device; it is only valid for SCSI disks. This `ioctl` can be done on any device the calling process has open. The SCSI INQUIRY data for the device is copied into the `struct inquiry` structure pointed to by *arg*. The `struct inquiry` structure is defined in `sys/dk.h`.

DKREADCAP
: Return the SCSI READ CAPACITY data for the device; it is only valid for SCSI disks. This `ioctl` can be done on any disk or CDROM device the calling process has open. The SCSI READ CAPACITY data for the device is copied into the `struct readcap` structure pointed to by *arg*. The `struct readcap` structure is defined in `sys/dk.h`. Note: the SCSI READ CAPACITY command returns the number of the last logical block on the media. This `ioctl` adds one to that number so it represents the actual capacity of the device. Logical block numbers start at zero.

V_GETSSZ
: Return the physical sector size of the CDROM. The *arg* parameter specifies a structure of type `io_arg` (defined in `sys/vtoc.h`). The `sectst` and `datasz` members of the `io_arg` structure are ignored. The `memaddr` member of the structure points to the address of an integer containing the sector size after a successful operation.

V_PDREAD
: Read the Physical Description Area of the disk. The *arg* parameter specifies a structure of type `io_arg` (defined in `sys/vtoc.h`). The `sectst` and `datasz` members of the `io_arg` structure are ignored. The `memaddr` member of the `io_arg` structure points to the address of a structure of type `pdsector` (defined in `sys/vtoc.h`) containing the requested data upon successful completion.

V_PDWRITE
: Write the Physical Description Area of the disk. The *arg* parameter specifies a structure of type pdinfo (defined in `sys/vtoc.h`). The `sectst` and `datasz` members of the `io_arg` structure are ignored. The `memaddr` member of the `io_arg` structure points to the address of a structure of type `pdsector` (defined in `sys/vtoc.h`) containing the requested data upon successful completion.

V_PREAD
: Read physical sectors. This interface assumes sectors are 512 bytes in length so the driver is responsible for mapping the request block to the correct portion of the correct sector on the disk regardless of the actual physical sector size. The *arg* parameter specifies a structure of type `io_arg` (defined in `sys/vtoc.h`). The `sectst` member of the `io_arg` structure contains the starting sector number and the `datasz` member contains the number of sectors. The `memaddr` member of the `io_arg` structure points to the address of a sufficiently large area containing the requested data upon successful

completion.

V_PWRITE

Write physical sectors. This interface assumes sectors are 512 bytes in length so the driver is responsible for mapping the requested block(s) to the correct portion of the correct sector on the disk regardless of the actual physical sector size. The *arg* parameter specifies a structure of type `io_arg` (defined in `sys/vtoc.h`). The `sectst` member of the `io_arg` structure contains the starting sector number and the `datasz` member contains the number of sectors. The `memaddr` member of the `io_arg` structure points to the address of a sufficiently large area containing the requested data upon successful completion.

V_RVTOC

Read the VTOC from the disk. The *arg* parameter specifies a structure of type `io_arg` (defined in `sys/vtoc.h`). The `sectst` and `datasz` members of the `io_arg` structure are ignored. The `memaddr` member of the `io_arg` structure points to the address of a structure of type `vtoc` (defined in `sys/vtoc.h`) containing the requested data upon successful completion.

V_WVTOC

Write the VTOC to the disk. The *arg* parameter specifies a structure of type `vtoc` (defined in `sys/vtoc.h`). The `sectst` and `datasz` members of the `io_arg` structure are ignored. The `memaddr` member of the `io_arg` structure points to the address of a structure of type `vtoc` (defined in `sys/vtoc.h`) containing the requested data upon successful completion.

DINIT CONSIDERATIONS

The utility `dinit`(1M) initially formats the disk and fixes any new bad spots occurring over time. Although a device driver redirects all future operations away from new bad spots, any existing data in the bad block is lost. Always use the `-s` option to `dinit` when attempting to fix new bad spots.

DDEFS CONSIDERATIONS

The utility `ddefs` defines disk characteristics. The output of the `ddefs` utility is a file normally saved in the `/etc/dskdefs` directory. This file is used as input to the `dinit`(1M) utility when it initializes a disk.

A brief description of the important fields follows.

Comment

Identification of the `ddefs` file to the user.

Disk type

Decimal equivalent of a two-byte field. Upper byte is the SCSI controller type; lower byte is the peripheral type. This field is not currently used by the MVME328 and SCSI1X7 drivers. Valid disk types are:

DISK	CONTROLLER TYPE	PERIPHERAL TYPE	DISK TYPE
mcdcIII	0x13	0x02	4098
mcdcIV	0x13	0x02	4866
mcdcV	0x13	0x02	4866
mcdcVII	0x13	0x02	4866
mfuj2613	0x13	0x02	4866
mfuj2614	0x13	0x02	4866
mfuj2624	0x13	0x02	4866

Format command
: Used by `dinit`(1M) for formatting. It is set to `none` for the MVME328 and SCSI1X7.

Diagnostic tracks
: Used by `dinit`(1M) to write diagnostic tracks on the disk. The default value for the MVME328 and SCSI1X7 is `no`.

Bad spot strategy
: The MVME328 and SCSI1X7 drivers consider all media as PERFECT.

Maximum number of bad spots
: The maximum number of new bad spots that can be added.

Number of sectors
: The total number of sectors on the disk.

Sector size
: The physical sector size of the disk.

Sectors per track
: The number of sectors per track on the disk.

Cylinders
: The total number of cylinders on the disk.

Heads
: The number of heads on the disk.

The following fields are not used by the MVME328 and SCSI1X7: Precompensation cylinder, Sector interleave, Spiral offset, Step rate, Starting head number, ECC error length, Attributes mask, Extended attributes mask, Attributes word, Gap byte 1, Gap byte 2, Gap byte 3, Gap byte 4, and Unformatted sector size.

Controller Attributes Word
: Identifies various characteristics of the disk controller configuration, as shown in the following table:

DEFINITION	SET(1)	RESET(0)
0x01000	Don't stop format if p/g list inaccessible	Stop format if inaccessible
0x10000	Don't turn on drive cache	Turn on drive cache
0x00800	Defect management zone = cylinder	Defect management zone = track

These are the only flags currently used by the MVME328 and SCSI1X7 device drivers.

Sector slip count
: Indicates the number of spare sectors to be reserved for the defined defect management zone. Note: changing this value can affect the usable capacity of the drive.

The following `ddefs` utility fields are ignored: `root` file system offset, `root` file system size, `/usr` file system size, `/usr` file system slice, swap size, and swap slice. The following `ddefs` utility fields have values entered based on how the disk is to be used: `slice count` and end-of-disk reserved area.

Alternates
: This number is multiplied by the number of heads to determine the number of spare tracks to be reserved at the end of the drive for defect management. Note: changing this value can affect the usable capacity of the drive.

SEE ALSO

`cdrom`(7), `floppy`(7), `intro`(7)

NAME

dlce - Data Link / Common Environment interface

SYNOPSIS

```
#include <sys/dlpi.h>
#include <sys/dlce.h>

fd = open("/dev/dlce0", O_RDWR);
```

DESCRIPTION

The dlce is a STREAMS-based cloned software driver used with the MVME374 Ethernet board/driver. The dlce interface conforms to the Data Link Provider Interface (DLPI).

The dlce driver can be opened directly, or indirectly from the clone device driver. During the TCP/IP startup, the dlce device is opened and linked to the IP and ARP STREAMS modules via the slink command. From then on, dlce converts all the outgoing packets, received from IP/ARP, to the format defined by Common Environment/BPP interface and passes these packets to the MVME374 driver (which is currently named MVME37X).

Upon receiving incoming packets from the MVME374 driver, dlce converts these packets to the STREAMS-based DLPI format messages and passes these packets to IP/ARP.

When the MVME37X package is installed, the postinstall script in the package creates the device nodes for the DLCE driver. The name of a device node is composed of the string "dlce" followed by the board number (0 or 1) of the MVME374 which the DLCE driver is associated with. The board number must be the same as the MVME374's cpu number minus 2 (cpu 0 and 1 are reserved for the Common Environment and the local cpu). For instance, an MVME374 with cpu 2 (as defined in the edt_data file), would have a device name of /dev/dlce0.

A dlce node major device number is the major device number of the clone device driver. A dlce minor device number is the major number of the dlce device, found in /etc/master.d/dlce, concatenated with the board number corresponding to this device. See intro(7) for the pictorial representation of the minor device number as passed to the device driver. For the dlce device driver, the bit fields in the minor format are defined as:

> The BOARD bits define the board device number. Boards are numbered from 0. The maximum board device number supported is 1.
>
> The MAJOR # bits correspond to the real major number of the dlce device as specified in the file /etc/master.d/dlce.

The device node name is also used as the Ethernet network interface name by cenet in the network database file /etc/strcf and by ifconfig in the script /etc/inet/rc.inet.

Each dlce device may have up to four (4) minor devices open simultaneously. This number is configurable by modifying the #DEV field in /etc/master.d/dlce.

USAGE

STREAM Message Processing

The following are the types of STREAMS messages the driver can process:

M_PROTO/M_PCPROTO

Four DLPI protocol messages are supported: DL_INFO_REQ, DL_UNITDATA_REQ, DL_BIND_REQ, and DL_UNBIND_REQ,. Unsupported message types that are received are ignored and the STREAM message is freed.

DL_INFO_REQ is a request for driver information. Driver information is passed back up the stream in a message of type dl_info_ack_t with dl_primitive set to DL_INFO_ACK. However, if enough memory is not available for the driver information, an error message of type dl_error_ack_t is sent back up the stream with dl_primitive set to DL_ERROR_ACK.

DL_UNITDATA_REQ is a request to transmit data. The message is in the dl_unitdata_req_t format. The driver will process this message and send data to the appropriate destination address. Most errors that can occur during this message are turned around in the message itself and sent back up stream in a message with dl_primitive set to DL_UDERROR_IND. If enough memory is not available for processing, an error message of type dl_error_ack_t is sent back up the stream with dl_primitive set to DL_ERROR_ACK.

DL_BIND_REQ is a request to bind a service access point (SAP) to the minor device number associated with the current stream. The request message is of type dl_bind_req_t. Once the stream has been bound, an acknowledgement message type dl_bind_ack_t is sent back up the stream. Errors generated during the processing of this message that cause an error message of type dl_error_ack_t to be sent back up the stream are: stream already bound, bad sap value, and cannot allocate memory for acknowledgement. Currently, the only SAPs supported by dlce are IP_SAP and ARP_SAP; IEEE802.3 frames are not supported.

DL_UNBIND_REQ is a request to unbind the minor device associated with the current stream. Errors generated during message processing that cause an error message of type dl_error_ack_t are: minor device is not bound and cannot allocate enough memory for acknowledgement. An acknowledgement message of type dl_ok_ack_t is generated when the stream has been unbound.

M_IOCTL

ioctl commands are received in messages of type iocblk. Command data must be stored in a connected message block type M_DATA. Some commands do not require M_DATA blocks; M_DATA block requirements are listed. Data passed back upstream is always contained in an M_DATA block.

A description of user ioctl stream messages can be found under the I_STR command in streamio(7). A sample code extract can be found in the *STREAMS Mechanism* chapter of the *STREAMS Programming Guide*.

SIOCGENADDR is a type of request to return the Ethernet address of the LANCE controller associated with the current queue. This command requires an M_DATA block of type `struct ifreq`.

`M_FLUSH`
: If the command is a read queue flush, the read queue of the driver is flushed and the message is passed back up stream. If the command is a write queue flush, the write queue of the driver is flushed.

FILES

`/dev/dlce_*`
`/usr/include/sys/dlpi.h`
`/usr/include/sys/dlce.h`
`/usr/include/sys/dlcecommon.h`
`/usr/include/sys/dlceuser.h`

SEE ALSO

`ifconfig`(1M), `slink`(1M), `strace`(1M), `edt_data`(4), `master`(4), `strcf`(4N), `arp`(7), `clone`(7), `intro`(7), `ip`(7), `streamio`(7)
Programmer's Guide: STREAMS
McGrath, G., *A STREAMS-based Data Link Provider Interface (DLPI)*, Version 1.3, AT&T Bell Laboratories, Summit, N.J., February 1989

NAME

e1x7 - MVME1X7 Local Area Network Interface

SYNOPSIS

```
#include <sys/dlpi.h>
#include <sys/macioctl.h>

fd = open("/dev/e1x7_c0d0", O_RDWR);
```

DESCRIPTION

The MVME1X7 on-board Intel LANC chip (82596CA) is a Local Area Network Controller for Ethernet and IEEE 802.3 compatible networks. The LANC can handle all IEEE802.3 Medium Access Control and channel interface functions. The e1x7 device driver supports TCP/IP and OSI protocol stacks.

The e1x7 is a STREAMS-based driver used with MVME1X7 cpu boards. The e1x7 interface conforms to the Data Link Provider Interface (DLPI). In addition, the e1x7 driver accepts the MAC management commands specified in the MAC Provider Interface (MPI). To account for possible cpu board expansion, the driver data structures are designed to accomodate more than one LANC controller on a single cpu board via changes to the edt_data and master.d files.

The e1x7 driver can be opened directly or indirectly from the clone device driver. During TCP/IP startup, the e1x7 device is clone opened and linked to the IP and ARP STREAMS modules via the slink command. From then on, e1x7 converts all the outgoing packets received from IP/ARP to the format defined by the LANC controller and then passes these packets to the chip. If the OSI-DP package is installed on the system and linked into the kernel, the e1x7 driver will accept outgoing packets from the DLR (OSI LLC1) module.

Upon receiving incoming packets from the LANC controller, e1x7 converts these packets to STREAMS-based DLPI format messages and passes these packets to the appropriate user (e.g., ARP, IP, or DLR).

The mvmecpu namer program, creates or deletes the device special files for the e1x7 driver at boot time. The device special filenames are composed of the string e1x7_c*y*d*z*, where *y* is the controller number and *z* is the minor device number. Controllers are numbered beginning at 0. The device special filename for the first controller in the system is /dev/e1x7_c0d0, for the second controller (if the cpu board has one) is /dev/e1x7_c1d0, and so on.

An e1x7 device special file major device number is the major device number of the clone device driver. An e1x7 minor device number is the major number of the e1x7 device, found in /etc/master.d/enet1x7, concatenated with the board number corresponding to this device. See intro(7) for the pictorial representation of the minor device number as passed to the device driver. For the e1x7 device driver, the bit fields in the minor format are defined as:

> The BOARD bits define the controller device number. Controllers are numbered from 0. The maximum controller device number supported is 1, i.e., two controllers.
>
> The MAJOR # bits correspond to the real (external) major number of the e1x7 device as specified in the file /etc/master.d/enet1x7.

The device special filename is also used as the Ethernet network interface name by cenet in the network database file /etc/strcf and by ifconfig in the script /etc/inet/rc.inet.

Each e1x7 device may have up to seven (7) minor devices open simultaneously.

USAGE

STREAM Message Processing

The following are the types of STREAMS messages the driver can process:

M_PROTO/M_PCPROTO

Six DLPI protocol message types are supported: DL_INFO_REQ, DL_UNITDATA_REQ, DL_BIND_REQ, DL_UNBIND_REQ, DL_ENABMULTI_REQ, and DL_DISABMULTI_REQ. Unsupported message types that are received cause an error message of type dl_error_ack_t with dl_errno set to DL_NOTSUPPORTED to be sent back up the stream.

DL_INFO_REQ is a request for driver information. Driver information is passed back up the stream in a message of type dl_info_ack_t with dl_primitive set to DL_INFO_ACK. However, if enough memory is not available for the driver information, an error message of type dl_error_ack_t is sent back up the stream with dl_primitive set to DL_ERROR_ACK.

DL_UNITDATA_REQ is a request to transmit data. The message is in the dl_unitdata_req_t format. The driver will process this message and send data to the appropriate destination address. Most errors that can occur during this message are turned around in the message itself and sent back up stream in a message with dl_primitive set to DL_UDERROR_IND. If enough memory is not available for processing, an error message of type dl_error_ack_t is sent back up the stream with dl_primitive set to DL_ERROR_ACK.

DL_BIND_REQ is a request to bind a service access point (SAP) to the minor device number associated with the current stream. The request message is of type dl_bind_req_t. A SAP type, as long as it is valid, is assumed to be an Ethernet binding if it is not equal to IEEE8023_TYPE. Any Ethernet type can be used as a binding SAP. Only one stream may use IEEE8023_TYPE as a SAP. All IEEE802.3 frames will be sent up this stream. If the OSI-DP package has been installed, the DLR module will bind to this SAP and will receive all 802.3 frames. Once the stream has been bound, an acknowledgement message type dl_bind_ack_t is sent back up the stream. Errors generated during the processing of this message that cause an error message of type dl_error_ack_t to be sent back up the stream are: stream already bound, bad sap value, and cannot allocate memory for acknowledgement.

DL_UNBIND_REQ is a request to unbind the minor device associated with the current stream. Errors generated during message processing that cause an error message of type dl_error_ack_t are: minor device is not bound and cannot allocate enough memory for acknowledgement. An acknowledgement message of type dl_ok_ack_t is generated when the stream has been unbound.

DL_ENABMULTI_REQ is a request to enable a multicast address on a per-stream basis. An individual stream may have a maximum of sixty-four multicast addresses in its table, subject to the following limitation. There may be no more than sixty-four unique addresses for all streams associated with each controller. An acknowledgement message of type dl_ok_ack_t is generated if the request is valid. A message of type dl_error_ack_t is generated with dl_primitive set to DL_BADADDR if the multicast address is invalid or dl_primitive set to DL_TOOMANY if there is no space left in the controller's multicast table.

DL_DISABMULTI_REQ is a request to disable a multicast address on a per-stream basis. The driver will not accept frames with this multicast address even if e1x7multi_all is enabled and the LANC is accepting multicast addresses. An acknowledgement message of type dl_ok_ack_t is generated if the request is valid. A message of type dl_error_ack_t is generated with dl_primitive set to DL_BADADDR if the multicast address is invalid or dl_primitive set to DL_NOTENAB if the requested address is not currently enabled.

M_IOCTL

ioctl commands are received in messages of type iocblk. There are many ioctl commands supported by the driver. Command data must be stored in a connected message block type M_DATA. Some commands do not require M_DATA blocks; M_DATA block requirements are listed. Data passed back upstream is always contained in an M_DATA block. All of the ioctl #defines used can be found in the file include/sys/macioctl.h.

A description of user ioctl stream messages can be found under the I_STR command in streamio(7). A sample code extract can be found in the *STREAMS Mechanism* chapter of the *STREAMS Programming Guide*.

MACDELAMCA is a request to delete all multicast table entries on the controller associated with this stream. This command does not require an M_DATA block. The driver will not accept any multicast frames even if e1x7multi_all is enabled and the LANC is accepting multicast addresses.

MACDELMCA is a request to delete one multicast address from a multicast table on a per-stream basis. This command requires an M_DATA block of type mc_frame. The driver will not accept frames with this multicast address even if e1x7multi_all is enabled and the LANC is accepting multicast addresses.

MACGETIA is a type of request to return the Ethernet address of the LANC controller associated with the current queue. This command does not require an M_DATA block.

MACGETMCA is a request to return the entire multicast table for the controller associated with the current queue. This command does not require an M_DATA block.

MACGETSTAT is a request to return a statistic the driver has been gathering. A returned value of -1 indicates the statistic was not available. This command requires an M_DATA block. The data block is an array of structures. Each structure has the following format (see macioctl.h):

```
struct macstat {
long name ;
long value ;
}
```

A table of number defines and their descriptions follow:

MACGETSTAT	
Name	Description
MACSTAT_DEV_TIMEOUTS	total number of device timeouts
MACSTAT_XMITED	number of successful transmits
MACSTAT_XMITED_DEF	number of deferred transmits
MACSTAT_XMITED_1COLL	number of transmits with >/=1 collision
MACSTAT_COLLISIONS	total number of collisions
MACSTAT_NOXMIT_BUFF	total number dropped frames because of no STREAM buffer
MACSTAT_NOXMIT_COLL	number of frames dropped due to excess collisions
MACSTAT_RECVD	number of frames successfully received
MACSTAT_RECVD_CKSUM	number of CRC errors
MACSTAT_RECVD_ALIGN	number of frames with alignment errors
MACSTAT_NORECV_RES	number of frames dropped because of resource lack
MACSTAT_NORECV_LENGTH	number of frames dropped because of bad length
MACSTAT_RECVD_MCAST	number of multicast frames received
MACSTAT_XMITED_MCAST	number of multicast frames transmitted
MACSTAT_NORECV_MCAST	number of multicast frames rejected
MACSTAT_NORECV_TYPE	number of frames dropped because of unbound type
MACSTAT_NOXMIT_CARRIER	number of times lost carrier
MACSTAT_NOXMIT_CTS	number of times lost CTS
MACSTAT_DMA_ERRORS	number of DMA errors
MACSTAT_RECVD_BCAST	number broadcast frames received
MACSTAT_OUT_OF_WINDOW	number of late collisions
MACSTAT_XMITED_BCAST	number of broadcast frames transmitted

MACSETIA is a request to set the Ethernet address for the LANC controller associated with the current stream. After executing MACSETIA, the networking subsystem *must* be stopped and then restarted. The address is immediately changed in the LANC and the non-volatile RAM on the cpu board.

MACSETMCA is a request to add one multicast address to a multicast table on a per-stream basis. This command requires an M_DATA block of type mc_frame. A multicast address must have the least significant bit of byte[0] of the Ethernet address set. An individual stream may have a maximum of sixty-four multicast addresses in its table, subject to the following limitation. There may be no more than sixty-four addresses for all streams associated with each controller.

SIOCGENADDR is a type of request to return the Ethernet address of the LANC controller associated with the current queue. This command requires an M_DATA block of type struct ifreq.

M_FLUSH

If the command is a read queue flush, the read queue of the driver is flushed and the message is passed back up stream. If the command is a write queue flush, the write queue of the driver is flushed.

Master.d Parameters

The driver's master.d file is partitioned into two sections. Section 1 declares data structure names to be accessed by the driver software, their type, and their initial value. Section 2 contains the parameter declarations used in section 1 for setting data structure values. Most data structures are defined as arrays, where the length of the array is determined by the number of LANC controllers in the Equipped Device Table. The following table lists the section 1 parameters, their default section 2 declaration and value, and their description. Some data structures mention that certain settings of a data structure may cause networking lock-up due to a LANC bug. These settings can cause errors when the A-1 step of the LANC chip is used. The B step of the LANC chip, when released, will correct these errors.

Master.d Parameters		
Parameter	Default	Description
e1x7buf_type	STREAM (1)	Use local or STREAM buffer control flag. *This parameter is only checked on cpu boards which can snoop the bus, for example, the 167. The 187 does not snoop the bus so this parameter's setting for the 187 is not used.* The driver running on the 187 only allocates local buffers. The other setting is LOCAL (0).
e1x7rcv_nmrfd	RFDS_DEFAULT (16)	Number of receive frames that can be processed by the LANC before requiring more cpu resources. The minimum number of receive frame descriptors is four. The larger the value the more system resources may be consumed.
e1x7rcv_szbuff	RBUFSZ_DEFAULT (1514)	The size of a receive buffer in bytes. Receive buffers can be chained together by the LANC if a frame larger than a receive buffer is being processed. The minimum size for a receive buffer is 60 bytes; the maximum is 1514 bytes. Receive buffer size must be even. The larger the value the more system resources are consumed.
e1x7rcv_nmbfdes	RBUFDES_DEFAULT (17)	The number of receive buffers allocated. The minimum number allowed is four. *However, due to a bug in the LANC chip, software must ensure that receive frame descriptors always run out before all of the receive buffers are used.* This means the value for the number of receive buffers must be > (number of receive frames * 1514)/receive buffer size. The larger the value the more system resources are consumed.
e1x7tx_nmcbl	NMTXCBL_DEFAULT (16)	The number of transmit frames that can be handled by the LANC. The minimum number allowed is four. The larger the value the more system resources are consumed.

<table>
<tr><th colspan="3">Master.d Parameters (cont.)</th></tr>
<tr><th>Parameter</th><th>Default</th><th>Description</th></tr>
<tr><td>e1x7tx_szbuff</td><td>TXBUFSIZ_DEFAULT (1514)</td><td>The size of a local (not stream) transmit buffer in bytes.

The minimum buffer size is 60; the maximum is 1514. However, due to a LANC bug, the size should be kept at the maximum. If more than one buffer is used per transmit frame, networking may at some point lock-up. This would probably not occur in single-segment networks but networks with repeaters may see this error.</td></tr>
<tr><td>e1x7tx_nmbfdes</td><td>TXBUFDES_DEFAULT (20)</td><td>The number of transmit buffer descriptors.

Transmit command blocks point to transmit buffer descriptors which then point to transmit buffers. With the default setting for e1x7tx_szbuff, each transmit descriptor is associated with one complete frame.</td></tr>
<tr><td>e1x7rcv_fifo</td><td>FIFO_DEFAULT (5)</td><td>This is an index into a table of LANC receive and transmit FIFO threshold values.

The LANC has independent 128 byte receive and 64 byte transmit FIFOs. The value 8 indicates a transmit threshold of 32 bytes and a receive threshold of 64 bytes. The table is listed below:
<table>
<tr><th>Value</th><th>Tx</th><th>Rx</th></tr>
<tr><td>0</td><td>0</td><td>128</td></tr>
<tr><td>1</td><td>4</td><td>120</td></tr>
<tr><td>2</td><td>8</td><td>112</td></tr>
<tr><td>3</td><td>12</td><td>104</td></tr>
<tr><td>4</td><td>16</td><td>96</td></tr>
<tr><td>5</td><td>20</td><td>88</td></tr>
<tr><td>6</td><td>24</td><td>80</td></tr>
<tr><td>7</td><td>28</td><td>72</td></tr>
<tr><td>8</td><td>32</td><td>64</td></tr>
<tr><td>9</td><td>36</td><td>56</td></tr>
<tr><td>10</td><td>40</td><td>48</td></tr>
<tr><td>11</td><td>44</td><td>40</td></tr>
<tr><td>12</td><td>48</td><td>32</td></tr>
<tr><td>13</td><td>52</td><td>24</td></tr>
<tr><td>14</td><td>56</td><td>16</td></tr>
<tr><td>15</td><td>60</td><td>8</td></tr>
</table></td></tr>
</table>

Master.d Parameters (cont.)		
Parameter	Default	Description
e1x7bus_ton	ON_BUS_THROTL (15)	On bus throttle timer in microseconds. This is the maximum amount of time the LANC can keep the local bus before releasing it. The maximum value allowed is 30; the minimum is one.
e1x7bus_tof	OFF_BUS_THROTL (1)	Off bus throttle timer in microseconds. This is the minimum amount of time the LANC must stay off the local bus after releasing it. The maximum value allowed is 50; the minimum is one.
e1x7dbug_lvl	DEBUG_LEVEL (0)	Debug level for debugging prints to the system console. LEVEL 0 indicates debugging is off. The maximum level is three. Each higher level will print more detailed debug information.
e1x7adpt_szing	RESERVED	Reserved, must not be changed.
e1x7adpt_pkwind	RESERVED	Reserved, must not be changed.
e1x7tdr	TDR_ENABLED (1)	Time Domain Reflectometry control flag. The LANC chip can help determine where and what kind of problems are in the network cabling. If this flag is enabled and if the software thinks that there may be a cable problem, a command will be launched to try and determine where and what the problem is. If a problem is found, a warning message is printed on the system console. If this flag is disabled, TDR_DISABLED (0), then no problem checking commands will be launched.
e1x7savbadframe	SVBD_DISABLED (0)	Control flag to tell the LANC whether to pass bad frames it receives to the driver or throw them away. Even though they are thrown away, the LANC keeps statistics on bad frames. The default state is to throw away bad frames. Bad frames can be saved by setting this value to SVBD_ENABLED (1). *However, due to a LANC bug, the SVBD_ENABLED setting may cause a networking lockup.*

Master.d Parameters (cont.)		
Parameter	Default	Description
e1x7loopback	OFF_LOOPBACK (0)	Control flag for LANC loopback modes. This flag must only be changed for hardware debug purposes. An Intel 82596 User's Manual is required. Other values are INT_LOOPBACK (1), NOLPBK_LOOPBACK (2), WLPBK_LOOPBACK (3).
e1x7promiscuous	PROM_DISABLED (0)	Control flag for enabling/disabling the LANC promiscuous mode. Enabling the mode means the LANC accepts all packets transmitted on the network. Disabling the mode means the LANC accepts only broadcast, multicast, and specific packets meant for it. It is up to software layers above the driver to set up service access points to accept all packet types when the mode is PROM_ENABLED (1). This parameter can override the setting of e1x7broadcast.
e1x7broadcast	ENAB_BROADCAST (0)	Control flag to enable/disable receipt of broadcast packets. The default is to receive all broadcast packets. DISAB_BROADCAST (1) is the other option.
e1x7car_filtwid	CARFILTWID_DEFAULT (0)	The width required of the Carrier Sense signal, in bit times, before it is recognized as being active. The maximum value is 7. Changes to this value may be useful in noisy cable environments.
e1x7car_source	EXT_CARSOURCE (0)	Control flag to specify internal/external generation of Carrier Sense. In external mode, Carrier Sense is fed through the CRS pin. In internal mode INT_CARSOURCE(1), presence of the receive clock is interpreted as Carrier Sense Active.
e1x7col_filtwid	COLFILTWID_DEFAULT (0)	Specifies the width required of CDT, in bit times, for the LANC to recognize that a collision has occurred. The maximum value allowed is 7.

Master.d Parameters (cont.)		
Parameter	Default	Description
e1x7col_source	EXT_COLSOURCE (0)	Specifies external/internal collision detect source. External collision detect is fed through the CDT pin. Internal detects the presence of carrier sense during transmission or the presence of the receive clock during transmission as a collision.
e1x7multi_all	DIS_MULTIALL (1)	Control flag to enable/disable the LANC from receiving all frames that have a multicast address in the destination address field. The default is disabled. The other option is EN_MULTIALL (0).
e1x7txqu_quall	DIS_QUALL (0)	Control flag to enable/disable queuing of all transmit packets for the driver's write service routine. *This flag is for software testing only.* The default setting is disabled. The other option is EN_QUALL (1).
e1x7txqu_drop	DIS_DROPALL (0)	Control flag to disable/enable dropping of all transmit packets in the driver's put routine, i.e., no data is sent out on the cable. *This flag is for protocol stack testing only.* The default setting is disabled. The other option is EN_DROPALL (1).
e1x7tx_lngchk	DIS_LENGCHK (1)	Control flag to disable/enable receive frame length checking and transmit frame padding on the LANC chip. This cannot be used for Ethernet software/hardware networks. It can only be used for IEEE802.3 compliant software and hardware networks. Also, due to a LANC bug, setting the flag to EN_LENGCHK may cause a networking lockup.

Debug Aids

The driver calls the STREAMS logger kernel routine, `strlog`. These messages are mostly error messages. A few are only informational. The trace messages are seen with the `strace`(1M) command. Additional trace messages can be seen when the driver is compiled with `#define` E1X7_DEBUG.

The module ID for this driver is hexadecimal `e17` or `0xe17` or `3607` decimal. There are four sub-IDs and three tracing priority levels. Priority levels are 1-3; level 3 gives the most detail.

Sub-ID	Description
3	Interrupt Level Trace
2	Stream Level Trace
1	Initialization Trace
0	Generic Code Trace

Also, as discussed earlier in the **Master.d Parameters** section, `e1x7dbug_lvl` can be set to print information to the system console. Note that a level 1 setting will cause statistics to be printed when all minor devices associated with a controller are closed.

Also, when the driver has been compiled with `#define` E1X7_DEBUG, a debugging subroutine can be called from within KDB, the kernel debugger. The subroutine's name is `e1x7debugger`.

Note that when the driver is compiled with `#define` DEBUG, E1X7_DEBUG is automatically defined.

FILES

`/dev/e1x7_*`
`/usr/include/sys/dlpi.h`
`/usr/include/sys/macioctl.h`
`/usr/include/sys/e1x7.h`

SEE ALSO

`ifconfig`(1M), `mvmecpu`(1M), `slink`(1M), `strace`(1M), `edt_data`(4), `master`(4), `strcf`(4N), `arp`(7), `clone`(7), `intro`(7), `ip`(7), `streamio`(7).
McGrath, G., *A STREAMS-based Data Link Provider Interface (DLPI)*, Version 1.3, AT&T Bell Laboratories, Summit, N.J., February 1989
LT-610 Programmer Guide, Preliminary version, Retix, Santa Monica, CA, 1991

NAME

/stand/edt_data - Equipped Device Table (EDT) Data File

DESCRIPTION

The Equipped Device Table data file describes board and device specific data used for configuring a kernel. Associated with some boards is an Extended EDT (XEDT) which describes subdevices of those boards and may be of zero length. The XEDT can be read by the program via an sysm68k/sysm88k(2) call XGETEDT on the special file associated with the board. Note that not all drivers may support this option.

COMMENTS

An EDT data file may contain comments. A comment begins with the character '#' and extends to the end of the line.

GENERAL DIRECTIVE INFORMATION

An EDT data file is composed of a collection EDT data file directives.

The template for the directives is:

```
directive name [options] [cpus(s)]
{
        body
}
```

directive is the name of the directive.

name specifies the name to be associated with the directive.

options specifies strings which are directive specific.

cpu(s) specifies which CPUs this directive should be limited to. If no **cpu(s)** are specified, the directive is associated with all CPUs that the kernel may support. Valid **cpu(s)** are "mvme141", "mvme167", "mvme181", "mvme187", "mvme188", and "mvme197".

It is possible for some directives to not have a **body**, in which case the open and close braces are dropped as well. If the directive does have a **body**, it is embedded in the open and close braces and consists of whitespace separated keyword and value pairs, one per line.

When a **number** is called for it may be expressed in decimal, octal, or hexadecimal. Hexadecimal numbers must be preceded with the sting "0x". Octal number must be preceded with a leading zero.

THE VECTOR-GROUP DIRECTIVE

The **vector-group** directive specifies that group of interrupt vectors should be assigned a name, be reserved from all but explicit use, and the starting location of the group.

The template for the **vector-group** directive is:

vector-group name [ignore] [cpu(s)]

```
{
        vector-assignment  starting-location
        number-of-vectors  number
}
```

If the **ignore** string is present, the directive will always be ignored (never included into a kernel).

The **number-of-vectors** keyword specifies the number of interrupt vectors in the group.

The **vector-assignment** keyword specifies the starting location of the group. The **starting-location** may be expressed as an:

An absolute vector displacement is defined as the interrupt vector number multiplied by 4.

The string "any" will allow the vector group to automatically assigned any acceptable location that is found.

A **modulo** alignment directive specifies that the group of vectors may be automatically assigned a locatation provided that the vector number of the starting vector of the group has a remainder of zero when it is divided by the specified number.

The form of this directive is "mod(specified-number)".

THE DRIVER DIRECTIVE

The **driver** directive specifies that a device driver is required to deal with a specific piece of hardware.

The template for the **driver** directive is:

```
driver name [ignore] [probe] [cpu(s)]
{
        id                 number
        io-address         number
        io-length          number
        memory-address            number
        memory-length             number
        interrupt-level           number
        vector-assignment  starting-location
        number-of-vectors  number
        aux-info           number number number number
}
```

The **probe** string specifies that this device should be probed for when the system boots. If this is string is missing, the device and its driver are considered "required" in order to build a kernel.

The **ignore** when used with the "required" driver (one that does not have **probe** specified) it will not be included in the kernel.

Drivers with both **probe** and **ignore** are handled differently. If all of the drivers for a specific type of device are marked ignore they will be excluded from the kernel. However if only some of the devices are ignored, they may still be included into the kernel for padding purposes: making sure that the infomration emitted into the kernel for the drivers to use isn't modified by the removal of a device.

The **id** keyword specifies a unique number that identifes each device that a driver may utilize.

The **io-address** keyword specifies the starting address of the short I/O area used by the device. If the device doesn't have a short I/O area this keyword may be dropped.

The **io-length** keyword specifies the length of the devices short I/O area. If the **io-address** keyword is present, this keyword must also be present.

The **memory-address** keyword specifies the starting address of an auxiliary memory area use by this device. If the device doesn't have an auxiliary memory area this keyword may be dropped.

The **memory-length** keyword specifies the length of the devices auxiliary memory area. If the **memory-address** keyword is present, this keyword must also be present.

The **interrupt-level** keyword specifies the interrupt level that this device should interrupt with.

The **vector-assignment** and **number-of-vectors** keywords function the same as in their **vector-group** context, however an additional **vector-assignment** technique is possible. This is the indexed reference to a vector group. An indexed reference is specified by the vector group name followed by the index number embedded in open and close square brackets.

The **aux-info** keyword is used to specify driver specific values that the driver may use in whatever way it sees fit. All four numbers must be present. This keyword is optional.

THE CPU-IGNORE-INTERRUPT-LEVEL DIRECTIVE

This directive is used to ignore certain interrupt levels. This is useful when VMEbus devices are co-resident with UNIX devices and UNIX must not handle the interrupts associated with those devices.

The template for the **ignore-cpu-interrupt-level** directive is:

ignore-cpu-interrupt-level none or levels

The keyword **none**, which is also the default if this directive isn't used, specifies that all interrupt level should be allowed. Otherwise the **levels** specify which interrupt levels to ignore, each level being specified by its level number (e.g. interrupt level 5 as the digit 5).

DRIVER WRITER INFORMATION

Each of the keywords in the body of the **driver** directive causes the cunix program to automatically generate variables which may be accessed by a driver. These variables then allow the driver to know how many device of its type are configured into the kernel, their locations, and characteristics.

Each variable begins with the drivers master.d file prefix, which is denoted by the string <prefix> below. The generated arrays have the device information stored in **id** keyword order.

Variable	Data type	Use
<prefix>_cnt	unsigned int	Specifies the number of devices configured into the kernel.
<prefix>_addr	array of caddr_t	Specifies the starting short I/O addresses of each device. Derived from the **io-address** keyword.
<prefix>_iolen	array of unsigned int	Specifies the size (in bytes) of the device's short I/O space. Derived from the **io-length** keyword.
<prefix>_maddr	array of unsigned int	Specifies the starting address of auxiliary memory area of each device. Derived from the **memory-address** keyword.
<prefix>_memlen	array of unsigned int	Specifies the size (in bytes) of the auxiliary memory area of each device. Derived from the **memory-length** keyword.
<prefix>_nvec	unsigned int	Specifies the number of vectors per device. Derived from the **number-of-vectors** keyword.
<prefix>_vec	array of unsigned int	Specifies each devices interrupt vector displacement (the interrupt vector number multiplied by four). Derived from the **vector-assignment** keyword.
<prefix>_ivec	array of unsigned int	Specifies the interrupt priority level of each device. Derived from the **interrupt-level** keuword.
<prefix>_aux	array of unsigned int	Specifies the auxiliary information for each device. Each device's information is a group of 4 elements. Derived from the **aux-info** keyword.

FILES

/usr/include/sys/edt.h

SEE ALSO

cunix(1M), sysm68k(2), sysm88k(2), boot(8), edtp(8)

NAME

enet1x7 - MVME1X7 Local Area Network Interface

SEE ALSO

e1x7(7)

NAME

.environ, .pref, .variables - user-preference variable files for FACE

DESCRIPTION

The .environ, .pref, and .variables files contain variables that indicate user preferences for a variety of operations. The .environ and .variables files are located under the user's $HOME/pref directory. The .pref files are found under $HOME/FILECABINET, $HOME/WASTEBASKET, and any directory where preferences were set via the organize command. Names and descriptions for each variable are presented below. Variables are listed one per line and are of the form *variable=value*.

Variables found in .environ include:

LOGINWIN[1-4] Windows that are opened when FACE is initialized

SORTMODE Sort mode for file folder listings. Values include the following hexadecimal digits:

- 1 sorted alphabetically by name
- 2 files most recently modified first
- 800 sorted alphabetically by object type

The values above may be listed in reverse order by "ORing" the following value:

- 1000 list objects in reverse order. For example, a value of 1002 will produce a folder listing with files least recently modified displayed first. A value of 1001 would produce a "reverse" alphabetical by name listing of the folder

DISPLAYMODE Display mode for file folders. Values include the following hexadecimal digits:

- 0 file names only
- 4 file names and brief description
- 8 file names, description, plus additional information

WASTEPROMPT Prompt before emptying wastebasket (yes/no)?

WASTEDAYS Number of days before emptying wastebasket

PRINCMD[1-3] Print command defined to print files.

UMASK Holds default permissions that files will be created with.

Variables found in .pref are the following:

SORTMODE which has the same values as the SORTMODE variable described in .environ above.

DISPMODE which has the same values as the DISPLAYMODE variable described in .environ above.

Variables found in .variables include:

EDITOR	Default editor
PS1	UNIX shell prompt

FILES

$HOME/pref/.environ
$HOME/pref/.variables
$HOME/FILECABINET/.pref
$HOME/WASTEBASKET/.pref

NAME

envmon - Environment Monitor Board driver

DESCRIPTION

The *envmon* driver provides a character-device interface to the Environment Monitor Board (ENVMON). Sometimes this board is also referred to as the EMB. The ENVMON itself is responsible for the following:

— Monitoring and controlling the state of an external Uninterruptable Power Supply (UPS). Monitoring of AC-FAIL and Low-Battery conditions is provided, along with control of AC output from the UPS.

— Monitoring and controlling the state of one to four External Chassis' (typically 3 plus a UPS). Monitoring of AC-FAIL and Over-Temperature is provided, along with control of external chassis DC-power.

— Monitoring the state of up to four Internal Chassis temperature sensors.

— Monitoring and controlling the state of the Internal Power Supply (low-voltage, enable/disable).

— Host notification, via VME Interrupt and VME-accessible status registers, of any AC-FAIL, Over-Temperature or UPS Low-battery conditions.

— System reset, system power-off or UPS and external chassis power-off under host program control.

— Automatic power-off of the system and/or UPS and external chassis, upon persistent Over Temperature condition.

— Transition-module push-buttons and remotable contacts for system Reset and Abort interrupt.

The *envmon* driver provides the following:

— Read access (via *ioctl*(2)) to the ENVMON status registers A and B, for determining UPS, External Chassis, and Temperature status.

— Indirect or direct write access (via *ioctl*) to the ENVMON control register, for generating test interrupts, generating VME SYSRESET, signalling all external units (including UPS') to turn off their power, or latching off internal and external power.

— Synchronous notification (via *select*(2) and *poll*(2)) of exception conditions (first failure bit set in Status Register A).

— Interface from *uadmin*(2) system call to ENVMON control register, to control system shutdown behavior.

— Handling of ENVMON Abort switch interrupts, by trapping to the configured debugger (ROM or kdb).

SYSTEM CALL INTERFACES

The following system calls and semantics are defined for the *envmon* interface:

Open/Close

Opening the device allows I/O from/to the resultant file descriptor. Only the super-user may open for write.

Upon success, *open*(2) returns a file descriptor. On error, -1 is returned, and *errno* is set to indicate the error.

[ENODEV]
The *envmon* driver is not configured.

[ENXIO]
No *envmon* board is installed on the system.

[EPERM]
Attempt to open for write by non super-user.

Issuing a *close* has no effect on the driver or the ENVMON, other than to disassociate the driver from the passed file descriptor.

Read/Write

There is no direct read/write access provided by the driver. Such calls will return -1, with *errno* set to **[ENODEV]**.

Ioctl

#include <sys/types.h>
#include <sys/envmon.h>
int ioctl (*s, request, arg*)
int *s, request;*
int **arg;*
or
ushort **arg;*
or
struct emb_stat **arg;*

The following table shows the *ioctl* requests defined for the *envmon* driver; a description for each follows the table.

Request	Arg	Action
EMBGETSTAT	struct emb_stat *	Get current contents of status registers A & B
EMBXTUOFF	NULL	Power off all externally connected units
EMBPWRDOWN	NULL	Latch internal and external power off
EMBTESTINT	NULL	Generate ENVMON test interrupt
EMBSYSRESET	NULL	Generate VME SYSRESET
EMBARM	int *	Setup ENVMON interface to *uadmin*(2)
EMBDISARM	NULL	Reset ENVMON interface to *uadmin*
EMBWRTCMD	ushort *	Write arbitrary value to Command Register

For all commands other than **EMBGETSTAT**, the device must be open for writing.

EMBGETSTAT

This request retrieves the **current** values of the A and B status registers (interrupt cause and external device type) into the *emb_stat* structure pointed to by *arg*. The driver reads register A twice before returning its value, so any previously latched, but no longer existent failure bits are not presented.

Macros are provided, in **envmon.h**, to decode the bits of registers A and B. The macros may be used as booleans, to determine the existence and nature of any failure conditions present, and/or to identify which devices are affected.

EMBXTUOFF

This request causes the ENVMON to send the power-off signal to all attached external units. If a UPS is attached, this should cause it to disengage its inverter and cease running on batteries, thus powering off the system and any other devices attached to the UPS.

There may be no return from this operation. It should only be used on a quiescent system. It is recommended that this command be issued indirectly, via the **EMBARM** *interface.*

If an attached UPS was not running on batteries, the result of this command on the UPS is UPS-specific. It may continue to run, until AC power is actually interrupted, at which time it would likely remove power to the system immediately.

EMBPWRDOWN

This request causes the ENVMON to turn off the system internal power supply, and also send the power-off signal to all attached external units. The board latches itself in this state until physically reset by an operator.

There is no return from this operation. It should only be used on a quiescent system. It is recommended that this command be issued indirectly, via the **EMBARM** *interface*. See the *CAVEATS* section for other concerns regarding **EMBPWRDOWN**.

EMBTESTINT

This request causes the ENVMON to generate a test interrupt to the system. This should, in turn, cause any *select*ing or *poll*ing process to be awoken. The copy of status register A returned by a subsequent **EMBGETSTAT**, however *will not* have the test interrupt bit set, as this will have been cleared by the interrupt service routine.

EMBSYSRESET

This request causes the ENVMON to generate a VME SYSRESET signal on the VME bus. *There is no return from this operation. It should only be used in emergencies on a quiescent system.*

EMBARM

This request exploits a hook in the *uadmin*(2) interface in the kernel, causing it to call the *envmon* driver with the integer request pointed to by *arg*, just prior to entering its infinite loop. This loop is normally entered when a system halt is requested with an invocation of the command:

> **uadmin** *x* **0**
> (or the equivalent system call **uadmin**(*x*, **AD_HALT**)).

If *x* is A_SHUTDOWN [2], all processes are killed, and the **root** filesystem unmounted before the *envmon* request is executed. This indirect method of executing the **EMBXTUOFF**, **EMBPWRDOWN**, or **EMBSYSRESET** commands should be used to ensure that **root** is umounted prior to system power-down or reset. It is primarily designed to be used after an automatic shutdown due to an over-temperature condition, or an AC power failure (when attached to a UPS).

If *uadmin* issues the **EMBXTUOFF** command when a UPS is attached (as indicated in status register B), it waits 10 seconds and then issues an **EMBSYSRESET**. This is done in the event that an attached UPS ignores the power-off signal if AC power has returned.

By default, *uadmin* is not armed to execute any ENVMON command after an AD_HALT request, unless the emb_halt_pwrdown *master.d* parameter has been set (see the *MASTER.D PARAMETERS* section).

The **EMBARM** request has no affect on the *uadmin* behavior after an AD_BOOT or AD_IBOOT request. This is controlled by the emb_boot_reset *master.d* parameter (see the *MASTER.D PARAMETERS* section).

EMBDISARM
arg is unused and should be NULL. This request causes *uadmin* to revert to the default response to an AD_HALT request, which is controlled by the emb_halt_pwrdown *master.d* parameter (see the *MASTER.D PARAMETERS* section).

EMBWRTCMD
arg should be a pointer of type **ushort**. This request writes the value pointed to by *arg* to the Command register of the ENVMON.

The value written must include the **EMBENACMD** bit if ENVMON interrupts are to be enabled.

Upon success, *ioctl*(2) returns zero. On error, -1 is returned, and *errno* is set to indicate the error.

[EBADF]
A request other than **EMBGETSTAT** was made, but the device is not open for writing.

[ENXIO]
An **EMBXTUOFF** or **EMBPWRDOWN** was requested, but the transition module was not connected.

[EFAULT]
arg points to an invalid or protected part of the process address space.

Select

It is possible to *select* on an *envmon* file descriptor for **exception conditions**. As long as there are no bits set in Status Register A, *select* will sleep (the length is controlled by the *timeout* argument; see *select*(2)).

When the ENVMON interrupts due to a power, temperature, or test interrupt, *select* will return an FD_SET indicating that the *envmon* file descriptor has an exception condition pending, the nature of which can be read with the **EMBGETSTAT** *ioctl* request. Whenever any bit is set in Status register A, *select* will return immediately. Thus, *select* cannot be used to wait for new exception conditions (one bits), unless all previous exceptions have been cleared (and Register A has returned to 0). Also, *select* cannot be used to wait for an exception condition to be cleared.

Once an exception condition has been raised, it is necessary to poll for status changes, using **EMBGETSTAT**.

NOTE: On temperature-sensor conditions, the ENVMON can interrupt thousands of times while a sensor crosses through or hovers near its threshold temperature. The driver attempts to de-bounce this effect by disabling ENVMON interrupts for 10 seconds whenever an interrupt is received. It is possible, however, for a *select* to return an FD_SET indicating an exception condition, but for that condition to not exist when an **EMBGETSTAT** is performed, or to exist for random **EMBGETSTAT** requests. This may continue indefinitely until the temperature rises sufficiently above the threshold value to stabilize the register A contents.

Poll

It is also possible to *poll* an *envmon* file descriptor for **out-of-band data** similar to using *select* for **exception conditions**. Use **POLLRDBAND** as the requested event.

Driverinfo

The *envmon* driver includes a **driverinfo(D2DK)** routine that implements the **DXGETEDT** command. Although, strictly speaking, the *envmon* driver does not support subdevices, it does report extended EDT information for the devices connected to the transition module. The "devices" are numbered 1 through 4 corresponding to the connector numbers on the transition module. The device types are determined by the state of pins 3 and 4 of the connectors. If pin 3 is grounded then "external-disk-chassis" is returned in the xedt structure; if pin 4 is grounded then "UPS" is returned. If both pins 3 and 4 are grounded then "problem-with-device" is returned. The number of extended EDT entries is equal to the number of connected devices that either indicate "external-disk-chassis", "UPS" or "problem-with-device" based on the state of pins 3, 4 and 5.

MASTER.D PARAMETERS

The following may be set in the */etc/master.d/envmon* file.

emb_boot_reset

When set to 1, the **EMBSYSRESET** command will be sent to the ENVMON whenever *uadmin*(1M) or *uadmin*(2) is invoked to do the BOOT or IBOOT function. This parameter is set to 1 by default.

emb_halt_pwrdown

When set to 1, the **EMBPWRDOWN** command will be sent to the ENVMON whenever *uadmin*(1M) or *uadmin*(2) is invoked to do the HALT function. When set to 0, the ENVMON is not, by default, sent any command in response to the HALT request. This parameter is set to 0 by default.

The default behavior is overidden by invoking the **EMBARM** *ioctl* to specify the ENVMON command to be sent. Invoking the **EMBDISARM** *ioctl* reverts to the default behavior as controlled by emb_halt_pwrdown.

MESSAGES

The following messages are printed for the **EMBSYSRESET**, **EMBXTUOFF**, and the **EMBPWRDOWN** commands:

`ENVMON: Asserting VME SYSRESET.`

This message is printed when the **EMBSYSRESET** command is sent to the ENVMON.

`ENVMON: Shutting off external devices.`

This message is printed when the **EMBXTUOFF** command is sent to the ENVMON.

`ENVMON: Shutting off internal power.`

This message is printed when the **EMBPWRDOWN** command is sent to the ENVMON.

CAVEATS

The driver disables ENVMON interrupts for 10 seconds following any interrupt (except ABORT). During this 10 second interval *all* ENVMON interrupts are disabled, including ABORT. The purpose of this delay is to compensate for the lack of any hysteresis in the temperature sensors.

When executing a **EMBPWRDOWN** request, the ENVMON logic expects power to be removed; therefore it also asserts the VME AC-FAIL and SYSRESET lines. Thus, this command will effect a system reset, even if the ENVMON is not connected to the internal power-supply or any external units. This command should not be executed *unless* the ENVMON is properly connected to the internal power supply. Otherwise, the system will reboot automatically with the external power-off signal asserted (and latched), and any connected disk-drive chassis would be inhibited from powering up. Also, if a UPS were attached which ignored this signal while AC was present, it would remove system power immediately, when an AC failure occured.

Once a UPS power-fail or Over-Temperature condition is raised, there is a finite amount of time available before the UPS or ENVMON will remove power from the system. In the case of an AC-failure, the UPS will power down the system when its batteries are exhausted, or possibly earlier if so programmed. Similarly, the ENVMON will cut power after a fixed timeout when Over-Temperature occurs.

The AC-fail or Over-Temperature conditions may occur in any order, so user programs that detect one condition and set a grace-period timer must monitor ENVMON status during the timing interval, since the Over-Temp and battery life time constants will differ. If the condition with the smaller timeout occurs second, the UPS or ENVMON could unexpectedly and ungracefully cause a power-down.

NOTES

The environmental monitor board is supported on the m88k architecture only.

FILES

/dev/envmon_c0
/etc/master.d/envmon
/usr/include/sys/envmon.h

SEE ALSO

prtconf(1M), intro(2), poll(2), select(2), sysm88k(2)

Environment Monitor Board Set User Guide (ENVMON/D1)

NAME

ethers - Ethernet address to hostname database or domain

DESCRIPTION

The ethers file contains information regarding the known (48 bit) Ethernet addresses of hosts on the Internet. For each host on an Ethernet, a single line should be present with the following information:

Ethernet-address official-host-name

Items are separated by any number of SPACE and/or TAB characters. A '#' indicates the beginning of a comment extending to the end of line.

The standard form for Ethernet addresses is *x*:*x*:*x*:*x*:*x*:*x* where *x* is a hexadecimal number between 0 and ff, representing one byte. The address bytes are always in network order. Host names may contain any printable character other than a SPACE, TAB, NEWLINE, or comment character. It is intended that host names in the ethers file correspond to the host names in the hosts(4) file.

The ether_line routine from the Ethernet address manipulation library, ethers(3N) may be used to scan lines of the ethers file.

FILES

/etc/ethers

SEE ALSO

ethers(3N), hosts(4)

NAME

/dev/fd - file descriptor files

DESCRIPTION

These files, conventionally called /dev/fd/0, /dev/fd/1, /dev/fd/2, and so on, refer to files accessible through file descriptors. If file descriptor *n* is open, these two system calls have the same effect:

```
fd = open("/dev/fd/n",mode);
fd = dup(n);
```

On these files creat(2) is equivalent to open, and mode is ignored. As with dup, subsequent reads or writes on fd fail unless the original file descriptor allows the operations.

For convenience in referring to standard input, standard output, and standard error, an additional set of names is provided: /dev/stdin is a synonym for /dev/fd/0, /dev/stdout for /dev/fd/1, and /dev/stderr for /dev/fd/2.

SEE ALSO

open(2), dup(2)

DIAGNOSTICS

open(2) returns -1 and EBADF if the associated file descriptor is not open.

NAME

filehdr - file header for common object files

SYNOPSIS

#include <filehdr.h>

DESCRIPTION

Every common object file begins with a 20-byte header. The following C struct declaration is used:

```
struct  filehdr
{
   unsigned short  f_magic ;   /* magic number */
   unsigned short  f_nscns ;   /* number of sections */
   long            f_timdat ;  /* time & date stamp */
   long            f_symptr ;  /* file ptr to symtab */
   long            f_nsyms ;   /* number of symtab entries */
   unsigned short  f_opthdr ;  /* sizeof(opt and header) */
   unsigned short  f_flags ;   /* flags */
} ;
```

f_symptr is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in fseek(3S) to position an I/O stream to the symbol table. The UNIX system optional header is 28 bytes. The valid magic numbers are given below:

```
#define MC68MAGIC  0520   /* M68000 family of processors */
#define MC88MAGIC  0555   /* M88000 family of processors */
#define I386MAGIC  0514   /* i386 Computer */
#define WE32MAGIC  0560   /* 3B2, 3B5, and 3B15 computers */
#define N3BMAGIC   0550   /* 3B20 computer */
#define NTVMAGIC   0551   /* 3B20 computer */

#define VAXWRMAGIC 0570   /* VAX writable text segments */
#define VAXROMAGIC 0575   /* VAX read only sharable
                             text segments */
```

The value in f_timdat is obtained from the time(2) system call. Flag bits currently defined are:

```
#define F_RELFLG   0000001   /* relocation entries stripped */
#define F_EXEC     0000002   /* file is executable */
#define F_LNNO     0000004   /* line numbers stripped */
#define F_LSYMS    0000010   /* local symbols stripped */
#define F_AR16WR   0000200   /* 16-bit DEC host */
#define F_AR32WR   0000400   /* 32-bit DEC host */
#define F_AR32W    0001000   /* non-DEC host */
#define F_BM32ID   0160000   /* WE32000 family ID field */
#define F_BM32B    0020000   /* file contains WE 32100 code */
#define F_BM32MAU  0040000   /* file reqs MAU to execute */
#define F_BM32RST  0010000   /* file contains restore
                                work around [3B5/3B2 only] */
```

SEE ALSO

`time`(2), `fseek`(3S).

NAME

filesystem - file system organization

SYNOPSIS

/

/usr

DESCRIPTION

The System V file system tree is organized for administrative convenience. Distinct areas within the file system tree are provided for files that are private to one machine, files that can be shared by multiple machines of a common architecture, files that can be shared by all machines, and home directories. This organization allows sharable files to be stored on one machine but accessed by many machines using a remote file access mechanism such as RFS or NFS. Grouping together similar files makes the file system tree easier to upgrade and manage.

The file system tree consists of a root file system and a collection of mountable file systems. The mount(1M) program attaches mountable file systems to the file system tree at mount points (directory entries) in the root file system or other previously mounted file systems. Two file systems, / (the root) and /usr, must be mounted in order to have a completely functional system. The root file system is mounted automatically by the kernel at boot time; the /usr file system is mounted by the /etc/rc.boot script, which is run as part of the booting process.

The root file system contains files that are unique to each machine. It contains the following directories:

/dev	Character and block special files. These device files provide hooks into hardware devices or operating system facilities. Typically, device files are built to match the kernel and hardware configuration of the machine.
/dev/term	Terminal devices.
/dev/pts	Pseudo-terminal devices.
/dev/xt	Devices used by layers.
/dev/sxt	Shell layers device files used by shl.
/etc	Machine-specific administrative configuration files and system administration databases. /etc may be viewed as the home directory of a machine, the directory that in a sense defines the machine's identity. Executable programs are no longer kept in /etc.
/home	Root of a subtree for user directories.
/mnt	Temporary mount point for file systems. This is an empty directory on which file systems may be temporarily mounted.
/opt	Root of a subtree for add-on application packages.
/proc	Root of a subtree for the process file system.
/sbin	Essential executables used in the booting process and in manual system recovery. The full complement of utilities is available only after /usr is mounted,

/tmp	Temporary files; initialized to empty during the boot operation.
/var	Root of a subtree for varying files. Varying files are files that are unique to a machine but that can grow to an arbitrary (that is, variable) size. An example is a log file.
/var/adm	System logging and accounting files.
/var/cron	cron's log file.
/var/mail	Where users' mail is kept.
/var/opt	Top-level directory used by application packages.
/var/preserve	Backup files for vi(1) and ex(1).
/var/spool	Subdirectories for files used in printer spooling, mail delivery, cron(1), at(1), etc.
/var/tmp	Transitory files; initialized to empty during the boot operation.

Because it is desirable to keep the root file system small and not volatile, on disk-based systems larger file systems are often mounted on /home, /opt, /usr, and /var.

The file system mounted on /usr contains architecture-dependent and architecture-independent sharable files. The subtree rooted at /usr/share contains architecture-independent sharable files; the rest of the /usr tree contains architecture-dependent files. By mounting a common remote file system, a group of machines with a common architecture may share a single /usr file system. A single /usr/share file system can be shared by machines of any architecture. A machine acting as a file server may export many different /usr file systems to support several different architectures and operating system releases. Clients usually mount /usr read-only so that they don't accidentally change any shared files. The /usr file system contains the following subdirectories:

/usr/bin	Most system utilities.
/usr/sbin	Executables for system administration.
/usr/games	Game binaries and data.
/usr/include	Include header files (for C programs, etc).
/usr/lib	Program libraries, various architecture-dependent databases, and executables not invoked directly by the user (system daemons, etc).
/usr/share	Subtree for architecture-independent sharable files.
/usr/share/man	Subdirectories for on-line reference manual pages (if present).
/usr/share/lib	Architecture-independent databases.
/usr/src	Source code for utilities and libraries.
/usr/ucb	Berkeley compatibility package binaries.
/usr/ucbinclude	Berkeley compatibility package header files.

`/usr/ucblib` Berkeley compatibility package libraries.

A machine with disks may export root file systems, swap files, and `/usr` file systems to diskless or partially-disked machines that mount them into the standard file system hierarchy. The standard directory tree for sharing these file systems from a server is:

`/export` The default root of the exported file system tree.

`/export/exec/`*architecture-name*
The exported `/usr` file system supporting *architecture-name* for the current release.

`/export/exec/`*architecture-name*`.`*release-name*
The exported `/usr` file system supporting *architecture-name* for System V *release-name*.

`/export/exec/share` The exported common `/usr/share` directory tree.

`/export/exec/share.`*release-name*
The exported common `/usr/share` directory tree for System V *release-name*.

`/export/root/`*hostname* The exported root file system for *hostname*.

`/export/swap/`*hostname* The exported swap file for *hostname*.

`/export/var/`*hostname* The exported `/var` directory tree for *hostname*.

SEE ALSO

`at`(1), `fsck`(1M), `init`(1M), `intro`(4) `mknod`(1M), `mount`(1M), `sh`(1), `vi`(1).

NAME

floppy - floppy support

DESCRIPTION

Slice number 15 selects the generic floppy interface. This interface provides BCS support for PC floppy emulation.

When you opens the generic floppy, the driver determines the geometry of the diskette in the drive and sets the drive geometry to match. If the device is a 5¼" drive, the diskette is assumed to be one of the following formats:

- 320KB PC/XT low-density format with 8 sectors per track
- 360KB PC/XT low-density format with 9 sectors per track
- 1.2MB PC/AT high-density format with 15 sectors per track

If the device is a 3½" drive, the diskette is assumed to be one of the following formats:

- 720KB PC/XT high density format with 9 sectors per track
- 1.44MB PS/2 high density format with 18 sectors per track
- 2.88MB super high density format with 36 sectors per track

If there is no diskette in the drive, the open still succeeds, but any attempt to read, write, or format the diskette fails, returning ENXIO. A diskette must be put in and the drive geometry set via the FL_SET_GEOMETRY or FL_GET_INFO ioctl for the open to succeed.

IOCTL COMMANDS

The floppy disks support several ioctl(2) functions on the character or raw devices. These functions permit control beyond the normal open(2), close(2), read(2), and write(2) system calls. Any attempt to utilize ioctl(2) functions not listed below causes an EINVAL error to be returned.

All FL_* commands are defined in sys/pcflio.h.

The operations supported by disks are listed below in alphabetical order.

DKFIXBADSPOT

Lock out a bad spot on the disk based on the information in the dkbadlst structure referenced by *arg*. The dkbadlst structure is defined in sys/dk.h.

DKFORMAT

Format a disk. The dkfmt structure is defined in sys/dk.h.

DKGETCFG

Get parameters associated with the disk and store them in the dkconfig structure referenced by *arg*. The dkconfig structure is defined in sys/dk.h. The disk is not accessed by this command.

DKGETINFO

Get parameters associated with the disk and store them in the dkblk0 structure referenced by *arg*. The dkblk0 structure is defined in sys/dk.h. The disk is not accessed by this command.

DKGETSLC

Get the Volume Table of Contents (VTOC) information for a disk and return the information in a structure of type struct motorola_vtoc (defined in sys/vtoc.h) referenced by *arg*. While the number of supported slices is determined by the number of slices defined in the ddefs file, all disks are expected to support 16 slices. The disk is not accessed by this command.

DKSETINFO
: Set parameters associated with the disk based on the values in the `dkblk0` structure referenced by *arg*. The disk is not accessed by this command.

DKSETSLC
: Set the Volume Table of Contents (VTOC) information for a disk and return the information in a structure of type `struct motorola_vtoc` (defined in `sys/vtoc.h`) referenced by *arg*. The disk is not accessed by this command.

DKSETCFG
: Get parameters associated with the disk and store them in the `config` structure referenced by *arg*. The disk is not accessed by this command.

DKINQUIRY
: Return the SCSI INQUIRY data for the device; it is only valid for SCSI disks. This `ioctl` can be done on any device the calling process has open. The SCSI INQUIRY data for the device is copied into the `struct inquiry` structure pointed to by *arg*. The `struct inquiry` structure is defined in `sys/dk.h`.

DKREADCAP
: Return the SCSI READ CAPACITY data for the device; it is only valid for SCSI disks. This `ioctl` can be done on any disk or CDROM device the calling process has open. The SCSI READ CAPACITY data for the device is copied into the `struct readcap` structure pointed to by *arg*. The `struct readcap` structure is defined in `sys/dk.h`. Note: the SCSI READ CAPACITY command returns the number of the last logical block on the media. This `ioctl` adds 1 to that number so it represents the actual capacity of the device. Logical block numbers start at zero.

FL_PC_LEVEL
: Return the level of PC floppy emulation support as specified in the BCS PC floppy emulation support supplement. The level is returned to an integer pointed to by `arg`.

FL_SET_GEOMETRY
: Set the geometry of the floppy drive, possibly overriding the current actual geometry of the diskette. The information is taken from the `struct fl_geometry` structure pointed to by *arg*. This function is only valid for the generic floppy device (slice 15). For any other device (slice number), this function fails, returning `EINVAL`. The geometry is selected by passing a structure containing the number of sectors per track and the number of cylinders. The driver then determines which of the supported geometries matches this geometry and sets the drive geometry accordingly.

The geometry is selected based on the following table for 5¼" drives.

nsect	ncyl	geometry
15	80	1.2MB PC/AT format
9	40	360KB PC/XT format
8	40	320KB PC/XT format

The geometry is selected based on the following table for 3½" drives.

nsect	ncyl	geometry
36	80	2.88MB SHD format
18	80	1.44MB PS/2 format
9	80	720KB PC/XT format

If no match is found, the `ioctl` fails, returning 1 and setting `errno` to `EINVAL`. A diskette does not have to be in the drive for this `ioctl` to succeed.

If the selected geometry does not match the actual geometry of the diskette in the drive, the results of reading or writing in this state are undetermined.

A subsequent format operation (`FL_FORMAT_TRACK` or `DKFORMAT`) uses the geometry selected by this operation.

`FL_GET_INFO`

Query the status of a floppy disk drive. The information is returned to the `struct fl_info` structure pointed to by *arg*. This command first determines if there is a diskette in the drive. If there is, it then determines if the drive door has been opened since the last `open`(2) or `FL_GET_INFO` operation. If the door has been opened, it determines the current diskette's geometry and sets the drive geometry accordingly.

If the door has not been opened and closed since the last `open`(2) or `FL_GET_INFO` operation, the command returns the current *drive* geometry. Note: this may be different than the current *diskette* geometry as the result of a previous `FL_SET_GEOMETRY` operation.

The *arg* parameter points to a `fl_diskinfo` structure filled in by this command as follows:

`fl_stat` Give status information for the drive since the last time this drive was opened or the last time this `ioctl` was called. Most of these bits are set as a result of some error condition for a previous I/O operation.

`FL_EMPTY` Set if there is no diskette in the drive.

`FL_OFFLINE` Set if the drive is offline. If the drive was online during the open but has since been disconnected, then this bit is set and everything else is cleared.

`FL_WRTLCK` Set if a previous write operation failed because the media is write-protected. It is cleared before each I/O or format operation.

FL_BLANK — Set if there is an unformatted diskette in the drive or if the diskette's geometry is not listed as being supported.

FL_SOFTERR — Set if the previous I/O failed with a soft error (CRC or seek error). It is cleared before any I/O or format operation.

FL_HARDERR — Set if the previous I/O failed due to a media or drive error. It is cleared before any I/O or format operation.

FL_NOTDONE — Set whenever an I/O or format operation is sent to the drive and cleared when the operation completes successfully or with a soft error. It is not cleared if the operation completes with a hard error.

fl_type — Indicate the type of floppy drive as follows:

fl_type	drive type
1	3½" low density
2	3½" high density
3	3½" low/high density
4	5¼" low density
5	5¼" high density
6	5¼" low/high density

fl_door — Set to 1 if a previous operation failed because of a UNIT_ATTENTION condition. This means the drive door has been opened and closed. Note: this ioctl does a SCSI TEST_UNIT_READY before returning status, which gets the UNIT_ATTENTION condition if no other I/O has been attempted since the door was opened. After returning the current value to the user, this field is cleared. It can also be cleared by the open(2) system call.

fl_nsect — If a diskette is in the drive and its geometry has been determined, this is the number of sectors per track on the diskette. Otherwise, it is zero.

fl_cyl — If a diskette is in the drive and its geometry has been determined, this is the total number of cylinders on the diskette. Otherwise, it is zero.

fl_res — This is cleared.

FL_FORMAT_TRACK

Format the specified track using the current drive geometry. The *arg* parameter points to an integer containing the track number to format. If the track number is invalid, the command fails, returning ERANGE.

FL_READ
Read buffered data after an error. This function is not currently supported. It always returns zero.

V_GETSSZ
Return the physical sector size of the CDROM. The *arg* parameter specifies a structure of type io_arg (defined in sys/vtoc.h). The sectst and datasz members of the io_arg structure are ignored. The memaddr member of the structure points to the address of an integer containing the sector size after a sucessful operation.

V_PDREAD
Read the Physical Description Area of the disk. The *arg* parameter specifies a structure of type io_arg (defined in sys/vtoc.h). The sectst and datasz members of the io_arg structure are ignored. The memaddr member of the io_arg structure points to the address of a structure of type pdsector (defined in sys/vtoc.h) which contain the requested data upon successful completion.

V_PDWRITE
Write the Physical Description Area of the disk. The *arg* parameter specifies a structure of type pdinfo (defined in sys/vtoc.h). The sectst and datasz members of the io_arg structure are ignored. The memaddr member of the io_arg structure points to the address of a structure of type pdsector (defined in sys/vtoc.h) which contain the requested data upon successful completion.

V_PREAD
Read physical sectors. This interface assumes sectors are 512 bytes in length so the driver is responsible from mapping the request block to the correct portion of the correct sector on the disk regardless of the actual physical sector size. The *arg* parameter specifies a structure of type io_arg (defined in sys/vtoc.h). The sectst member of the io_arg structure contains the starting sector number and the datasz member contains the number of sectors. The memaddr member of the io_arg structure points to the address of an sufficiently large area which contain the requested data upon successful completion.

V_PWRITE
Write physical sectors. This interface assumes sectors are 512 bytes in length so the driver is responsible from mapping the requested block(s) to the correct portion of the correct sector on the disk regardless of the actual physical sector size. The *arg* parameter specifies a structure of type io_arg (defined in sys/vtoc.h). The sectst member of the io_arg structure contains the starting sector number and the datasz member contains the number of sectors. The memaddr member of the io_arg structure points to the address of an sufficiently large area which contain the requested data upon successful completion.

V_RVTOC
Read the Volume Table of Contents (VTOC) from the disk. The *arg* parameter specifies a structure of type io_arg (defined in sys/vtoc.h). The sectst and datasz members of the io_arg structure are ignored. The memaddr member of the io_arg structure points to the address of a

structure of type vtoc (defined in sys/vtoc.h) which contain the requested data upon successful completion.

V_WVTOC
: Write the Volume Table of Contents (VTOC) to the disk. The *arg* parameter specifies a structure of type vtoc (defined in sys/vtoc.h). The sectst and datasz members of the io_arg structure are ignored. The memaddr member of the io_arg structure points to the address of a structure of type vtoc (defined in sys/vtoc.h) which contain the requested data upon successful completion.

DINIT CONSIDERATIONS

The utility dinit(1M) is used to format floppy disks.

DDEFS CONSIDERATIONS

The utility ddefs defines disk characteristics. The output of the ddefs utility is a file normally saved in the /etc/dskdefs directory. This file is used as input to the dinit(1M) utility when it initializes a disk.

There are no standards for floppy ddef files.

SEE ALSO

cdrom(7), disk(7), intro(7)

NAME

fs (generic) - format of a file system volume

DESCRIPTION

File system volume format is entirely *FSType*-specific. See fs_*FSType*(4) for information.

SEE ALSO

fs_s5(4), fs_ufs(4).

NAME

fs (bfs) - format of the bfs file system volume

SYNOPSIS

```
#include <sys/types.h>
#include <sys/fs/bfs.h>
```

DESCRIPTION

The bfs superblock is stored on sector 0. Its format is:

```
struct bdsuper
{
    long bdsup_bfsmagic;      /* Magic number */
    off_t bdsup_start;        /* Filesystem data start offset */
    off_t bdsup_end;          /* Filesystem data end offset */

    /*
     * Sanity words
     */
    daddr_t bdcp_fromblock;  /* "From" block of current transfer */
    daddr_t bdcp_toblock;    /* "To" block of current transfer */
    daddr_t bdcpb_fromblock; /* Backup of "from" block */
    daddr_t bdcpb_toblock;   /* Backup of "to" block */
    long bdsup_filler[121];  /* Padding */
};

#define BFS_MAGIC   0xBADFACE/* bfs magic number */
```

The sanity words are used to promote sanity during compaction. They are used by fsck(1M) to recover from a system crash at any point during compaction. See the sections on the bfs file system in the *Machine and User Management* book for a description of compaction.

SEE ALSO

bfs-specific, inode(4).

NAME

fs (s5) - format of s5 file system volume

SYNOPSIS

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/fs/s5filsys.h>
```

DESCRIPTION

Every file system storage volume has a common format for certain vital information. Every such volume is divided into a certain number of 512-byte long sectors. Sector 0 is unused and is available to contain a bootstrap program or other information.

Sector 1 is the super-block. The format of a super-block is:

```
struct    filsys
{
  ushort  s_isize;          /* size in blocks of i-list */
  daddr_t s_fsize;          /* size in blocks of entire volume */
  short   s_nfree;          /* number of addresses in s_free */
  daddr_t s_free[NICFREE];  /* free block list */
  short   s_ninode;         /* number of i-nodes in s_inode */
  o_ino_t s_inode[NICINOD]; /* free i-node list */
  char    s_flock;          /* lock during free list */
                            /* manipulation */
  char    s_ilock;          /* lock during i-list manipulation */
  char    s_fmod;           /* super block modified flag */
  char    s_ronly;          /* mounted read-only flag */
  time_t  s_time;           /* last super block update */
  short   s_dinfo[4];       /* device information */
  daddr_t s_tfree;          /* total free blocks*/
  o_ino_t s_tinode;         /* total free i-nodes */
  char    s_fname[6];       /* file system name */
  char    s_fpack[6];       /* file system pack name */
  long    s_fill[12];       /* ADJUST to make */
                            /* sizeof filsys be 512 */
  long    s_state;          /* file system state */
  long    s_magic;          /* magic number to denote new file
                            /* system */
  long    s_type;           /* type of new file system */
};

#define  FsMAGIC     0xfd187e21  /* s_magic number */

#define  Fs1b        1           /* 512-byte block */
#define  Fs2b        2           /* 1024-byte block */
#define  Fs4b        3           /* 2048-byte block */

#define  FsOKAY      0x7c269d38  /* s_state: clean */
```

```
#define FsACTIVE    0x5e72d81a  /* s_state: active */
#define FsBAD       0xcb096f43  /* s_state: bad root */
#define FsBADBLK    0xbadbc14b  /* s_state: bad block */
                                /* corrupted it */
```

s_type indicates the file system type. Currently, three types of file systems are supported: the original 512-byte logical block, the 1024-byte logical block, and the 2048-byte logical block. s_magic is used to distinguish the s5 file system from other FSTypes. The s_type field is used to determine the blocksize of the file system; 512-bytes, 1K, or 2K. The operating system takes care of all conversions from logical block numbers to physical sector numbers.

s_state indicates the state of the file system. A cleanly unmounted, not damaged file system is indicated by the FsOKAY state. After a file system has been mounted for update, the state changes to FsACTIVE. A special case is used for the root file system. If the root file system appears damaged at boot time, it is mounted but marked FsBAD. Lastly, after a file system has been unmounted, the state reverts to FsOKAY.

s_isize is the address of the first data block after the i-list; the i-list starts just after the super-block, namely in block 2; thus the i-list is s_isize-2 blocks long. s_fsize is the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block numbers; if an ''impossible'' block number is allocated from the free list or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The s_free array contains, in s_free[1], ..., s_free[s_nfree-1], up to 49 numbers of free blocks. s_free[0] is the block number of the head of a chain of blocks constituting the free list. The first long in each free-chain block is the number (up to 50) of free-block numbers listed in the next 50 longs of this chain member. The first of these 50 blocks is the link to the next member of the chain. To allocate a block: decrement s_nfree, and the new block is s_free[s_nfree]. If the new block number is 0, there are no blocks left, so give an error. If s_nfree became 0, read in the block named by the new block number, replace s_nfree by its first word, and copy the block numbers in the next 50 longs into the s_free array. To free a block, check if s_nfree is 50; if so, copy s_nfree and the s_free array into it, write it out, and set s_nfree to 0. In any event set s_free[s_nfree] to the freed block's number and increment s_nfree.

s_tfree is the total free blocks available in the file system.

s_ninode is the number of free i-numbers in the s_inode array. To allocate an i-node: if s_ninode is greater than 0, decrement it and return s_inode[s_ninode]. If it was 0, read the i-list and place the numbers of all free i-nodes (up to 100) into the s_inode array, then try again. To free an i-node, provided s_ninode is less than 100, place its number into s_inode[s_ninode] and increment s_ninode. If s_ninode is already 100, do not bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the i-node is really free or not is maintained in the i-node itself.

s_tinode is the total free i-nodes available in the file system.

s_flock and s_ilock are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of s_fmod on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

s_ronly is a read-only flag to indicate write-protection.

s_time is the last time the super-block of the file system was changed, and is the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (UTC). During a reboot, the s_time of the super-block for the root file system is used to set the system's idea of the time.

s_fname is the name of the file system and s_fpack is the name of the pack.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. Also, i-nodes are 64 bytes long. I-node 1 is reserved for future use. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. For the format of an i-node and its flags, see inode(4).

SEE ALSO

mount(2).

fsck(1M), fsdb(1M), mkfs(1M), s5-specific inode(4)

NAME

fs (ufs) - format of ufs file system volume

SYNOPSIS

```
#include <sys/param.h>
#include <sys/types.h>
#include <sys/fs/ufs_fs.h>
```

DESCRIPTION

Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A file system is described by its super-block, and by the information in the cylinder group blocks. The super-block is critical data and is replicated before each cylinder group block to protect against catastrophic loss. This is done at mkfs time; the critical super-block data does not change, so the copies need not normally be referenced further.

```
/*
 * Super block for a file system.
 */
#define FS_MAGIC    0x011954
#define FSACTIVE    0x5e72d81a        /* fs_state: mounted */
#define FSOKAY      0x7c269d38        /* fs_state: clean */
#define FSBAD       0xcb096f43        /* fs_state: bad root */

struct  fs {
        struct  fs *fs_link;          /* linked list of file systems */
        struct  fs *fs_rlink;         /* used for incore super blocks */
        daddr_t fs_sblkno;            /* addr of super-block in filesys */
        daddr_t fs_cblkno;            /* offset of cyl-block in filesys */
        daddr_t fs_iblkno;            /* offset of inode-blocks in filesys */
        daddr_t fs_dblkno;            /* offset of first data after cg */
        long    fs_cgoffset;          /* cylinder group offset in cylinder */
        long    fs_cgmask;            /* used to calc mod fs_ntrak */
        time_t  fs_time;              /* last time written */
        long    fs_size;              /* number of blocks in fs */
        long    fs_dsize;             /* number of data blocks in fs */
        long    fs_ncg;               /* number of cylinder groups */
        long    fs_bsize;             /* size of basic blocks in fs */
        long    fs_fsize;             /* size of frag blocks in fs */
        long    fs_frag;              /* number of frags in a block in fs */
/* these are configuration parameters */
        long    fs_minfree;           /* minimum percentage of free blocks */
        long    fs_rotdelay;          /* num of ms for optimal next block */
        long    fs_rps;               /* disk revolutions per second */
/* these fields can be computed from the others */
        long    fs_bmask;             /* ''blkoff'' calc of blk offsets */
        long    fs_fmask;             /* ''fragoff'' calc of frag offsets */
        long    fs_bshift;            /* ''lblkno'' calc of logical blkno */
        long    fs_fshift;            /* ''numfrags'' calc number of frags */
/* these are configuration parameters */
        long    fs_maxcontig;         /* max number of contiguous blks */
        long    fs_maxbpg;            /* max number of blks per cyl group */
```

```
/* these fields can be computed from the others */
        long    fs_fragshift;         /* block to frag shift */
        long    fs_fsbtodb;           /* fsbtodb and dbtofsb shift constant */
        long    fs_sbsize;            /* actual size of super block */
        long    fs_csmask;            /* csum block offset */
        long    fs_csshift;           /* csum block number */
        long    fs_nindir;            /* value of NINDIR */
        long    fs_inopb;             /* value of INOPB */
        long    fs_nspf;              /* value of NSPF */
        long    fs_optim;             /* optimization preference, see below */
        long    fs_state;             /* file system state */
        long    fs_sparecon[2];       /* reserved for future constants */
/* a unique id for this filesystem (currently unused and unmaintained) */
        long    fs_id[2];             /* file system id */
/* sizes determined by number of cylinder groups and their sizes */
        daddr_t fs_csaddr;            /* blk addr of cyl grp summary area */
        long    fs_cssize;            /* size of cyl grp summary area */
        long    fs_cgsize;            /* cylinder group size */
/* these fields should be derived from the hardware */
        long    fs_ntrak;             /* tracks per cylinder */
        long    fs_nsect;             /* sectors per track */
        long    fs_spc;               /* sectors per cylinder */
/* this comes from the disk driver slicing */
        long    fs_ncyl;              /* cylinders in file system */
/* these fields can be computed from the others */
        long    fs_cpg;               /* cylinders per group */
        long    fs_ipg;               /* inodes per group */
        long    fs_fpg;               /* blocks per group * fs_frag */
/* this data must be re-computed after crashes */
        struct  csum fs_cstotal;      /* cylinder summary information */
/* these fields are cleared at mount time */
        char    fs_fmod;              /* super block modified flag */
        char    fs_clean;             /* file system is clean flag */
        char    fs_ronly;             /* mounted read-only flag */
        char    fs_flags;             /* currently unused flag */
        char    fs_fsmnt[MAXMNTLEN];  /* name mounted on */
/* these fields retain the current block allocation info */
        long    fs_cgrotor;           /* last cg searched */
        struct  csum *fs_csp[MAXCSBUFS];/* list of fs_cs info buffers */
        long    fs_cpc;               /* cyl per cycle in postbl */
        short   fs_postbl[MAXCPG][NRPOS];/* head of blocks for each rotation */
        long    fs_magic;             /* magic number */
        u_char  fs_rotbl[1];          /* list of blocks for each rotation */
};

/*
 * Cylinder group block for a file system.
 */

#define CG_MAGIC      0x090255
struct  cg {
        struct  cg *cg_link;          /* linked list of cyl groups */
        struct  cg *cg_rlink;         /* used for incore cyl groups */
        time_t  cg_time;              /* time last written */
        long    cg_cgx;               /* we are the cgx'th cylinder group */
        short   cg_ncyl;              /* number of cyl's this cg */
        short   cg_niblk;             /* number of inode blocks this cg */
```

```
          long    cg_ndblk;             /* number of data blocks this cg */
          struct  csum cg_cs;           /* cylinder summary information */
          long    cg_rotor;             /* position of last used block */
          long    cg_frotor;            /* position of last used frag */
          long    cg_irotor;            /* position of last used inode */
          long    cg_frsum[MAXFRAG];    /* counts of available frags */
          long    cg_btot[MAXCPG];      /* block totals per cylinder */
          short   cg_b[MAXCPG][NRPOS];  /* positions of free blocks */
          char    cg_iused[MAXIPG/NBBY];/* used inode map */
          long    cg_magic;             /* magic number */
          u_char  cg_free[1];           /* free block map */
};
```

SEE ALSO

ufs-specific inode(4)

NAME

fspec - format specification in text files

DESCRIPTION

It is sometimes convenient to maintain text files on the UNIX system with non-standard tabs (that is, tabs that are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by UNIX system commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets <: and :>. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

t*tabs* The t parameter specifies the tab settings for the file. The value of tabs must be one of the following:

1. a list of column numbers separated by commas, indicating tabs set at the specified columns
2. a - followed immediately by an integer *n*, indicating tabs at intervals of *n* columns
3. a - followed by the name of a "canned" tab specification

Standard tabs are specified by t-8, or equivalently, t1,9,17,25, and so on. The canned tabs that are recognized are defined by the tabs(1) command.

s*size* The s parameter specifies a maximum line size. The value of *size* must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.

m*margin* The m parameter specifies a number of spaces to be prepended to each line. The value of *margin* must be an integer.

d The d parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

e The e parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are t-8 and m0. If the s parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

If a format specification can be disguised as a comment, it is not necessary to code the d parameter.

SEE ALSO

ed(1), newform(1), tabs(1).

NAME

`fstypes` - file that registers distributed file system packages

DESCRIPTION

`fstypes` resides in directory `/etc/dfs` and lists distributed file system utilities packages installed on the system. The file system indicated in the first line of the file is the default file system. When Distributed File System (DFS) Administration commands are entered without the option `-F` *fstypes*, the system takes the file system type from the first line of the `fstypes` file.

The default package can be changed by editing the `fstypes` file with any supported text editor.

SEE ALSO

`dfmounts`(1M), `dfshares`(1M), `share`(1M), `shareall`(1M), `unshare`(1M)

NAME

group - group file

DESCRIPTION

The file /etc/group contains for each group the following information:

group name
encrypted password
numerical group ID
comma-separated list of all users allowed in the group

group is an ASCII file. The fields are separated by colons; each group is separated from the next by a new-line.

Because of the encrypted passwords, the group file can and does have general read permission and can be used, for example, to map numerical group IDs to names.

During user identification and authentication, the supplementary group access list is initialized sequentially from information in this file. If a user is in more groups than the system is configured for, {NGROUPS_MAX}, a warning will be given and subsequent group specifications will be ignored.

SEE ALSO

groups(1), newgrp(1M), passwd(1), getgroups(2), initgroups(3C), unistd(4).

NAME

`holidays` - holiday file

DESCRIPTION

The file `/etc/acct/holidays` lists holiday and prime-time information. The accounting system can use this information to give users a discount for non-prime time system use.

The file `/etc/acct/holidays` is a link to the current year's holiday file in the directory `/etc/acct/database`. This directory contains several files with the names `holiday.`*yyyy*, where *yyyy* is the number of a year.

When the system is booted, the file `/etc/rc2.d/S50holiday` is executed to link `/etc/acct/holidays` to the holiday file for the current year in `/etc/acct/database`. If `/etc/acct/database` has no holiday file for the current year, `/etc/rc2.d/S50holiday` links `/etc/acct/holidays` to the last file in `/etc/acct/database`. If there are no files in `/etc/acct/database`, `/etc/rc2.d/S50holiday` prints an error message and exits.

The holiday file contains three types of lines:

Comment Lines
: Any line marked by an asterisk in the first column is treated as a comment. Comments can appear anywhere in the file.

Year Designation Line
: This line must be the first non-comment line in the file and must appear only once. The line consists of three fields of four digits each (leading white space is ignored). The first field is the year, the second the prime time start, and the third the non-prime time start (prime time end). Prime time start and non-prime time start are specified with a 24 hour clock.

Holidays Lines
: These lines contain two fields: a date field and a description field. The date field is specified as *month/day*, where *month* and *day* are one or two digit numbers. The description field is commentary that is not used by the accounting programs.

The following is an example of a holiday file:

```
* Curr     Prime Non-Prime
* Year     Start Start
*
  1992     0800  1700
*
*
* Memorial Day is the last Monday in May
* Labor Day is the first Monday in September
* Thanksgiving Day is the fourth Thursday in November
*
* only the first column (month/day) is significant.
*
* month/day     Company
*          Holiday
*
1/1        New Years Day
5/25       Memorial Day
```

```
7/4        Indep. Day
9/7        Labor Day
11/26      Thanksgiving
11/27      day after
12/24      Christmas Eve
12/25      Christmas Day
```

NOTES

Do not put any blank lines into the holiday file. Blank lines will cause the `runacct` command to fail.

FILES

`/etc/acct/holidays`
`/etc/acct/database/*`
`/etc/rc2.d/S50holiday`

SEE ALSO

`runacct`(1M).

NAME

hosts - host name data base

SYNOPSIS

/etc/hosts

DESCRIPTION

The hosts file contains information regarding the known hosts on the DARPA Internet. For each host a single line should be present with the following information:

Internet-address official-host-name aliases

Items are separated by any number of SPACE and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts.

Network addresses are specified in the conventional '.' notation using the inet_addr routine from the Internet address manipulation library, inet(3N). Host names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

EXAMPLE

Here is a typical line from the /etc/hosts file:

```
192.9.1.20        gaia                          # John Smith
```

FILES

/etc/hosts

SEE ALSO

gethostent(3N), inet(3N)

NAME

hosts.equiv, .rhosts - trusted hosts by system and by user

DESCRIPTION

The /etc/hosts.equiv file contains a list of trusted hosts. When an rlogin(1) or rsh(1) request is received from a host listed in this file, and when the user making the request is listed in the /etc/passwd file, then the remote login is allowed with no further checking. The library routine ruserok (see rcmd(3N)) will make this verification. In this case, rlogin does not prompt for a password, and commands submitted through rsh are executed. Thus, a remote user with a local user ID is said to have equivalent access from a remote host named in this file.

The format of the hosts.equiv file consists of a one-line entry for each host, of the form:

hostname [*username*]

The *hostname* field normally contains the name of a trusted host from which a remote login can be made. However, an entry consisting of a single '+' indicates that all known hosts are to be trusted. A hostname must be the official name as listed in the hosts(4N) database. This is the first name given in the hosts database entry; hostname aliases are not recognized.

The User .rhosts File

Whenever a remote login is attempted, the remote login daemon checks for a .rhosts file in the home directory of the user attempting to log in. A user's .rhosts file has the same format as the hosts.equiv file, and is used to give or deny access only for the *specific user* attempting to log in from a given host. While an entry in the hosts.equiv file allows remote login access to *any* user from the indicated host, an entry in a user's .rhosts file only allows access from a named host to the user in whose home directory the .rhosts file appears. When this file is used, permissions in the user's home directory should allow read and search access by anyone, so it may be located and read. When a user attempts a remote login, his .rhosts file is, in effect, prepended to the hosts.equiv file for permission checking. Thus, if a host is specified in the user's .rhosts file, login access is allowed.

FILES

/etc/hosts.equiv
/etc/passwd
~/.rhosts
/etc

SEE ALSO

rlogin(1N), rsh(1N), hosts(4N), passwd(4)

NAME

ICMP - Internet Control Message Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_icmp.h>

s = socket(AF_INET, SOCK_RAW, proto);

t = t_open("/dev/icmp", O_RDWR);
```

DESCRIPTION

ICMP is the error and control message protocol used by the Internet protocol family. It is used by the kernel to handle and report errors in protocol processing. It may also be accessed by programs using the socket interface or the Transport Level Interface (TLI) for network monitoring and diagnostic functions. When used with the socket interface, a raw socket type is used. The protocol number for ICMP, used in the *proto* parameter to the socket call, can be obtained from `getprotobyname()` [see `getprotoent`(3N)]. ICMP file descriptors and sockets are connectionless, and are normally used with the `t_sndudata` / `t_rcvudata` and the `sendto()` / `recvfrom()` calls.

Outgoing packets automatically have an Internet Protocol (IP) header prepended to them. Incoming packets are provided to the user with the IP header and options intact.

ICMP is an datagram protocol layered above IP. It is used internally by the protcol code for various purposes including routing, fault isolation, and congestion control. Receipt of an ICMP redirect message will add a new entry in the routing table, or modify an existing one. ICMP messages are routinely sent by the protocol code. Received ICMP messages may be reflected back to users of higher-level protocols such as TCP or UDP as error returns from system calls. A copy of all ICMP message received by the system is provided to every holder of an open ICMP socket or TLI descriptor.

SEE ALSO

`send`(2), `getprotoent`(3N), `recvfrom`(3N), `t_rcvudata`(3N), `t_sndudata`(3N), `routing`(4), `inet`(7), `ip`(7)

Postel, Jon, *Internet Control Message Protocol — DARPA Internet Program Protocol Specification*, RFC 792, Network Information Center, SRI International, Menlo Park, Calif., September 1981

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

`EISCONN` An attempt was made to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected.

`ENOTCONN` An attempt was made to send a datagram, but no destination address is specified, and the socket has not been connected.

ENOBUFS	The system ran out of memory for an internal data structure.
EADDRNOTAVAIL	An attempt was made to create a socket with a network address for which no network interface exists.

NOTES

Replies to ICMP echo messages which are source routed are not sent back using inverted source routes, but rather go back through the normal routing mechanisms.

NAME

`if.ignore` - data base of ignored network interfaces

DESCRIPTION

The `if.ignore` file allows a system administrator to specify network interfaces that should be ignored by certain network applications. Use of this file is determined by the individual application. This file is referenced by the `ifignore` library function.

Each line of the file has the following format:

interface [*server*] [*server*] . . .

Items are separated by any number of blanks and/or tab characters. *server* names should be the device or service alias as it appears in the `/etc/services` file. The *server* names are optional and specify network services which should ignore the given *interface*. If no server names are supplied on a particular line, the corresponding *interface* should be ignored by all network services (which consult this file).

EXAMPLES

The following example illustrates how the `if.ignore` file might be used:

```
sl0     who   timed
sl1     router
ppp0
```

No `rwhod` or `timed` packets should be broadcast over the *sl0* or *ppp0* interfaces. Likewise, no `routed` packets should be broadcast over the `sl1` or `ppp0` interfaces. Furthermore, the `ifignore()` library function will return a non-zero value for all services requiring the `ppp0` interface and it will return zero for any interfaces other than `ppp0`, `sl0`, or `sl1`.

FILES

`/etc/if.ignore`

SEE ALSO

`routed`(1M), `rwhod`(1M), `timed`(1M), `ifignore`(3N), `services`(4)

NAME

inet - Internet protocol family

SYNOPSIS

```
#include &lt;sys/types.h&gt;
#include &lt;netinet/in.h&gt;
```

DESCRIPTION

The Internet protocol family implements a collection of protocols which are centered around the *Internet Protocol* (IP) and which share a common address format. The Internet family protocols can be accessed via the socket interface, where they support the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types, or the Transport Level Interface (TLI), where they support the connectionless (T_CLTS) and connection oriented (T_COTS_ORD) service types.

PROTOCOLS

The Internet protocol family comprises the Internet Protocol (IP), the Address Resolution Protocol (ARP), the Internet Control Message Protocol (ICMP), the Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP).

TCP supports the socket interface's SOCK_STREAM abstraction and TLI's T_COTS_ORD service type. UDP supports the SOCK_DGRAM socket abstraction and the TLI T_CLTS service type. See tcp(7) and udp(7). A direct interface to IP is available via both TLI and the socket interface; See ip(7). ICMP is used by the kernel to handle and report errors in protocol processing. It is also accessible to user programs; see icmp(7). ARP is used to translate 32-bit IP addresses into 48-bit Ethernet addresses; see arp(7).

The 32-bit IP address is divided into network number and host number parts. It is frequency-encoded; The most-significant bit is zero in Class A addresses, in which the high-order 8 bits represent the network number. Class B addresses have their high order two bits set to 10 and use the high-order 16 bits as the network number field. Class C addresses have a 24-bit network number part of which the high order three bits are 110. Sites with a cluster of IP networks may chose to use a single network number for the cluster; This is done by using subnet addressing. The host number portion of the address is further subdivided into subnet number and host number parts. Within a subnet, each subnet appears to be an individual network; Externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following ioctl(2) commands; They have the same form as the SIOCSIFADDR command [see if(3N)].

SIOCSIFNETMASK — Set interface network mask. The network mask defines the network part of the address; If it contains more of the address than the address type would indicate, then subnets are in use.

SIOCGIFNETMASK — Get interface network mask.

ADDRESSING

IP addresses are four byte quantities, stored in network byte order. IP addresses should be manipulated using the byte order conversion routines [see byteorder(3N)].

Addresses in the Internet protocol family use the following structure:

```
struct sockaddr_in {
      short    sin_family;
      u_short  sin_port;
      struct   in_addr sin_addr;
      char     sin_zero[8];
};
```

Library routines are provided to manipulate structures of this form; See inet(3N).

The sin_addr field of the sockaddr_in structure specifies a local or remote IP address. Each network interface has its own unique IP address. The special value INADDR_ANY may be used in this field to effect wildcard matching. Given in a bind(2) call, this value leaves the local IP address of the socket unspecified, so that the socket will receive connections or messages directed at any of the valid IP addresses of the system. This can prove useful when a process neither knows nor cares what the local IP address is or when a process wishes to receive requests using all of its network interfaces. The sockaddr_in structure given in the bind(2) call must specify an in_addr value of either IPADDR_ANY or one of the system's valid IP addresses. Requests to bind any other address will elicit the error EADDRNOTAVAI. When a connect(2) call is made for a socket that has a wildcard local address, the system sets the sin_addr field of the socket to the IP address of the network interface that the packets for that connection are routed via.

The sin_port field of the sockaddr_in structure specifies a port number used by TCP or UDP. The local port address specified in a bind(2) call is restricted to be greater than IPPORT_RESERVED (defined in <netinet/in.h>) unless the creating process is running as the super-user, providing a space of protected port numbers. In addition, the local port address must not be in use by any socket of same address family and type. Requests to bind sockets to port numbers being used by other sockets return the error EADDRINUSE. If the local port address is specified as 0, then the system picks a unique port address greater than IPPORT_RESERVED. A unique local port address is also picked when a socket which is not bound is used in a connect(2) or sendto [see send(2)] call. This allows programs which do not care which local port number is used to set up TCP connections by simply calling socket(2) and then connect(2), and to send UDP datagrams with a socket(2) call followed by a sendto(2) call.

Although this implementation restricts sockets to unique local port numbers, TCP allows multiple simultaneous connections involving the same local port number so long as the remote IP addresses or port numbers are different for each connection. Programs may explicitly override the socket restriction by setting the SO_REUSEADDR socket option with setsockopt [see getsockopt(3N)].

TLI applies somewhat different semantics to the binding of local port numbers. These semantics apply when Internet family protocols are used via the TLI.

SEE ALSO

ioctl(2), send(2), bind(3N), connect(3N), getsockopt(3N), if(3N), byteorder(3N), gethostent(3N), getnetent(3N), getprotoent(3N), getservent(3N), socket(3N), arp(7), icmp(7), ip(7), tcp(7), udp(7)

Network Information Center, *DDN Protocol Handbook* (3 vols.), Network Information Center, SRI International, Menlo Park, Calif., 1985

NOTES

The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

NAME

inetd.conf - Internet servers database

DESCRIPTION

The inetd.conf file contains the list of servers that inetd(1M) invokes when it receives an Internet request over a socket. Each server entry is composed of a single line of the form:

service-name socket-type protocol wait-status uid server-program server-arguments

Fields can be separated by either SPACE or TAB characters. A '#' (pound-sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search this file.

service-name — The name of a valid service listed in the file /etc/services. For RPC services, the value of the *service-name* field consists of the RPC service name, followed by a slash and either a version number or a range of version numbers (for example, mountd/1).

socket-type — Can be one of:

stream	for a stream socket,
dgram	for a datagram socket,
raw	for a raw socket,
seqpacket	for a sequenced packet socket

protocol — Must be a recognized protocol listed in the file /etc/protocols. For RPC services, the field consists of the string rpc followed by a slash and the name of the protocol (for example, rpc/udp for an RPC service using the UDP protocol as a transport mechanism).

wait-status — nowait for all but single-threaded datagram servers — servers which do not release the socket until a timeout occurs (such as comsat(1M) and talkd(1M)). These must have the status wait. Although tftpd(1M) establishes separate pseudo-connections, its forking behavior can lead to a race condition unless it is also given the status wait.

uid — The user ID under which the server should run. This allows servers to run with access privileges other than those for root.

server-program — Either the pathname of a server program to be invoked by inetd to perform the requested service, or the value internal if inetd itself provides the service.

server-arguments — If a server must be invoked with command-line arguments, the entire command line (including argument 0) must appear in this field (which consists of all remaining words in the entry). If the server expects inetd to pass it the address of its peer (for compatibility with 4.2BSD executable daemons), then the first argument to the command should be specified as '%A'.

FILES

/etc/inetd.conf
/etc/services
/etc/protocols

SEE ALSO

rlogin(1), rsh(1), comsat(1M), inetd(1M), talkd(1M), tftpd(1M), services(4)

NAME

inittab - script for init

DESCRIPTION

The file /etc/inittab controls process dispatching by init. The processes most typically dispatched by init are daemons.

The inittab file is composed of entries that are position dependent and have the following format:

id : *rstate* : *action* : *process*

Each entry is delimited by a newline, however, a backslash (\ ) preceding a newline indicates a continuation of the entry. Up to 512 characters per entry are permitted. Comments may be inserted in the *process* field using the convention for comments described in sh(1). There are no limits (other than maximum entry size) imposed on the number of entries in the inittab file. The entry fields are:

id This is one to four characters used to uniquely identify an entry.

rstate This defines the run level in which this entry is to be processed. Run-levels effectively correspond to a configuration of processes in the system. That is, each process spawned by init is assigned a run level or run levels in which it is allowed to exist. The run levels are represented by a number ranging from 0 through 6. As an example, if the system is in run level 1, only those entries having a 1 in the *rstate* field are processed. When init is requested to change run levels, all processes that do not have an entry in the *rstate* field for the target run level are sent the warning signal SIGTERM and allowed a 5-second grace period before being forcibly terminated by the kill signal SIGKILL. The *rstate* field can define multiple run levels for a process by selecting more than one run level in any combination from 0 through 6. If no run level is specified, then the process is assumed to be valid at all run levels 0 through 6. There are three other values, a, b and c, which can appear in the *rstate* field, even though they are not true run levels. Entries which have these characters in the *rstate* field are processed only when an init or telinit process requests them to be run (regardless of the current run level of the system). See init(1M). They differ from run levels in that init can never enter run level a, b or c. Also, a request for the execution of any of these processes does not change the current run level. Furthermore, a process started by an a, b or c command is not killed when init changes levels. They are killed only if their line in inittab is marked off in the *action* field, their line is deleted entirely from inittab, or init goes into single-user state.

action Key words in this field tell init how to treat the process specified in the *process* field. The actions recognized by init are as follows:

respawn If the process does not exist, then start the process; do not wait for its termination (continue scanning the inittab file), and when the process dies, restart the process. If the process currently exists, do nothing and continue scanning the inittab file.

`wait` — When `init` enters the run level that matches the entry's *rstate*, start the process and wait for its termination. All subsequent reads of the `inittab` file while `init` is in the same run level cause `init` to ignore this entry.

`once` — When `init` enters a run level that matches the entry's *rstate*, start the process, do not wait for its termination. When it dies, do not restart the process. If `init` enters a new run level and the process is still running from a previous run level change, the program is not restarted.

`boot` — The entry is to be processed only at `init`'s boot-time read of the `inittab` file. `init` is to start the process, not wait for its termination; and when it dies, not restart the process. In order for this instruction to be meaningful, the *rstate* should be the default or it must match `init`'s run level at boot time. This action is useful for an initialization function following a hardware reboot of the system.

`bootwait` — The entry is to be processed the first time `init` goes from single-user to multi-user state after the system is booted. (If `initdefault` is set to `2`, the process runs right after the boot.) `init` starts the process, waits for its termination and, when it dies, does not restart the process.

`powerfail` — Execute the process associated with this entry only when `init` receives a power fail signal, `SIGPWR` [see `signal`(2)].

`powerwait` — Execute the process associated with this entry only when `init` receives a power fail signal, `SIGPWR`, and wait until it terminates before continuing any processing of `inittab`.

`off` — If the process associated with this entry is currently running, send the warning signal `SIGTERM` and wait 5 seconds before forcibly terminating the process with the kill signal `SIGKILL`. If the process is nonexistent, ignore the entry.

`ondemand` — This instruction is really a synonym for the `respawn` action. It is functionally identical to `respawn` but is given a different keyword in order to divorce its association with run levels. This instruction is used only with the `a`, `b` or `c` values described in the *rstate* field.

`initdefault` — An entry with this action is scanned only when `init` is initially invoked. `init` uses this entry, if it exists, to determine which run level to enter initially. It does this by taking the highest run level specified in the *rstate* field and using that as its initial state. If the *rstate* field is empty, this is interpreted as `0123456` and `init` therefore enters run level `6`. Additionally, if `init` does not find an `initdefault` entry in `inittab`, it requests an initial run level from the user at reboot time.

`sysinit` Entries of this type are executed before `init` tries to access the console (that is, before the `Console Login:` prompt). It is expected that this entry will be only used to initialize devices on which `init` might try to ask the run level question. These entries are executed and waited for before continuing.

process This is a command to be executed. The entire `process` field is prefixed with `exec` and passed to a forked `sh` as `sh -c 'exec` *command*`'`. For this reason, any legal `sh` syntax can appear in the *process* field.

SEE ALSO

`init`(1M), `ttymon`(1M), `sh`(1), `who`(1). `exec`(2), `open`(2), `signal`(2).

NAME

inode (generic) - format of an inode

DESCRIPTION

Inode format is entirely *FSType*-specific. See inode_*FSType*(4) for information.

SEE ALSO

inode_s5(4), inode_ufs(4).

NAME

inode (bfs) - format of a bfs i-node

SYNOPSIS

```
#include <sys/types.h>
#include <sys/fs/bfs.h>
```

DESCRIPTION

```
struct bfs_dirent
{
        ushort  d_ino;                  /* inode number */
        daddr_t d_sblock;               /* Start block */
        daddr_t d_eblock;               /* End block */
        daddr_t d_eoffset;              /* EOF disk offset (absolute) */
        struct  bfsvattr d_fattr;       /* File attributes */
};
```

For the meaning of the defined type daddr_t see types(5). The bfsvattr structure appears in the header file sys/fs/bfs.h.

SEE ALSO

fs_bfs(4), types(5).

NAME

inode (s5) - format of an s5 i-node

SYNOPSIS

```
#include <sys/types.h>
#include <sys/fs/s5ino.h>
```

DESCRIPTION

An i-node for a plain file or directory in an s4 file system has the following structure defined by sys/fs/s5ino.h.

```
/* Inode structure as it appears on a disk block. */

struct dinode
{
       o_mode_t      di_mode;      /* mode and type of file */
       o_nlink_t     di_nlink;     /* number of links to file */
       o_uid_t       di_uid;       /* owner's user id */
       o_gid_t       di_gid;       /* owner's group id */
       off_t         di_size;      /* number of bytes in file */
       char          di_addr[39];  /* disk block addresses */
       unsigned char di_gen;       /* file generation number */
       time_t        di_atime;     /* time last accessed */
       time_t        di_mtime;     /* time last modified */
       time_t        di_ctime;     /* time status last changed */
};

/*
 * Of the 40 address bytes:
 *    39 are used as disk addresses
 *    13 addresses of 3 bytes each
 *    and the 40th is used as a
 *   file generation number
 */
```

For the meaning of the defined types off_t and time_t see types(5).

SEE ALSO

stat(2), l3tol(3C), fs_s5(4), types(5).

NAME

inode (ufs) - format of a ufs inode

SYNOPSIS

```
#include <sys/param.h>
#include <sys/types.h>
#include <sys/vnode.h>
#include <sys/fs/ufs_inode.h>
```

DESCRIPTION

The I node is the focus of all local file activity in UNIX. There is a unique inode allocated for each active file, each current directory, each mounted-on file, each mapping, and the root. An inode is 'named' by its dev/inumber pair. Data in icommon is read in from permanent inode on the actual volume.

```
#define EFT_MAGIC 0x90909090     /* magic cookie for EFT */
#define NDADDR   12              /* direct addresses in inode */
#define NIADDR    3              /* indirect addresses in inode */

struct inode {
     struct   inode *i_chain[2];/* must be first */
     struct   vnode i_vnode;    /* vnode associated with this inode */
     struct   vnode *i_devvp;   /* vnode for block I/O */
     u_short  i_flag;
     dev_t    i_dev;            /* device where inode resides */
     ino_t    i_number;         /* i number, 1-to-1 with device address */
     off_t    i_diroff;         /* offset in dir, where we found last entry */
     struct   fs *i_fs;         /* file sys associated with this inode */
     struct   dquot *i_dquot;   /* quota structure controlling this file */
     short    i_owner;          /* proc index of process locking inode */
     short    i_count;          /* number of inode locks for i_owner */
     short    i_rwowner;        /* proc index of process holding rwlock */
     daddr_t  i_nextr;          /* next byte read offset (read-ahead) */
     struct inode  *i_freef;    /* free list forward */
     struct inode **i_freeb;    /* free list back */
     ulong    i_vcode;          /* version code attribute */
     ulong    i_mapcnt;         /* mappings to file pages */
     int      *i_map;           /* block list for the corresponding file */
     struct   icommon {
         o_mode_t ic_smode;       /* 0: mode and type of file */
         short    ic_nlink;       /* 2: number of links to file */
         o_uid_t  ic_suid;        /* 4: owner's user id */
         o_gid_t  ic_sgid;        /* 6: owner's group id */
         quad     ic_size;      /* 8: number of bytes in file */
#ifdef _KERNEL
         struct timeval ic_atime;/* 16: time last accessed */
         struct timeval ic_mtime;/* 24: time last modified */
         struct timeval ic_ctime;/* 32: last time inode changed */
#else
         time_t  ic_atime;        /* 16: time last accessed */
         long    ic_atspare;
         time_t  ic_mtime;        /* 24: time last modified */
         long    ic_mtspare;
         time_t  ic_ctime;        /* 32: last time inode changed */
         long    ic_ctspare;
#endif
```

```
        daddr_t ic_db[NDADDR];  /* 40: disk block addresses */
        daddr_t ic_ib[NIADDR];  /* 88: indirect blocks */
        long    ic_flags;       /* 100: status, currently unused */
        long    ic_blocks;      /* 104: blocks actually held */
        long    ic_gen;         /* 108: generation number */
        mode_t  ic_mode;        /* 112: EFT version of mode*/
        uid_t   ic_uid;         /* 116: EFT version of uid */
        gid_t   ic_gid;         /* 120: EFT version of gid */
        ulong   ic_eftflag;     /* 124: indicate EFT version*/

    } i_ic;
};

struct dinode {
    union {
        struct  icommon di_icom;
        char    di_size[128];
    } di_un;
};
```

SEE ALSO

ufs-specific fs(4)

NAME

intro - introduction to special files

DEVICE NAMING CONVENTIONS

This section describes various special files that refer to specific hardware peripherals and system device drivers. STREAMS [see intro(2)] software drivers, modules, and the STREAMS-generic set of ioctl(2) system calls are also described.

The names of the entries for hardware related files 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 system device driver are discussed where applicable.

Device specific special files take the form *prefix*_c*X*d*Ysuffix*, where *prefix* uniquely defines the type of device, *X* specifies the controller number (starting from zero) of the stated device type, *Y* specifies the logical device number (starting from zero) for the device attached to the stated controller, and *suffix* specifies device-dependent information.

In addition to the device-specific special files, the system also provides *generic* special files. These special files simplify the access to commonly used devices by providing device-independent aliases (for example, ctape1) for the first cartridge tape drive.

Device *prefixes*:

Prefix	Description
m187	MVME187 CPU SCSI host adapter; M88K only
m167	MVME167 CPU SCSI host adapter; M68K only
m328	MVME328 SCSI host adapter; M68K and M88K

Hard disk, floppy, and CDROM *suffixes*:

Suffix	Description
sZ	*Z* specifies the slice on the device

Cartridge tape *suffixes*:

The variable mode suffixes will exist only if the device is capable of supporting variable mode.

Suffix	Description
<NULL>	operate in fixed block size mode, rewind on close
n	operate in fixed block size mode, no rewind on close
f	operate in fixed block size mode, rewind on close
fn	operate in fixed block size mode, no rewind on close
v	operate in variable block size mode, rewind on close
vn	operate in variable block size mode, no rewind on close

Nine-track tape suffixes:

The fixed block size mode suffixes will exist only if the device is capable of supporting fixed block mode.

Suffix	Speed	Density	Rewind on close	Variable/Fixed Mode
<NULL>	high	3	yes	variable
n	high	3	no	variable
f	high	3	yes	fixed
fn	high	3	no	fixed
v	high	3	yes	variable
vn	high	3	no	variable
l0f	low	0	yes	fixed
l0fn	low	0	no	fixed
l0v	low	0	yes	variable
l0vn	low	0	no	variable
h0f	high	0	yes	fixed
h0fn	high	0	no	fixed
h0v	high	0	yes	variable
h0vn	high	0	no	variable
l1f	low	1	yes	fixed
l1fn	low	1	no	fixed
l1v	low	1	yes	variable
l1vn	low	1	no	variable
h1f	high	1	yes	fixed
h1fn	high	1	no	fixed
h1v	high	1	yes	variable
h1vn	high	1	no	variable
l2f	low	2	yes	fixed
l2fn	low	2	no	fixed
l2v	low	2	yes	variable
l2vn	low	2	no	variable
h2f	high	2	yes	fixed
h2fn	high	2	no	fixed
h2v	high	2	yes	variable
h2vn	high	2	no	variable
l3f	low	3	yes	fixed
l3fn	low	3	no	fixed
l3v	low	3	yes	variable
l3vn	low	3	no	variable
h3f	high	3	yes	fixed
h3fn	high	3	no	fixed
h3v	high	3	yes	variable
h3vn	high	3	no	variable

Generic device names:

The *N* specifies the generic device number; *suffix* is the device dependent suffix appended to the generic device name.

Name	Description
ctape*Nsuffix*	cartridge tapes
ninetrack*Nsuffix*	9-track tapes
disk*N*	the whole disk slice of the disk
cdrom*N*	the whole disk slice of the CDROM
floppy*Nsuffix*	floppy disk drives

The disk, floppy, and CDROM device specific files are located in the /dev/{*r*}dsk directories; tape specific files are located in the /dev/rmt directory.

The generic disk, floppy, and CDROM device special files are located in the /dev/{*r*}SA directories; tape specific files are located in the /dev/rmt and /dev/rSA directories.

NETWORKING INFORMATION

The following policy applies to new or enhanced network device drivers (for example m376). A network TCP/IP node major device number is the major device number of the clone device driver. A network minor device number is the major number of the real device driver found in /etc/master.d, concatenated with the board number to which this device corresponds. Following is a pictorial representation of the minor device number as passed to the device driver.

Network TCP/IP Node Minor Device Number

The driver interprets the minor number as follows:

MINOR DEVICE #			
bit	17 16 15	14	13 12 11 10 9 8 7 6 5 4 3 2 1 0
	BOARD	RESRV	MAJOR #

where:

- The BOARD bits define the board device number. Boards are numbered from 0. The maximum board device number supported depends on the particular device.
- The RESRV bit must be set. This bit indicates to the clone driver that the entire minor device number must be passed to the cloned device driver.
- The MAJOR # bits correspond to the real major number of the network device as specified in the file /etc/master.d.

The device node name is also used as the Ethernet network interface name by cenet in the network database file /etc/strcf.

SCSI-1 HOST ADAPTER COMMON MINOR FORMAT

All SCSI-1 host adapters utilize the following common device minor format.

	MAJOR	MINOR						
bit	31 - 18	17 16 15	14	13 12 11	10	9 8 7	6 5	4 3 2 1 0
		SCSI LUN	TBD	SCSI CTRL	SCSI BUS	SCSI ADDR	TBD	DEVICE INFO

As indicated in the preceding table, the controller number is located in the high-order bits of the minor format. This allows for support of more than eight controllers in the future. Each device driver should support a minimum of eight controllers where applicable. The driver info bits in the minor format are defined as follows:

Device	Bits	Description
disks	0-3 4	slice number (0-f) reserved
all tapes	0 1	rewind/no rewind fixed/variable block mode
streaming tapes (archive, exabyte,etc.)	2-4	no operation
start/stop tapes (9-track)	2 3-4	low/high speed density selection

SCSI-2/3 HOST ADAPTER COMMON MINOR FORMAT

All SCSI-2/3 host adapters utilize the following common device minor format.

	MAJOR	MINOR					
bit	31 - 18	17	16 15	14 13 12 11 10	9 8 7	6 5	4 3 2 1 0
		TBD	SCSI CTRL	SCSI ADDR	SCSI LUN	TBD	DEVICE INFO

As indicated in the previous table, the controller number is located in the high-order bits of the minor format. This allows for support of more controllers in the future. The driver info bits in the minor format are defined as follows:

Device	Bits	Description
disks	0-3 4	slice number (0-f) reserved
all tapes	0 1	rewind/no rewind fixed/variable block mode
streaming tapes (archive, exabyte,etc.)	2-4	no operation
start/stop tapes (9-track)	2 3-4	low/high speed density selection

SEE ALSO

`cdrom`(7), `disk`(7), `floppy`(7), `tape`(7)

NAME

`intro` - introduction to file formats

DESCRIPTION

This section outlines the formats of various files. The C structure declarations for the file formats are given where applicable. Usually, the header files containing these structure declarations can be found in the directories `/usr/include` or `/usr/include/sys`. For inclusion in C language programs, however, the syntax `#include` *<filename.h> or* `#include <sys/`*filename.h> should be used.*

Because the UNIX operating system now allows the existence of multiple file system types, there are several instances of multiple manual pages with the same name. These pages all display the name of the FSType to which they pertain centered and in parentheses at the top of the page.

NAME

IP - Internet Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_RAW, proto);

t = t_open ("/dev/rawip", O_RDWR);

d = open ("/dev/ip", O_RDWR);
```

DESCRIPTION

IP is the internetwork datagram delivery protocol that is central to the Internet protocol family. Programs may use IP through higher-level protocols such as the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), or may interface directly to IP. See tcp(7) and udp(7). Direct access may be via the socket interface (using a raw socket) or the Transport Level Interface (TLI). The protocol options defined in the IP specification may be set in outgoing datagrams.

The STREAMS driver /dev/rawip is the TLI transport provider that provides raw access to IP. The device /dev/ip is the multiplexing STREAMS driver that implements the protocol processing of IP. The latter connects below to datalink providers [interface drivers, see if(3N)], and above to transport providers such as TCP and UDP.

Raw IP sockets are connectionless and are normally used with the sendto() and recvfrom() calls, [(see send(2) and recv(2)] although the connect(2) call may also be used to fix the destination for future datagrams [in which case the read(2) or recv(2) and write(2) or send(2) calls may be used]. If proto is zero, the default protocol, IPPROTO_RAW, is used. If proto is non-zero, that protocol number will be set in outgoing datagrams and will be used to filter incoming datagrams. An IP header will be generated and prepended to each outgoing datagram; received datagrams are returned with the IP header and options intact.

A single socket option, IP_OPTIONS, is supported at the IP level. This socket option may be used to set IP options to be included in each outgoing datagram. IP options to be sent are set with setsockopt() [see getsockopt(2)]. The getsockopt(2) call returns the IP options set in the last setsockopt() call. IP options on received datagrams are visible to user programs only using raw IP sockets. The format of IP options given in setsockopt() matches those defined in the IP specification with one exception: the list of addresses for the source routing options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.

At the socket level, the socket option SO_DONTROUTE may be applied. This option forces datagrams being sent to bypass the routing step in output. Normally, IP selects a network interface to send the datagram, and possibly an intermediate gateway, based on an entry in the routing table. See routing(4). When SO_DONTROUTE is set, the datagram will be sent using the interface whose network number or full IP address matches the destination address. If no interface matches, the error ENETUNRCH will be returned.

Raw IP datagrams can also be sent and received using the TLI connectionless primitives.

Datagrams flow through the IP layer in two directions: from the network *up* to user processes and from user processes *down* to the network. Using this orientation, IP is layered *above* the network interface drivers and *below* the transport protocols such as UDP and TCP. The Internet Control Message Protocol (ICMP) is logically a part of IP. See `icmp`(7).

IP provides for a checksum of the header part, but not the data part of the datagram. The checksum value is computed and set in the process of sending datagrams and checked when receiving datagrams. IP header checksumming may be disabled for debugging purposes by patching the kernel variable `ipcksum` to have the value zero.

IP options in received datagrams are processed in the IP layer according to the protocol specification. Currently recognized IP options include: security, loose source and record route (LSRR), strict source and record route (SSRR), record route, stream identifier, and internet timestamp.

The IP layer will normally forward received datagrams that are not addressed to it. Forwarding is under the control of the kernel variable *ipforwarding*: if *ipforwarding* is zero, IP datagrams will not be forwarded; if *ipforwarding* is one, IP datagrams will be forwarded. *ipforwarding* is usually set to one only in machines with more than one network interface (internetwork routers). This kernel variable can be patched to enable or disable forwarding.

The IP layer will send an ICMP message back to the source host in many cases when it receives a datagram that can not be handled. A time exceeded ICMP message will be sent if the time to live field in the IP header drops to zero in the process of forwarding a datagram. A destination unreachable message will be sent if a datagram can not be forwarded because there is no route to the final destination, or if it can not be fragmented. If the datagram is addressed to the local host but is destined for a protocol that is not supported or a port that is not in use, a destination unreachable message will also be sent. The IP layer may send an ICMP source quench message if it is receiving datagrams too quickly. ICMP messages are only sent for the first fragment of a fragmented datagram and are never returned in response to errors in other ICMP messages.

The IP layer supports fragmentation and reassembly. Datagrams are fragmented on output if the datagram is larger than the maximum transmission unit (MTU) of the network interface. Fragments of received datagrams are dropped from the reassembly queues if the complete datagram is not reconstructed within a short time period.

Errors in sending discovered at the network interface driver layer are passed by IP back up to the user process.

SEE ALSO

`read`(2), `write`(2), `connect`(3N), `getsockopt`(3N), `recv`(3N), `send`(3N), `routing`(4), `icmp`(7), `inet`(7) `tcp`(7), `udp`(7)

Postel, Jon, *Internet Protocol - DARPA Internet Program Protocol Specification*, RFC 791, Network Information Center, SRI International, Menlo Park, Calif., September 1981

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

EACCESS	A IP broadcast destination address was specified and the caller was not the privileged user.
EISCONN	An attempt was made to establish a connection on a socket which already had one, or to send a datagram with the destination address specified and the socket was already connected.
EMSGSIZE	An attempt was made to send a datagram that was too large for an interface, but was not allowed to be fragmented (such as broadcasts).
ENETUNREACH	An attempt was made to establish a connection or send a datagram, where there was no matching entry in the routing table, or if an ICMP destination unreachable message was received.
ENOTCONN	A datagram was sent, but no destination address was specified, and the socket had not been connected.
ENOBUFS	The system ran out of memory for fragmentation buffers or other internal data structure.
EADDRNOTAVAIL	An attempt was made to create a socket with a local address that did not match any network interface, or an IP broadcast destination address was specified and the network interface does not support broadcast.

The following errors may occur when setting or getting IP options:

EINVAL	An unknown socket option name was given.
EINVAL	The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

NOTES

Raw sockets should receive ICMP error packets relating to the protocol; currently such packets are simply discarded.

Users of higher-level protocols such as TCP and UDP should be able to see received IP options.

NAME

issue - issue identification file

DESCRIPTION

The file /etc/issue contains the issue or project identification to be printed as a login prompt. issue is an ASCII file that is read by program getty and then written to any terminal spawned or respawned from the *lines* file.

FILES

/etc/issue

SEE ALSO

login(1).

NAME

iuart - hardware specific console driver for the MVME141 and MVME181/188

DESCRIPTION

This STREAMS-based driver provides console I/O when the system is running on an MVME141, MVME181 or MVME188 CPU board. This driver is accessable only through the standard console device special files /dev/console (/dev/contty00), /dev/contty (/dev/contty01), and /dev/conctl.

The device special files eventually access the STREAMS-based console driver which, when used in conjunction with the STREAMS line discipline module ldterm, supports the termios(2) and termio(7) processing.

FILES

/dev/console
/dev/contty
/dev/contty??
/dev/conctl

SEE ALSO

dcon(1A), mvmecpu(1M), termios(2), cons1x7(7), console(7), ldterm(7), termio(7).

NAME

kbd - generalized string translation module

DESCRIPTION

The STREAMS module kbd is a programmable string translation module. It can perform two types of operations on an input stream: the first type is simple byte-swapping via a lookup table, the second is string translation. It is useful for *codeset conversion* and *compose-key* or *dead-key* character production on terminals and production of overstriking sequences on printers. It may also be used for minor types of key-rebinding, expansion of abbreviations, and keyboard re-arrangement (an example of the latter would be swapping the positions of the **Y** and **Z** keys, required for German keyboards, or providing Dvorak keyboard emulation for QWERTY keyboards). The manual entry kbdcomp(1M) discusses table construction, the input language, and contains sample uses. This document is intended mainly to aid administrators in configuring the module on a particular system; the user interface to the module is solely through the commands kbdload and kbdset.

The kbd module works by modifying an input stream according to instructions embodied in tables. It has no built in "default" tables. Some tables may be loaded when the system is first brought up by pushing the module and loading standard or often-used tables [see kbdload(1M)] which are retained in main-memory across invocations and made available to all users. These are called *public tables*. Users may also load *private tables* at any time—these do not remain resident.

With the kbdset command, users may query the module for a list of available and attached tables, attach various tables, and set the optional per-user *hot-key*, *hot-key mode*, and *verbose string* for their particular invocation.

When a user attaches more than one table, the user's *hot-key* may be used to cycle to the next table in the list. If only one table is specified, the *hot-key* may be used to toggle translation on and off. When multiple tables are in use, the *hot-key* may be used to cycle through the list of tables [see kbdset(1) for a description of the available modes].

In its initial state, kbd scans input for occurrences of bytes beginning a translation sequence. Upon receiving such a byte, it attempts to match subsequent bytes of the input to programmed sequences. Input is buffered beginning with the byte which caused the state change and is *released* if a match is not found. When a match fails, the first byte of the invalid sequence is sent upstream, the buffered input is "shifted," and the scan begins again with the resulting input sequence. If the current table contains an error entry, its value (one or more bytes) is substituted for the offending input byte. When a sequence is found to be valid, the entire sequence is replaced with the *result string* specified for it.

The kbd may be used in either the *read* or *write* directions, or both simultaneously. Maps and hot-keys may be specified independently for input and output.

The kbd also supports the use of *external kernel-resident functions* as if they were tables; once declared and attached (via kbdload and kbdset respectively) they may be used as simple tables or members of composites. To accomplish this, kbd understands the registration functions of the alp module and can access any function registered with that module. Further information on external functions and their definition is contained in alp(7). External functions are especially useful in supporting multi-byte codeset conversions that would be difficult or impossible with normal kbd tables.

LIMITATIONS

It is not an error to attach multiple tables without defining a hot-key (but the tables will not all be accessible). It is recommended that the user's hot-key be set before loading and attaching tables to avoid unpleasant side effects when an unfamiliar arrangement is first loaded.

Each user has a limitation on the amount of memory that may be used for private and attached tables. This "quota" is controlled by the kbd_umem variable described below. When a user that is not the super-user attempts to load a table or create a *composite table*, the quota is checked, and the load will fail if it would cause the quota to be exceeded. When a composite table is attached, the space for *attachment* (which requires more space than the composite table itself) is charged against this quota (attachment of simple tables is *not* charged against the quota). The quota is enforced only when loading new tables. Detaching temporarily from un-needed composite tables may reduce the current allocation enough to load a table that would otherwise fail due to quota enforcement. To minimize chances of failure while loading tables, it is advisable to load all required tables and make all required composite tables before attaching any of them.

CONFIGURATION PARAMETERS

The master (or space.c) file contains some configurable parameters.

NKBDU is the maximum number of tables that may be attached by a single user. The number should be enough to cover uncommon cases, but must be at least 2. Default: 6.

ZUMEM, from which the variable kbd_umem is assigned, is the maximum number of bytes that a user (other than the super user) may have allocated to private tables (i.e., the quota). Default: 4096.

KBDTIME is the default timer value for *timeout mode*. It is the number of *clock ticks* allowed before timing out. The value of one "clock tick" depends on the hardware, but is usually 1/100 or 1/60 of a second. A timeout value of 20 is 1/5 second at 100Hz; with a 60Hz clock, a value of 12 produces a 1/5 second timeout.) Values from 5 to 400 inclusive are allowed by the module; if the value set for KBDTIME is outside this range, the module forces it to the nearest limit. (This value is only a default; users may change their particular Stream to use a different value depending on their own preferences, terminal baud-rate, and typing speed.)

CAVEATS

NULL characters may not be used in result or input *strings*, because they are used as string delimiters.

One should be able to obtain information on timeout values of currently attached tables, and be able to reset values more easily.

EXAMPLE

The shell script below installs the kbd STREAMS module into a stream and attaches two example mapping tables to the input side of the stream. The example mapping tables are assumed to be included in the BOS binary distribution. The Dvorak table maps the keyboard as if it were arranged in the Dvorak style, and the Deutsche table just transposes keys Y and Z.

The small C program generates an escape sequence needed by the example. Build and run it first.

The script assumes your session was started by an `rlogin` to the machine. You may have to modify it if your stream is not the same as the one expected below. Use `strconf` to check your stream.

After running the script, the Dvorak map will be enabled. Entering the hot key, control-underbar (^_), will change to the Deutche map. Entering the hot key again will change to a clear keyboard with no mapping.

```
# begin example script

current_tty_settings="'stty -g'"
current_tty_streams_modules="'strconf'"

streams_modules_i_know_of="ttcompat
ldterm
ptem
pts"

if [ "$current_tty_streams_modules" = "$streams_modules_i_know_of" ]
then
      #pop off ttcompat and ldterm
        strchg -p
        strchg -p

      #push kbd and put ldterm and ttcompat back
        strchg -h kbd,ldterm,ttcompat

      #restore the stty settings
        stty $current_tty_settings
else
        echo "Sorry. I only know about default pty stream modules."
        exit 255
fi

# load the two maps and attach them to the input side of the stream
kbdload /usr/lib/kbd/Dvorak
kbdload /usr/lib/kbd/Deutsche
kbdset -a Dvorak -a Deutsche

# set the hot key to control-underbar and mode 1 (see kbdset(1))
# include a string to use for verbose map changes with the hot key
kbdset -k '^_' -m 1 -v 'cat Ver.Set.Str'

#end example script

/* This program creates the file Ver.Set.Str containing the escape
 * sequence string needed in the kbd module usage example.
 * Build and run it once before running the example script.
 */ #include <stdio.h> #include <sys/types.h>
```

```
/* save cursor, goto-status-line, clear-to-end-of-line,
 * (%n), restore cursor
 */

char str[] = {  0x1b, '7',
                0x1b, '[', '?', 'j',
                0x1b, '[', 'K',
                '(', '%', 'n', ')',
                0x1b, '8'
                };

main() {
   FILE *fid;

   fid = fopen("Ver.Set.Str", "w");
   fwrite(str, sizeof(char), 15, fid);
   fclose(fid); }
```

FILES

/usr/lib/kbd	directory containing system standard table files.
/usr/lib/kbd/*.map	source for some system table files.

SEE ALSO

kbdcomp(1M), kbdload(1M), kbdset(1), alp(7).

NAME

ldterm - standard STREAMS terminal line discipline module

DESCRIPTION

ldterm is a STREAMS module that provides most of the termio(7) terminal interface. This module does not perform the low-level device control functions specified by flags in the c_cflag word of the termio/termios structure or by the IGNBRK, IGNPAR, PARMRK, or INPCK flags in the c_iflag word of the termio/termios structure; those functions must be performed by the driver or by modules pushed below the ldterm module. All other termio/termios functions are performed by ldterm; some of them, however, require the cooperation of the driver or modules pushed below ldterm and may not be performed in some cases. These include the IXOFF flag in the c_iflag word and the delays specified in the c_oflag word.

ldterm also handles EUC and multi-byte characters.

The remainder of this section describes the processing of various STREAMS messages on the read- and write-side.

Read-side Behavior

Various types of STREAMS messages are processed as follows:

M_BREAK When this message is received, either an interrupt signal is generated or the message is treated as if it were an M_DATA message containing a single ASCII NUL character, depending on the state of the BRKINT flag.

M_DATA This message is normally processed using the standard termio input processing. If the ICANON flag is set, a single input record (''line'') is accumulated in an internal buffer and sent upstream when a line-terminating character is received. If the ICANON flag is not set, other input processing is performed and the processed data are passed upstream.

If output is to be stopped or started as a result of the arrival of characters (usually CNTRL-Q and CNTRL-S), M_STOP and M_START messages are sent downstream. If the IXOFF flag is set and input is to be stopped or started as a result of flow-control considerations, M_STOPI and M_STARTI messages are sent downstream.

M_DATA messages are sent downstream, as necessary, to perform echoing.

If a signal is to be generated, an M_FLUSH message with a flag byte of FLUSHR is placed on the read queue. If the signal is also to flush output, an M_FLUSH message with a flag byte of FLUSHW is sent downstream.

M_CTL If the size of the data buffer associated with the message is the size of struct iocblk, ldterm will perform functional negotiation to determine where the termio(7) processing is to be done. If the command field of the iocblk structure (ioc_cmd) is set to MC_NO_CANON, the input canonical processing normally performed on M_DATA messages is disabled and those messages are passed upstream unmodified; this is for the use of modules or drivers that perform their own input processing, such as a pseudo-terminal in TIOCREMOTE mode connected to a program that performs this processing. If the command is

MC_DO_CANON, all input processing is enabled. If the command is MC_PART_CANON, then an M_DATA message containing a termios structure is expected to be attached to the original M_CTL message. The ldterm module will examine the iflag, oflag, and lflag fields of the termios structure and from then on will process only those flags which have not been turned ON. If none of the above commands are found, the message is ignored; in any case, the message is passed upstream.

M_FLUSH The read queue of the module is flushed of all its data messages and all data in the record being accumulated are also flushed. The message is passed upstream.

M_IOCACK The data contained within the message, which is to be returned to the process, are augmented if necessary, and the message is passed upstream.

All other messages are passed upstream unchanged.

Write-side Behavior

Various types of STREAMS messages are processed as follows:

M_FLUSH The write queue of the module is flushed of all its data messages and the message is passed downstream.

M_IOCTL The function of this ioctl is performed and the message is passed downstream in most cases. The TCFLSH and TCXONC ioctls can be performed entirely in the ldterm module, so the reply is sent upstream and the message is not passed downstream.

M_DATA If the OPOST flag is set, or both the XCASE and ICANON flags are set, output processing is performed and the processed message is passed downstream along with any M_DELAY messages generated. Otherwise, the message is passed downstream without change.

All other messages are passed downstream unchanged.

IOCTLS

The following ioctls are processed by the ldterm module. All others are passed downstream. EUC_WSET and EUC_WGET are I_STR ioctl calls whereas other ioctls listed here are TRANSPARENT ioctls.

TCGETS/TCGETA
: The message is passed downstream; if an acknowledgment is seen, the data provided by the driver and modules downstream are augmented and the acknowledgement is passed upstream.

TCSETS/TCSETSW/TCSETSF/TCSETA/TCSETAW/TCSETAF
: The parameters that control the behavior of the ldterm module are changed. If a mode change requires options at the stream head to be changed, an M_SETOPTS message is sent upstream. If the ICANON flag is turned on or off, the read mode at the stream head is changed to message-nondiscard or byte-stream mode, respectively. If the TOSTOP flag is turned on or off, the tostop mode at the stream head is turned on or off, respectively.

TCFLSH — If the argument is 0, an M_FLUSH message with a flag byte of FLUSHR is sent downstream and placed on the read queue. If the argument is 1, the write queue is flushed of all its data messages and an M_FLUSH message with a flag byte of FLUSHW is sent upstream and downstream. If the argument is 2, the write queue is flushed of all its data messages and an M_FLUSH message with a flag byte of FLUSHRW is sent downstream and placed on the read queue.

TCXONC — If the argument is 0 and output is not already stopped, an M_STOP message is sent downstream. If the argument is 1 and output is stopped, an M_START message is sent downstream. If the argument is 2 and input is not already stopped, an M_STOPI message is sent downstream. If the argument is 3 and input is stopped, an M_STARTI message is sent downstream.

TCSBRK — The message is passed downstream, so the driver has a chance to drain the data and then send and an M_IOCACK message upstream.

EUC_WSET — This call takes a pointer to an eucioc structure, and uses it to set the EUC line discipline's local definition for the code set widths to be used for subsequent operations. Within the stream, the line discipline may optionally notify other modules of this setting via M_CTL messages.

EUC_WGET — This call takes a pointer to an eucioc structure, and returns in it the EUC code set widths currently in use by the EUC line discipline.

SEE ALSO

termios(2), console(7), ports(7), termio(7).

NAME

limits - header file for implementation-specific constants

SYNOPSIS

```
#include <limits.h>
```

DESCRIPTION

The header file limits.h is a list of minimal magnitude limitations imposed by a specific implementation of the operating system.

```
CHAR_BIT      8                        /* max # of bits in a "char" */
CHAR_MAX      127                      /* max value of a "char" */
CHAR_MIN      128                      /* min value of a "char" */
CHILD_MAX     25                       /* max # of processes per user id */
CLK_TCK       100                      /* clock ticks per second */
DBL_DIG       15                       /* digits of precision of a "double" */
DBL_MAX       1.7976931348623157E+308  /* max decimal value of a "double"*/
DBL_MIN       2.2250738585072014E-308  /* min decimal value of a "double"*/
FCHR_MAX      1048576                  /* max size of a file in bytes */
FLT_DIG       6                        /* digits of precision of a "float" */
FLT_MAX       3.40282347e+38F          /* max decimal value of a "float" */
FLT_MIN       1.17549435E-38F          /* min decimal value of a "float" */
INT_MAX       2147483647               /* max value of an "int" */
INT_MIN       (-2147483647-1)          /* min value of an "int" */
LINK_MAX      1024                     /* max # of links to a single file */
LOGNAME_MAX   8                        /* max # of characters in a login name */
LONG_BIT      32                       /* # of bits in a "long" */
LONG_MAX      2147483647               /* max value of a "long int" */
LONG_MIN      (-2147483647-1)          /* min value of a "long int" */
MAX_CANON     255                      /* max bytes in a line for canonical
                                       processing */
MAX_INPUT     512                      * max size of a char input buffer */
MB_LEN_MAX    5                        /* max # of bytes in a multibyte
                                       character */
NAME_MAX      14                       /* max # of characters in a file name */
NGROUPS_MAX   16                       /* max # of groups for a user */
NL_ARGMAX     9                        /* max value of "digit" in calls to the
                                       NLS printf() and scanf() */
NL_LANGMAX    14                       /* max # of bytes in a LANG name */
NL_MSGMAX     32767                    /* max message number */
NL_NMAX       1                        /* max # of bytes in N-to-1 mapping
                                       characters */
NL_SETMAX     255                      /* max set number */
NL_TEXTMAX    255                      /* max # of bytes in a message string */
NZERO         20                       /* default process priority */
OPEN_MAX      25                       /* max # of files a process can have
                                       open */
PASS_MAX      8                        /* max # of characters in a password */
PATH_MAX      1024                     /* max # of characters in a path name */
PID_MAX       30000                    /* max value for a process ID */
PIPE_BUF      5120                     /* max # bytes atomic in write to a pipe */
PIPE_MAX      5120                     /* max # bytes written to a pipe
```

```
                                        in a write */
SCHAR_MAX       127                     /* max value of a "signed char" */
SCHAR_MIN       (-128)                  /* min value of a "signed char" */
SHRT_MAX        32767                   /* max value of a "short int" */
SHRT_MIN        (-32768)                /* min value of a "short int" */
STD_BLK         1024                    /* # bytes in a physical I/O block */
SYS_NMLN        256                     /* 4.0 size of utsname elements */
                                        /* also defined in sys/utsname.h */
SYSPID_MAX      1                       /* max pid of system processes */
UCHAR_MAX       255                     /* max value of an "unsigned char" */
UID_MAX         60002                   /* max value for a user or group ID */
UINT_MAX        4294967295              /* max value of an "unsigned int" */
ULONG_MAX       4294967295              /* max value of an "unsigned long int" */
USHRT_MAX       65535                   /* max value of an "unsigned short int" */
USI_MAX         4294967295              /* max decimal value of an "unsigned" */
WORD_BIT        32                      /* # of bits in a "word" or "int" */
```

The following POSIX definitions are the most restrictive values to be used by a POSIX conformant application. Conforming implementations shall provide values at least this large.

```
_POSIX_ARG_MAX          4096    /* max length of arguments to exec */
_POSIX_CHILD_MAX        6       /* max # of processes per user ID */
_POSIX_LINK_MAX         8       /* max # of links to a single file */
_POSIX_MAX_CANON        255     /* max # of bytes in a line of input */
_POSIX_MAX_INPUT        255     /* max # of bytes in terminal
                                input queue */
_POSIX_NAME_MAX         14      /* # of bytes in a filename */
_POSIX_NGROUPS_MAX       0      /* max # of groups in a process */
_POSIX_OPEN_MAX         16      /* max # of files a process can have open */
_POSIX_PATH_MAX         255     /* max # of characters in a pathname */
_POSIX_PIPE_BUF         512     /* max # of bytes atomic in write
                                to a pipe */
```

NAME

`lo` - software loopback network interface

SYNOPSIS

```
d = open ("/dev/loop", O_RDWR);
```

DESCRIPTION

The `loopback` device is a software datalink provider (interface driver) that returns all packets it receives to their source without involving any hardware devices. It is a STREAMS device conforming to the datalink provider interface (DLPI). See `if`(7) for a general description of network interfaces.

The `loopback` interface is used to access Internet services on the local machine. Because it is available on all machines, including those with no hardware network interfaces, programs can use it for guaranteed access to local servers. A typical application is the `comsat`(1M) server which accepts notification of mail delivery from a local client. The loopback interface is also used for performance analysis and testing.

By convention, the name of the loopback interface is `lo0`, and it is configured with Internet address 127.0.0.1. This address may be changed with the `SIOCSIFADDR` `ioctl()`.

SEE ALSO

`comsat`(1M), `if`(7), `inet`(7)

NAME

log - interface to STREAMS error logging and event tracing

DESCRIPTION

log is a STREAMS software device driver that provides an interface for console logging and for the STREAMS error logging and event tracing processes (strerr(1M), strace(1M)). log presents two separate interfaces: a function call interface in the kernel through which STREAMS drivers and modules submit log messages; and a subset of ioctl(2) system calls and STREAMS messages for interaction with a user level console logger, an error logger, a trace logger, or processes that need to submit their own log messages.

Kernel Interface

log messages are generated within the kernel by calls to the function strlog:

```
strlog(mid, sid, level, flags, fmt, arg1, ...)
short mid, sid;
char level;
ushort flags;
char *fmt;
unsigned arg1;
```

Required definitions are contained in sys/strlog.h, sys/log.h, and sys/syslog.h. *mid* is the STREAMS module id number for the module or driver submitting the log message. *sid* is an internal sub-id number usually used to identify a particular minor device of a driver. *level* is a tracing level that allows for selective screening out of low priority messages from the tracer. *flags* are any combination of SL_ERROR (the message is for the error logger), SL_TRACE (the message is for the tracer), SL_CONSOLE (the message is for the console logger), SL_FATAL (advisory notification of a fatal error), and SL_NOTIFY (request that a copy of the message be mailed to the system administrator). *fmt* is a printf(3S) style format string, except that %s, %e, %E, %g, and %G conversion specifications are not handled. Up to NLOGARGS (currently 3) numeric or character arguments can be provided.

User Interface

log is opened via the clone interface, /dev/log. Each open of /dev/log obtains a separate stream to log. In order to receive log messages, a process must first notify log whether it is an error logger, trace logger, or console logger via a STREAMS I_STR ioctl call (see below). For the console logger, the I_STR ioctl has an ic_cmd field of I_CONSLOG, with no accompanying data. For the error logger, the I_STR ioctl has an ic_cmd field of I_ERRLOG, with no accompanying data. For the trace logger, the ioctl has an ic_cmd field of I_TRCLOG, and must be accompanied by a data buffer containing an array of one or more struct trace_ids elements. Each trace_ids structure specifies an *mid*, *sid*, and *level* from which message will be accepted. strlog will accept messages whose *mid* and *sid* exactly match those in the trace_ids structure, and whose level is less than or equal to the level given in the trace_ids structure. A value of -1 in any of the fields of the trace_ids structure indicates that any value is accepted for that field.

Once the logger process has identified itself via the ioctl call, log will begin sending up messages subject to the restrictions noted above. These messages are obtained via the getmsg(2) system call. The control part of this message contains a log_ctl structure, which specifies the *mid*, *sid*, *level*, *flags*, time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, a sequence number, and a priority. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of log messages can be determined.

The priority is comprised of a priority code and a facility code, found in <sys/syslog.h>. If SL_CONSOLE is set in *flags*, the priority code is set as follows. If SL_WARN is set, the priority code is set to LOG_WARNING. If SL_FATAL is set, the priority code is set to LOG_CRIT. If SL_ERROR is set, the priority code is set to LOG_ERR. If SL_NOTE is set, the priority code is set to LOG_NOTICE. If SL_TRACE is set, the priority code is set to LOG_DEBUG. If only SL_CONSOLE is set, the priority code is set to LOG_INFO. Messages originating from the kernel have the facility code set to LOG_KERN. Most messages originating from user processes will have the facility code set to LOG_USER.

Different sequence numbers are maintained for the error and trace logging streams, and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by NLOGARGS words for the arguments to the format string, aligned on the first word boundary following the format string.

A process may also send a message of the same structure to log, even if it is not an error or trace logger. The only fields of the log_ctl structure in the control part of the message that are accepted are the *level*, *flags*, and *pri* fields; all other fields are filled in by log before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to NLOGARGS) must be packed one word each, on the next word boundary following the end of the format string.

ENXIO is returned for I_TRCLOG ioctls without any trace_ids structures, or for any unrecognized I_STR ioctl calls. Incorrectly formatted log messages sent to the driver by a user process are silently ignored (no error results).

Processes that wish to write a message to the console logger may direct their output to /dev/conslog, using either write(2) or putmsg(2).

EXAMPLES

Example of I_ERRLOG notification.

```
struct strioctl ioc;

ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0;    /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;

ioctl(log, I_STR, &ioc);
```

Example of I_TRCLOG notification.

```
struct trace_ids tid[2];

tid[0].ti_mid = 2;
tid[0].ti_sid = 0;
tid[0].ti_level = 1;

tid[1].ti_mid = 1002;
tid[1].ti_sid = -1;   /* any sub-id will be allowed */
tid[1].ti_level = -1; /* any level will be allowed */

ioc.ic_cmd = I_TRCLOG;
ioc.ic_timout = 0;
ioc.ic_len = 2 * sizeof(struct trace_ids);
ioc.ic_dp = (char *)tid;

ioctl(log, I_STR, &ioc);
```

Example of submitting a `log` message (no arguments).

```
struct strbuf ctl, dat;
struct log_ctl lc;
char *message = "Don't forget to pick up some milk
                 on the way home";

ctl.len = ctl.maxlen = sizeof(lc);
ctl.buf = (char *)&lc;

dat.len = dat.maxlen = strlen(message);
dat.buf = message;

lc.level = 0;
lc.flags = SL_ERROR|SL_NOTIFY;

putmsg(log, &ctl, &dat, 0);
```

FILES

/dev/log
/dev/conslog
<sys/log.h>
<sys/strlog.h>
<sys/syslog.h>

SEE ALSO

strace(1M), strerr(1M), intro(2), getmsg(2), putmsg(2), write(2), clone(7).

NAME

loginlog - log of failed login attempts

DESCRIPTION

After five unsuccessful login attempts, all the attempts are logged in the file /var/adm/loginlog. This file contains one record for each failed attempt. Each record contains the login name, tty specification, and time.

This is an ASCII file. Each field within each entry is separated from the next by a colon. Each entry is separated from the next by a new-line.

By default, loginlog does not exist, so no logging is done. To enable logging, the log file must be created with read and write permission for owner only. Owner must be root and group must be sys.

FILES

/var/adm/loginlog

SEE ALSO

login(1), passwd(1).

NAME

lp1x7 - line printer device driver

DESCRIPTION

lp1x7 provides an interface to any of the standard Printronix- or Centronics-type parallel line printers using the parallel port on the MVME187 and MVME167 CPU boards.

Printers under System V Release 4 must appear as write-only terminals and are configured using a terminal type in terminfo(4). The lpadmin(1M) command is used to configure the printer.

If printing to the raw device, stty(1) settings can be changed, altering the output. If printing using the lp(1) subsystem, the STREAMS module ldterm will be pushed onto the stream automatically and will handle all canonical processing.

The ioctl(2) system calls available are a subset of those available to terminals and are discussed in depth in the termio(7) and termios(7) manpages. Because printers appear as write-only terminals, modifying the input flags for any of these ioctls has no effect on the driver. A list of the supported calls and a brief description follows.

EUC_MSAVE, EUC_MREST, EUC_IXLOFF, EUC_IXLON, EUC_OXLOFF, EUC_OXLON
: These ioctls are for international character handling and will be utilized in the future. They are simply acknowledged. For more information about the proper handling of these ioctls, refer to the *STREAMS Programming Guide*.

TCGETS
: The argument is a pointer to a termios structure. The current printer parameters are retrieved and stored in that structure.

TCSETS
: The argument is a pointer to a termios structure. The current printer parameters are set from the values stored in that structure. The change is immediate.

TCSETSW
: The argument is a pointer to a termios structure. The current printer parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.

TCSETSF
: The argument is a pointer to a termios structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and the change occurs. Because there are no input characters from a printer, this command has the same effect as the TCSETSW command.

TCGETA
: The argument is a pointer to a termio structure. The current terminal parameters are retrieved and parameters that can be stored in a termio structure are stored in that structure.

TCSETA
: The argument is a pointer to a termio structure. Those terminal parameters that can be stored in a termio structure are set from the values stored in that structure. The change is immediate.

TCSETAW
: The argument is a pointer to a termio structure. Those terminal parameters that can be stored in a termio structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.

TCSETAF
: The argument is a pointer to a termio structure. Those terminal parameters that can be stored in a termio structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and the change occurs. Because there are no input characters from a printer, this command has the same effect as the TCSETAW command.

TCSBRK
: This command is acknowledged but no action takes place.

TCFLUSH
: This command is transformed by ldterm into the STREAMS message M_FLUSH. The transformation only takes place if ldterm has been pushed onto the stream either by the lp subsystem or by the user.

TCXONC
: This command is transformed by ldterm into a STREAMS message, M_START or M_STOP, depending on which message is appropriate. The transformation only takes place if ldterm has been pushed onto the stream either by the lp subsystem or by the user.

For more information about the above ioctls and error messages generated by them, see termio(7).

FILES

/dev/xedt/lp187_c0
: on the MVME187 CPU.

/dev/printer/lp187_c0d0
: on the MVME187 CPU

/dev/xedt/lp167_c0
: on the MVME167 CPU

/dev/printer/lp167_c0d0
: on the MVME167 CPU

SEE ALSO

lp(1), stty(1), lpadmin(1M), ioctl(2), terminfo(4), ldterm(7), mvme187(7), mvme167(7), termio(7), termios(7).

NAME

m376 - MVME376 Local Area Network Interface

SYNOPSIS

```
#include <sys/dlpi.h>
#include <sys/macioctl.h>

fd = open("/dev/m376_c0", O_RDWR);
```

DESCRIPTION

The MVME376 is a VMEbus Local Area Network Controller for Ethernet and IEEE 802.3 compatible networks. The MVME376 utilizes the on-board combination of an Am7990 Local Area Network Controller (LANCE), an Am7992B Serial Interface Adapter (SIA), and 256Kbytes of dual ported RAM. The m376 device driver supports TCP/IP and OSI protocol stacks. A maximum of 4 (four) boards may be configured in a single system.

The m376 is a STREAMS-based software driver used with the MVME376 Ethernet board. The m376 interface conforms to the Data Link Provider Interface (DLPI). In addition, the m376 driver accepts the MAC management commands specified in the MAC Provider Interface (MPI).

The m376 driver can be opened directly, or indirectly from the clone device driver. During the TCP/IP startup, the m376 device is clone opened and linked to the IP and ARP STREAMS modules via the slink command. From then on, m376 converts all the outgoing packets received from IP/ARP to the format defined by the MVME376 board and then passes these packets to the board. If the OSI-DP package is installed on the system and linked into the kernel, the m376 driver will accept outgoing packets from the DLR (OSI LLC1) module.

Upon receiving incoming packets from the MVME376 board, m376 converts these packets to the STREAMS-based DLPI format messages and passes these packets to the appropriate user (e.g., ARP, IP, or DLR).

The mvme376 namer program, creates or deletes the device special files for the m376 driver at boot time. The device special filenames are composed of the string m376_c*y*, where *y* is the controller number. Controllers are numbered beginning at 0. The device special filename for the first controller in the system is /dev/m376_c0, for the second controller (if the system has one) is /dev/m376_c1, and so on.

An m376 node major device number is the major device number of the clone device driver. An m376 minor device number is the major number of the m376 device, found in /etc/master.d/mvme376, concatenated with the board number corresponding to this device. See intro(7) for the pictorial representation of the minor device number as passed to the device driver. For the m376 device driver, the bit fields in the minor format are defined as:

> The BOARD bits define the board device number. Boards are numbered from 0. The maximum board device number supported is 3.
>
> The MAJOR # bits correspond to the real major number of the m376 device as specified in the file /etc/master.d/mvme376.

The device node name is also used as the Ethernet network interface name by cenet in the network database file /etc/strcf and by ifconfig in the script /etc/inet/rc.inet.

Each m376 device may have up to seven (7) minor devices open simultaneously.

USAGE

STREAM Message Processing

The following are the types of STREAMS messages the driver can process:

M_PROTO/M_PCPROTO

Six DLPI protocol message types are supported: DL_INFO_REQ, DL_UNITDATA_REQ, DL_BIND_REQ, DL_UNBIND_REQ, DL_ENABMULTI_REQ, and DL_DISABMULTI_REQ. Unsupported message types that are received cause an error message of type dl_error_ack_t with dl_errno set to DL_NOTSUPPORTED to be sent back up the stream.

DL_INFO_REQ is a request for driver information. Driver information is passed back up the stream in a message of type dl_info_ack_t with dl_primitive set to DL_INFO_ACK. However, if enough memory is not available for the driver information, an error message of type dl_error_ack_t is sent back up the stream with dl_primitive set to DL_ERROR_ACK.

DL_UNITDATA_REQ is a request to transmit data. The message is in the dl_unitdata_req_t format. The driver will process this message and send data to the appropriate destination address. Most errors that can occur during this message are turned around in the message itself and sent back up stream in a message with dl_primitive set to DL_UDERROR_IND. If enough memory is not available for processing, an error message of type dl_error_ack_t is sent back up the stream with dl_primitive set to DL_ERROR_ACK.

DL_BIND_REQ is a request to bind a service access point (SAP) to the minor device number associated with the current stream. The request message is of type dl_bind_req_t. A SAP type, as long as it is valid, is assumed to be an Ethernet binding if it is not equal to IEEE8023_TYPE. Any Ethernet type can be used as a binding SAP. Only one stream may use IEEE8023_TYPE as a SAP. All IEEE802.3 frames will be sent up this stream. If the OSI-DP package has been installed, the DLR module will bind to this SAP and will receive all 802.3 frames. Once the stream has been bound, an acknowledgement message type dl_bind_ack_t is sent back up the stream. Errors generated during the processing of this message that cause an error message of type dl_error_ack_t to be sent back up the stream are: stream already bound, bad sap value, and cannot allocate memory for acknowledgement.

DL_UNBIND_REQ is a request to unbind the minor device associated with the current stream. Errors generated during message processing that cause an error message of type dl_error_ack_t are: minor device is not bound and cannot allocate enough memory for acknowledgement. An acknowledgement message of type dl_ok_ack_t is generated when the stream has been unbound.

DL_ENABMULTI_REQ is a request to enable a multicast address on a per-stream basis. An individual stream may have a maximum of sixty-four multicast addresses in its table, subject to the following limitation. There may be no more than sixty-four unique addresses for all streams associated with each controller. An acknowledgement message of type `dl_ok_ack_t` is generated if the request is valid. A message of type `dl_error_ack_t` is generated with `dl_primitive` set to DL_BADADDR if the multicast address is invalid or `dl_primitive` set to DL_TOOMANY if there is no space left in the controller's multicast table.

DL_DISABMULTI_REQ is a request to disable a multicast address on a per-stream basis. An acknowledgement message of type `dl_ok_ack_t` is generated if the request is valid. A message of type `dl_error_ack_t` is generated with `dl_primitive` set to DL_BADADDR if the multicast address is invalid or `dl_primitive` set to DL_NOTENAB if the requested address is not currently enabled.

M_IOCTL

`ioctl` commands are received in messages of type `iocblk`. There are many `ioctl` commands supported by the driver. Command data must be stored in a connected message block type M_DATA. Some commands do not require M_DATA blocks; M_DATA block requirements are listed. Data passed back upstream is always contained in an M_DATA block. All of the `ioctl` `#defines` used can be found in the file `include/sys/macioctl.h`.

A description of user `ioctl` stream messages can be found under the I_STR command in `streamio`(7). A sample code extract can be found in the *STREAMS Mechanism* chapter of the *STREAMS Programming Guide*.

MACDELAMCA is a request to delete all multicast table entries on the controller associated with this stream. This command does not require an M_DATA block.

MACDELMCA is a request to delete one multicast address from a multicast table on a per-stream basis. This command requires an M_DATA block of type `mc_frame`.

MACGETIA is a type of request to return the Ethernet address of the LANCE controller associated with the current queue. This command does not require an M_DATA block.

MACGETMCA is a request to return the entire multicast table for the controller associated with the current queue. This command does not require an M_DATA block.

MACGETSTAT is a request to return a statistic the driver has been gathering. A returned value of -1 indicates the statistic was not available. This command requires an M_DATA block. The data block is an array of structures. Each structure has the following format (see `macioctl.h`):

```
struct macstat {
long name ;
long value ;
}
```

A table of number defines and their descriptions follow:

MACGETSTAT	
Name	Description
MACSTAT_DEV_TIMEOUTS	total number of device timeouts
MACSTAT_XMITED	number of successful transmits
MACSTAT_XMITED_DEF	number of deferred transmits
MACSTAT_XMITED_1COLL	number of transmits with >/=1 collision
MACSTAT_COLLISIONS	total number of collisions
MACSTAT_NOXMIT_BUFF	total number dropped frames because of no STREAM buffer
MACSTAT_NOXMIT_COLL	number of frames dropped due to excess collisions
MACSTAT_RECVD	number of frames successfully received
MACSTAT_RECVD_CKSUM	number of CRC errors
MACSTAT_RECVD_ALIGN	number of frames with alignment errors
MACSTAT_NORECV_RES	number of frames dropped because of resource lack
MACSTAT_NORECV_LENGTH	number of frames dropped because of bad length
MACSTAT_RECVD_MCAST	number of multicast frames received
MACSTAT_XMITED_MCAST	number of multicast frames transmitted
MACSTAT_NORECV_MCAST	number of multicast frames rejected
MACSTAT_NORECV_TYPE	number of frames dropped because of unbound type
MACSTAT_NOXMIT_CARRIER	number of times lost carrier
MACSTAT_NOXMIT_CTS	number of times lost CTS
MACSTAT_DMA_ERRORS	number of DMA errors
MACSTAT_RECVD_BCAST	number broadcast frames received
MACSTAT_OUT_OF_WINDOW	number of late collisions
MACSTAT_XMITED_BCAST	number of broadcast frames transmitted

MACSETIA is a request to set the Ethernet address for the LANCE controller associated with the current stream. After executing MACSETIA, the networking subsystem *must* be stopped and then restarted. The address is immediately changed in the LANCE and the non-volatile RAM on the cpu board.

MACSETMCA is a request to add one multicast address to a multicast table on a per-stream basis. This command requires an M_DATA block of type `mc_frame`. A multicast address must have the least significant bit of byte[0] of the Ethernet address set. An individual stream may have a maximum of sixty-four multicast addresses in its table, subject to the following limitation. There may be no more than sixty-four addresses for all streams associated with each controller.

SIOCGENADDR is a type of request to return the Ethernet address of the LANCE controller associated with the current queue. This command requires an M_DATA block of type `struct ifreq`.

`M_FLUSH`
: If the command is a read queue flush, the read queue of the driver is flushed and the message is passed back up stream. If the command is a write queue flush, the write queue of the driver is flushed.

FILES

/dev/m376_*
/usr/include/sys/dlpi.h
/usr/include/sys/macioctl.h
/usr/include/sys/mvme376.h

SEE ALSO

ifconfig(1M), mvme376(1M), slink(1M), strace(1M), edt_data(4), master(4), strcf(4N), arp(7), clone(7), intro(7), ip(7), streamio(7).
McGrath, G., *A STREAMS-based Data Link Provider Interface (DLPI)*, Version 1.3, AT&T Bell Laboratories, Summit, N.J., February 1989
LT-610 Programmer Guide, Preliminary version, Retix, Santa Monica, CA, 1991

NAME

mailcnfg - initialization information for mail and rmail

DESCRIPTION

The /etc/mail/mailcnfg file contains initialization information for the mail and rmail commands. Each entry in mailcnfg consists of a line of the form

Keyword = *Value*

Leading whitespace, whitespace surrounding the equal sign, and trailing whitespace is ignored. *Keyword* may not contain embedded whitespace, but whitespace may appear within *Value*. Undefined keywords or badly formed entries are silently ignored.

Keyword Definitions

DEBUG
: Takes the same values as the -x invocation option of mail. This provides a way of setting a system-wide debug/tracing level. Typically DEBUG is set to a value of 2, which provides minimal diagnostics useful for debugging mail and rmail failures. The value of the -x mail invocation option will override any specification of DEBUG in mailcnfg.

CLUSTER
: To identify a closely coupled set of systems by one name to all other systems, set *Value* to the cluster name. This string is used to supply the ...remote from... information on the From header line rather than the system nodename returned by uname(2).

FAILSAFE
: In the event that the /var/mail directory is accessed via RFS or NFS within a cluster (see CLUSTER above), provisions must be made to allow for the directory not being available when local mail is to be delivered (remote system crash, RFS or NFS problems, and so on). *Value* is a string that indicates where to forward the current message for delivery. Typically this is the remote system that actually *owns* /var/mail. In this way, the message is queued for delivery to that system when it becomes available. For example, assume a cluster of systems (sysa, sysb, sysc) where /var/mail is physically mounted on sysc and made available to the other machines via RFS or NFS. If sysc were to crash, the RFS/NFS-accessible /var/mail would become unavailable and local deliveries of mail would go to /var/mail on the local system. When /var/mail is re-mounted via RFS/NFS, all messages deposited in the local directory would be hidden and essentially lost. To prevent this, if FAILSAFE is defined in mailcnfg, mail and rmail check for the existence of /var/mail/:saved, a required subdirectory. If this subdirectory does not exist, mail assumes that the RFS/NFS-accessible /var/mail is not available and invokes the failsafe mechanism of automatically forwarding the message to *Value*. In this example *Value* would be sysc!%n. The %*n* keyword is expanded to be the recipient name [see mail(1) for details] and thus the message would be forwarded to sysc!*recipient_name*. Because sysc is not

	available, the message remains on the local system until sysc is available, and then sent there for delivery.
DEL_EMPTY_MFILE	If not specified, the default action of mail and rmail is to delete empty mailfiles if the permissions are 0660 and to retain empty mailfiles if the permissions are anything else. If *Value* is yes, empty mailfiles are always deleted, regardless of file permissions. If *Value* is no, empty mailfiles are never deleted.
DOMAIN	This string is used to supply the system domain name in place of the domain name returned by getdomainame(3).
SMARTERHOST	This string may be set to a smarter host which may be referenced within the mail surrogate file via %X.
%*mailsurr_keyword*	As described in mailsurr(4), certain pre-defined single letter keywords are textually substituted in surrogate command fields before they are executed. While none of the predefined keywords may be changed in meaning, new ones may be defined to provide a shorthand notation for long strings (such as /usr/lib/mail/surrcmd) which may appear repeatedly within the mailsurr file. Upper case letters are reserved for future use and will be ignored if encountered here.

FILES

/etc/mail/mailcnfg
/etc/mail/mailsurr
/var/mail/:saved
/usr/lib/mail/surrcmd

SEE ALSO

mail(1) uname(2), getdomainame(3), mailsurr(4).

NOTES

If /var/mail is accessed via RFS or NFS and the subdirectory /var/mail/:saved is not removed from the local system, the FAILSAFE mechanism will be subverted.

NAME

mailsurr - surrogate commands for routing and transport of mail

DESCRIPTION

The mailsurr file contains routing and transport surrogate commands used by the mail command. Each entry in mailsurr has three whitespace-separated, single quote delimited fields:

'sender' *'recipient'* *'command'*

or a line that begins

```
Defaults:
```

Entries and fields may span multiple lines, but leading whitespace on field continuation lines is ignored. Fields must be less than 1024 characters long after expansion (see below).

The sender and recipient fields are regular expressions. If the sender and recipient fields match those of the message currently being processed, the associated command is invoked.

The *command* field may have one of the following five forms:

A[ccept]
D[eny]
T[ranslate] R=[|]*string*
< S=. . .;C=. . .;F=. . .; *command*
> *command*

Regular Expressions

The sender and recipient fields are composed of regular expressions (REs) which are digested by the regexp(5) compile and advance procedures in the C library. The regular expressions matched are those from ed(1), with simple parentheses () playing the role of \(\) and the addition of the + and ? operators from egrep(1). Any single quotes embedded within the REs *must* be escaped by prepending them with a backslash or the RE is not interpreted properly.

The mail command prepends a circumflex (^) to the start and appends a dollar sign ($) to the end of each RE so that it matches the entire string. Therefore it would be an error to use ^*RE*$ in the sender and recipient fields. To provide case insensitivity, all REs are converted to lower case before compilation, and all sender and recipient information is converted to lower case before comparison. This conversion is done only for the purposes of RE pattern matching; the information contained within the message's header is *not* modified.

The sub-expression pattern matching capabilities of regexp may be used in the command field, that is, (. . .), where 1 ≤ *n* ≤ 9. Any occurrences of *n* in the replacement string are themselves replaced by the corresponding (. . .) substring in the matched pattern. The sub-expression fields from both the sender and recipient fields are accessible, with the fields numbered 1 to 9 from left to right.

Accept and Deny Commands

Accept instructs rmail to continue its processing with the mailsurr file, but to ignore any subsequent matching Deny. That is, unconditionally accept this message for delivery processing. Deny instructs rmail to stop processing the mailsurr file and to send a negative delivery notification to the originator of the message.

Whichever is encountered first takes precedence.

Translate Command

`Translate` allows optional on-the-fly translation of recipient address information. The *recipient* replacement string is specified as R=*string*.

For example, given a command line of the form

```
'.+' '([^!]+)@(.+)\.EUO\.ATT\.com' 'Translate R=attmail!\\2!\\1'
```

and a recipient address of `rob@sysa.EUO.ATT.COM` the resulting recipient address would be `attmail!sysa!rob`.

Should the first character after the equal sign be a '|', the remainder of the string is taken as a command line to be directly executed by `rmail`. If any sh(1) syntax is required (metacharacters, redirection, and so on), then the surrogate command must be of the form:

sh -c "*shell command line...*"

Special care must be taken to escape properly any embedded back-slashes and single or double quotes, since `rmail` uses double quoting to group whitespace delimited fields that are meant to be considered as a single argument to `execl`(2). It is assumed that the executed command will write one or more replacement strings on `stdout`, one per line. If more than one line is returned, each is assumed to be a different recipient for the message. This mechanism is useful for mailing list expansions. As stated above, any occurrences of *n* are replaced by the appropriate substring *before* the command is executed. If the invoked command does not return at least one replacement string (no output or just a newline), the original string is *not* modified. For example, the command line

```
'.+' '(.+)' 'Translate R=|/usr/bin/findpath \\1'
```

allows local routing decisions to be made.

If the recipient address string is modified, `mailsurr` is rescanned from the beginning with the new address(es), and any prior determination of `Accept` (see above) is discarded.

< *command*

The intent of a < command is that it is invoked as part of the transport and delivery mechanism, with the ready-for-delivery message available to the command at its standard input. As such, there are three conditions possible when the command exits:

Success — The command successfully delivered the message. What actually constitutes successful delivery may be different within the context of different surrogates. The `rmail` process assumes that no more processing is required for the message for the current recipient.

Continue — The command performed some function (logging remote message traffic, for example) but did not do what would be considered message delivery. The `rmail` process continues to scan the `mailsurr` file looking for some other delivery mechanism.

Failure — The command encountered some catastrophic failure. The `rmail` process stops processing the message and sends to the originator of the message a non-delivery notification that includes any `stdout` and `stderr` output generated by the command.

The semantics of the < command field in the `mailsurr` file allow the specification of exit codes that constitute success, continue, and failure for each surrogate command individually. The syntax of the exit state specification is:

< WS [*exit_state_id*=*ec*[, *ec*[,. . .]] ;][*exit_state_id*=*ec*[,*ec*[,. . .]] ;
[. . .]]] WS *surrogate_cmd_line*

WS is whitespace. *exit_state_id* can have the value `S`, `C`, or `F`. *exit_state_id*s can be specified in any order. *ec* can be:

any integer $0 \le n \le 255$ [Negative exit values are not possible. See `exit`(2) and `wait`(2).]

a range of integers of the form *lower_limit-upper_limit* where the limits are $\ge$ 0 and $\le$ 255, and

`*`, which implies *anything*

For example, a command field of the form:

`'< S=1-5,99;C=0,12;F=*;` *command* `%R'`

indicates that exit values of 1 through 5, and 99, are to be considered success, values of 0 (zero) and 12 indicate continue, and that anything else implies failure. If not explicitly supplied, default settings are `S=0;C=*;`.

It may be possible for ambiguous entries to exist if two exit states have the same value, for example, `S=12,23;C=*;F=23,52;` or `S=*;C=9;F=*;`. To account for this, `rmail` looks for *explicit* exit values (that is, *not* "*") in order of success, continue, failure. Not finding an explicit match, `rmail` then scans for "*" in the same order.

It is possible to eliminate an exit state completely by setting that state's value to an impossible number. Since exit values must be between 0 and 255 (inclusive), a value of 256 is a good one to use. For example, if you had a surrogate command that was to log all message traffic, a `mailsurr` entry of

```
'(.+)' '(.+)' '<S=256;C=*; /usr/lib/mail/surrcmd/logger \\1 \\2'
```

would always indicate continue.

Surrogate commands are executed by `rmail` directly. If any shell syntax is required (metacharacters, redirection, and so on), then the surrogate command must be of the form:

`sh -c "`*shell command line. . .*`"`

Special care must be taken to properly escape any embedded back-slashes and other characters special to the shell as stated in the "Translate" section above.

If there are no matching < commands, or all matching < commands exit with a continue indication, `rmail` attempts to deliver the message itself by assuming that the recipient is local and delivering the message to `/var/mail/`*recipient*.

> command

The intent of a > command is that it is invoked *after* a successful delivery to do any post-delivery processing that may be required. Matching > commands are executed only if some < command indicates a successful delivery (see the previous section) or local delivery processing is successful. The `mailsurr` file is rescanned and all matching > commands, not just those following the successful < command, are executed in order. The exit status of an > command is ignored.

Defaults: Line

The default settings may be redefined by creating a separate line in the `mailsurr` file of the form

`Defaults:` [`S=...;`][`C=...;`][`F=...;`]

`Defaults:` lines are honored and the indicated default values redefined when the line is encountered during the normal processing of the `mailsurr` file. Therefore, to redefine the defaults globally, the `Defaults:` line should be the first line in the file. It is possible to have multiple `Defaults:` lines in the `mailsurr` file, where each subsequent line overrides the previous one.

Surrogate Command Keyword Replacement.

Certain special sequences are textually-substituted in surrogate commands before they are invoked:

`%n`	the recipient's full name.
`%R`	the full return path to the originator (useful for sending replies, delivery failure notifications, and so on)
`%c`	value of the `Content-Type:` header line if present.
`%C`	"`text`" or "`binary`", depending on an actual scan of the content. This is independent of the value of any `Content-Type` header line encountered (useful when calling `ckbinarsys`.)
`%S`	the value of the `Subject:` header line, if present.
`%l`	value of the `Content-Length:` header line.
`%L`	the local system name. This will be either `CLUSTER` from `mailcnfg` or the value returned by `uname`.
`%U`	the local system name, as returned by `uname`.
`%X`	the value of `SMARTERHOST` in `mailcnfg`.
`%D`	the local domain name. This will be either `DOMAIN` from `mailcnfg`, or the value returned by `getdomainame`.
`\\`*n*	as described above, the corresponding (. . .) substring in the matched patterns. This implies that the `regexp` limitation of 9 substrings is applied to the sender and recipient REs collectively.
`%`*keywords*	Other keywords as specified in `/etc/mail/mailcnfg`. See `mailcnfg`(4).

The sequences `%L`, `%U`, `%D`, and `%`*keywords* are permitted within the sender and recipient fields as well as in the command fields.

An example of the `mailsurr` entry that replaces the `uux` "built-in" of previous versions of `rmail` is:

```
'.+'  '([^@!]+)!(.+)'  '< /usr/bin/uux - \\1!rmail (\\2)'
```

Mail Surrogate Examples

Some examples of mail surrogates include the distribution of message-waiting notifications to LAN-based recipients and lighting Message-Waiting Lamps, the ability to mail output to printers, and the logging of all `rmail` requests between remote systems (messages passing through the local system). The following is a sample `mailsurr` file:

```
#
# Some common remote mail surrogates follow. To activate any
# or all of them, remove the '#' (comment indicators) from
# the beginning of the appropriate lines. Remember that they
# will be tried in the order they are encountered in the file,
# so put preferred surrogates first.

#     Prevent all shell meta-characters
'.+'  '.*[';&|^<>()].*'        'Deny'

#     Map all names of the form local-machine!user -> user
'.+'  '%L!(.+)'                'Translate R=\1'

#     Map all names of the form uname!user -> user
#     Must be turned on when using mail in a cluster environment.
#'.+' '%U!(.+)'                'Translate R=\1'

#     Map all names of the form user@host -> host!user
'.+'  '([^!@]+)@(.+)'          'Translate R=\2!\1'

#     Map all names of the form host.uucp!user -> host!user
'.+'  '([^!@]+)\.uucp!(.+)'  'Translate R=\1!\2'

#     Map all names of the form host.local-domain!user -> host!user
#     DOMAIN= within /etc/mail/mailcnfg will override getdomainame(3).
'.+'  '([^!@]+)%D!(.+)'        'Translate R=\1!\2'

#     Allow access to 'attmail' from remote system 'sysa'
'sysa!.*'    'attmail!.+'    'Accept'

#     Deny access to 'attmail' from all other remotes
'.+!.+'      'attmail!.+'    'Deny'

#     Send mail for 'laser' to attached laser printer
#     Make certain that failures are reported via return mail.
'.+'  'laser'      '< S=0;F=*; lp -dlaser'

#     Run all local names through the mail alias processor
#
'.+'  '[^!@]+'           'Translate R=|/usr/bin/mailalias %n'

#     For remote mail via nusend
```

```
#'.+'  '([^!]+)!(.+)'   '< /usr/bin/nusend -d \\1 -s -e -!"rmail \\2" -'

#      For remote mail via usend
'.+'   '([^!]+)!(.+)'
          '< /usr/bin/usend -s -d\\1 -uNoLogin -!"rmail \\2" - '

#      For remote mail via uucp
'.+'   '([^!@]+)!.+'   '<S=256;C=0;
              /usr/lib/mail/surrcmd/ckbinarsys -t %C -s \\1'
'.+'   '([^!@]+)!(.+)'   '< /usr/bin/uux - \\1!rmail (\\2)'

#      For remote mail via smtp
#'.+'  '([^!@]+)!(.+)'         '< /usr/lib/mail/surrcmd/smtpqer %R %n'

#      If none of the above work, then let a router change the address.
#'.+'  '.*[!@].*'       'Translate R=| /usr/lib/mail/surrcmd/smail -A %n'

#      If none of the above work, then ship remote mail off to a smarter host.
#      Make certain that SMARTERHOST= is defined within /etc/mail/mailcnfg.
#'.+'  '.*[!@].*'               'Translate R=%X!%n'

#      Log successful message deliveries
'(.+)' '(.+)' '>/usr/lib/mail/surrcmd/logger \1 \2'
```

Note that invoking `mail` to read mail does not involve the `mailsurr` file or any surrogate processing.

Security

Surrogate commands execute with the permissions of `rmail` (user ID of the invoker, group ID of mail). This allows surrogate commands to validate themselves, checking that their effective group ID was `mail` at invocation time. This requires that all additions to `mailsurr` be scrutinized before insertion to prevent any unauthorized access to users' mail files. All surrogate commands are executed with the path `/usr/lib/mail/surrcmd:/usr/bin`.

Debugging New mailsurr Entries

To debug `mailsurr` files, use the `-T` option of the `mail` command. The `-T` option requires an argument that is taken as the pathname of a test `mailsurr` file. If null (as in `-T ""`), the system `mailsurr` file is used. Enter

`mail -T` *test_file recipient*

and some trivial message (like "`testing`"), followed by a line with either just a dot ("`.`") or a cntl-D. The result of using the `-T` option is displayed on standard output and shows the inputs and resulting transformations as `mailsurr` is processed by the `mail` command for the indicated *recipient*.

Mail messages will never be sent or delivered when using the `-T` option.

FILES

`/etc/mail/mailsurr`

/usr/lib/mail/surrcmd/*	surrogate commands
/etc/mail/mailcnfg	initialization information for mail

SEE ALSO

ckbinarsys(1M), ed(1), egrep(1), mail(1), sh(1), uux(1), exec(2), exit(2), wait(2), getdomainname(3) popen(3), mailcnfg(4), regexp(5).

NOTES

It would be unwise to install new entries into the system mailsurr file without verifying at least their syntactical correctness via 'mail -T ...' as described above.

NAME

master - master configuration database

DESCRIPTION

The master configuration database is a collection of files. Each file contains configuration information for a device or module that may be included in the system. A file is named with the module name to which it applies. This collection of files is maintained in a directory called /etc/master.d. Each file has an identical format. For convenience, this collection of files will be referred to as the master file, as though it were a single file. Treating the master file as a single file allows a reference to the master file to be understood to mean the individual file in the master.d directory that corresponds to the name of a device or module. The file is used by the mkboot(1M) program to obtain device information to generate the device driver and configurable module files. It is also used by the sysdef(1M) program to obtain the names of supported devices. master consists of two parts; they are separated by a line with a dollar sign ($) in column 1. Part 1 contains device information for both hardware and software devices, and loadable modules. Part 2 contains parameter declarations used in Part 1. Any line with an asterisk (*) in column 1 is treated as a comment.

Part 1. Description

Hardware devices, software drivers and loadable modules are defined with a line containing the following information. Field 1 must begin in the left-most position on the line. Fields are separated by white space (tab or blank).

Field 1: element characteristics:

o	specify only once
r	required device
b	block device
c	character device
h	hardware driver
d	dispatch driver
j	file-system driver
n	new-style device driver
e	executable-type driver
t	initialize cdevsw[].d_ttys
s	software driver
f	STREAMS driver
m	STREAMS module
M	multi-threaded driver or module
[0-9]	processor number for a staticly bound driver or module
x	not a driver; a loadable module
none	no flags for this driver or module

Note: A streams device or module which has no M flag or processor number in Field 1, will be staticly bound to the boot processor. For other drivers, the module will be allowed to float between processors, but will only execute on one processor at a time.

Field 2: handler prefix (4 characters maximum)
Field 3: hardware/software driver external major number; "-" if not a software/hardware driver, or to be assigned during execution of `drvinstall`(1M)
Field 4: number of sub-devices per device; "-" if none
Field 5: dependency list (optional); this is a comma-separated list of other drivers or modules that must be present in the configuration if this module is to be included

For each module, two classes of information are required by `mkboot`: external routine references and variable definitions. Routine and variable definition lines begin with white space and immediately follow the initial module specification line. These lines are free form, thus they may be continued arbitrarily between non-blank tokens as long as the first character of a line is white space.

Part 1. Routine Reference Lines

If the UNIX system kernel or other dependent module contains external references to a module, but the module is not configured, then these external references would be undefined. Therefore, the routine reference lines are used to provide the information necessary to generate appropriate dummy functions at boot time when the driver is not loaded. The format of a routine reference is as follows:

routine_name `()` *action*

The valid actions and their meanings are:

`{}`	*routine_name* `(){}`
`{nosys}`	`{return nosys();`
`{nodev}`	`{return nodev();}`
`{false}`	`{return 0;}`
`{true}`	`{return 1;}`
`{nopkg}`	`{return nopkg();}`
`{noreach}`	panic the system

Part 1. Variable Definition Lines

Variable definition lines are used to generate all variables required by the module. The variable generated may be of arbitrary size, be initialized or not, or be arrays containing an arbitrary number of elements. Variable references are defined as follows:

Field 1: *variable_name*

Field 2: `[` *expr* `]` - optional field used to indicate array size

Field 3: `(`*length*`)` - required field indicating the size of the variable

Field 4: `={` *expr*`, ... }` - optional field used to initialize individual elements of a variable

The *length* field is mandatory. It is an arbitrary sequence of length specifiers, each of which may be one of the following:

`%i`	an integer
`%l`	a long integer
`%s`	a short integer

`%c`	a single character
`%`*number*	a field which is *number* bytes long
`%`*number* `c`	a character string which is *number* bytes long

For example, the length field

```
( %8c %1 %0x58 %1 %c %c )
```

could be used to identify a variable consisting of a character string 8-bytes long, a long integer, a 0x58 byte structure of any type, another long integer, and two characters. Appropriate alignment of each `%` specification is performed (`%`*number* is word-aligned) and the variable length is rounded up to the next word boundary during processing.

The expressions for the optional array size and initialization are infixed expressions consisting of the usual operators for addition, subtraction, multiplication, and division: +, -, *, and /. Multiplication and division have the higher precedence, but parentheses may be used to override the default order. The builtin functions `min` and `max` accept a pair of expressions, and return the appropriate value. The operands of the expression may be any mixture of the following:

`&`*name* — address of *name*, where *name* is any symbol defined by the kernel, any module loaded, or any variable definition line of any module loaded

`#`*name* — `sizeof` *name* where *name* is any variable name defined by a variable definition for any module loaded; the size is that of the individual variable—not the size of an entire array

`#C` — number of controllers present; this number is determined by the EDT for hardware devices, or by the number provided in the system file for non-hardware drivers or modules

`#C(`*name*`)` — number of controllers present for the module *name*; this number is determined by the EDT for hardware devices, or by the number provided in the system file for non-hardware drivers or modules

`#D` — number of devices per controller taken directly from the current master file entry

`#D(`*name*`)` — number of devices per controller taken directly from the master file entry for the module *name*

`#M` — the internal major number assigned to the current module if it is a device driver; zero of this module is not a device driver

`#M(`*name*`)` — the internal major number assigned to the module *name* if it is a device driver: zero if that module is not a device driver

name — value of a parameter as defined in the second part of `master`

number — arbitrary number (octal, decimal, or hex allowed)

string — a character string enclosed within double quotes (all of the character string conventions supported by the C language are allowed); this operand has a value which is the address of a character array containing the specified string

When initializing a variable, one initialization expression should be provided for each `%i`, `%l`, `%s`, or `%c` of the length field. The only initializers allowed for a `%`*number* `c` are either a character string (the string may not be longer than *number*), or an explicit zero. Initialization expressions must be separated by commas, and variable initialization proceeds element by element. Note that `%`*number* specification cannot be initialized—they are set to zero. Multiple elements of an array may be initialized; uninitialized elements are set to zero. If there are more initializers than size specifications, it is an error and execution of the `mkboot` program is aborted. In the case of an array, `mkboot` will report an error only if the array's dimension is a literal. C UNIX will report an error if the dimension is a symbol or expression and too many initializers are given. If there are fewer initializations than size specifications, zeros will be used to pad the variable. For example:

```
={ "V2.L1", #C*#D, max(10,#D), #C(OTHER), #M(OTHER) }
```

would be a possible initialization of the variable whose length field was given in the preceding example.

Part 2. Description

Parameter declarations may be used to define a value symbolically. Values can be associated with identifiers and these identifiers may be used in the *variable definition* lines. Parameters are defined as follows:

identifier = *value*

The *identifier* may have a maximum of 8 characters. The *value* may be a number (decimal, octal, or hex) or a string.

EXAMPLE

A sample `master` file for a tty device driver would be named `atty` if the device appeared in the EDT as `ATTY`. The driver is a character device, the driver prefix is `at`. In addition, another driver named `ATLOG` is necessary for the correct operation of the software associated with this device.

```
*FLAG PREFIX SOFT #DEV DEPENDENCIES/VARIABLES
 tca   at     -    2     ATLOG
                          atpoint(){false}
                          at_tty[#C*#D] (%0x58)
                          at_cnt(%i) ={ #C*#D }
                          at_logmaj(%i) ={ #M(ATLOG) }
                          at_id(%8c) ={ ATID }
                          at_table(%i%l%31%s)
                              ={ max(#C,ATMAX),
                                 &at_tty,
                                 #C }
$
ATID = "fred"
ATMAX = 6
```

This `master` file causes a routine named `atpoint` to be generated by the boot program if the `ATTY` driver is not loaded, and there is a reference to this routine from any other module loaded. When the driver is loaded, the variables `at_tty`, `at_cnt`, `at_logmaj`, `at_id`, and `at_table` are allocated and initialized as specified. Because of the `t` flag, the `d_ttys` field in the character device switch table is initialized to point to `at_tty` (the first variable definition line contains the variable whose address will be stored in `d_ttys`). The ATTY driver would reference these variables by coding:

```
extern struct tty at_tty[];
extern int at_cnt;
extern int at_logmaj;
extern char at_id[8];
extern struct {
        int member1;
        struct tty *member2;
        char junk[31];
        short member3;
        } at_table;
```

FILES

/etc/master.d/*

SEE ALSO

drvinstall(1M), mkboot(1M), sysdef(1M), system(4).

NAME

mem, kmem - core memory

DESCRIPTION

The file /dev/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 /dev/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 /dev/kmem is the same as /dev/mem except that kernel virtual memory rather than physical memory is accessed.

FILES

/dev/mem
/dev/kmem

NOTES

Some of /dev/kmem cannot be read because of write-only addresses or unequipped memory addresses.

NAME

memregion - core memory by region

DESCRIPTION

The special files in the directory /dev/memregion provide access to individual memory regions defined in the system's edt_data file. Each memory region has at least one entry named /dev/memregion/*N*, where *N* is the id specified in the edt_data file. Each region can also have an additional alias in the directory.

Offsets in a /dev/memregion file correspond to byte offsets from the start of the associated memory region, *not* to physical addresses within the region.

FILES

/dev/memregion/*

NOTES

The special file /dev/mem corresponds to the union of all files in /dev/memregion. Offsets in /dev/mem correspond to physical addresses, so there will be ''holes'' if the memory regions are not contiguous.

SEE ALSO

edt_data(4), mem(7).

NAME

mnttab - mounted file system table

SYNOPSIS

#include <sys/mnttab.h>

DESCRIPTION

The file /etc/mnttab contains information about devices that have been mounted by the mount command. The information is in the following structure, defined in sys/mnttab.h:

```
struct  mnttab {
        char    *mnt_special;
        char    *mnt_mountp;
        char    *mnt_fstype;
        char    *mnt_mntopts;
        char    *mnt_time;
};
```

The fields in the mount table are space-separated and show the block special device, the mount point, the file system type of the mounted file system, the mount options, and the time at which the file system was mounted.

SEE ALSO

mount(1M), getmntent(1M), setmnt(1M).

NAME

mt - tape interface

DESCRIPTION

The files /dev/rmt/ctape? refer to cartridge tape controllers (CTC) and associated tape drives. The files /dev/rmt/ninetrack? refer to nine-track tape controllers and associated tape drives. These special device files and the /dev/rSA/ctape? and /dev/rSA/ninetrack? special files are linked to the respective controller specific names in the /dev/rmt directory.

The finc(1M), frec(1M), and labelit(1M) commands require the ctape magnetic tape filenames to work correctly with the CTC. No other CTC commands require these filenames.

FILES

/dev/rmt/ctape*
/dev/rmt/ninetrack*
/dev/rSA/ctape*
/dev/rSA/ninetrack*

SEE ALSO

finc(1M), frec(1M), labelit(1M)

NAME

mvme167 - MVME167 CPU

DESCRIPTION

The mvme167 is a CPU platform with an MC68040 MPU, 16, 32, 40, 48, or 64 MB of dual-ported onboard (mezzanine) memory, 8 KB of battery backup static RAM, 128 Kb of volatile static RAM, a time-of-day clock/calendar, an Ethernet transceiver interface (Intel 82596CA), four EIA-232-D serial communication ports (Cirrus Logic CD2400/2401), a SCSI-2 bus interface (NCR 53C710), a Centronics-compatible parallel printer port, configurable local and VMEbus address maps, four tick timers, and four ROM sockets of which two contain the MVME167BUG Debugger and Diagnostic Package.

SPECIAL CONSIDERATIONS

The mvme167 uses three integrated circuits for controlling the VMEbus interface (vmechip2), peripheral interrupts (pccchip2), and local memory (memc040). Unless otherwise specified, the configurable registers which control the memory, peripheral, or VMEbus interfaces are unchanged from what is described in the *MVME167BUG User's Manual*. This section describes those registers which are different from the ROM debugger settings.

The vmechip2 provides a mechanism for mapping onboard memory to the VMEbus (VMEbus accesses to this memory are issued on the local bus) and it provides mechanisms for mapping VMEbus addresses to the local bus (local bus accesses are issued on the VMEbus). All mappings are mapped one-to-one (a local bus access of 0xB0000000 is always converted to a VMEbus access of 0xB0000000 and vice versa). The following two tables describe how these mappings are set.

Local to VMEbus Mappings:

Memory Description	Attributes
Local Memory (0 .. DRAMSIZE - 1)	A32, A24, Write Posting
Local SRAM (0xFFFE0000 .. 0xFFE1FFFF)	A32

VMEbus to Local Mappings:

Memory Description	Attributes
General A32 VMEbus Memory (DRAMSIZE .. 0xEDFFFFFF)	A32, D32
General A24 VMEbus Memory (0xEE000000 .. 0xEEFFFFFF)	A24, D32
General A32 VMEbus Memory (0xEF000000 .. 0xEFFFFFFF)	A32, D32
A24 F-Page Memory (0xF0000000 .. 0xF0FFFFFF)	A24, D32
A32 F-Page Memory (0xF1000000 .. 0xFF7FFFFF)	A32, D32
VMEbus Short I/O (0xFFFF0000 .. 0xFFFFFFFF)	A16, D16

Both the F-Page and the Short I/O map decoders are enabled.

The vmechip2 controls the local bus to VMEbus requester. It is set so that VMEbus FAIR mode arbitration is used, the VMEbus is released when the transaction is completed, and the VMEbus request level has the value configured in the mvmecpu master.d file. The bus grant timeout timer is enabled, VMEbus access timeout value is set to 32 milliseconds, the VMEbus global timeout value is set to 256 microseconds, and the local bus timeout value is set to 8 microseconds.

The vmechip2 also controls various I/O related operations including DMA, a set of general purpose timers, and various local and VMEbus interrupts. All DMA registers are set to zero. Both timers' registers on the vmechip2 are initialized to zero and timer 1 is set up as a free running clock. The board control register is cleared, and the VMEbus control register word (0xFFF40048) has the MCLR bit (bit 11) set to 1 and all other bits reset to zero. The RESET button, ABORT, ACFAIL, write posting, parity, and all VMEbus interrupt levels are enabled. VMEbus interrupt request levels 1 through 7 are mapped to local interrupt request levels 1 through 7. The VMEX and VMEY interrupt vectors (used for interrupts generated by the vmechip2 itself) are set based on the interrupt vector values in the VMEX and VMEY entries of the edt_data file.

The pccchip2 controls all onboard peripherals. The high order 4 bits of the interrupt vector used by each of the onboard devices is set based on the interrupt vector level specified for the PCC2 module in the edt_data file. The two timers on the pcchip2 are initialized to an OFF state. Timer 1 is used by the operating system as a time base and is reinitialized when the system clock is started. General purpose I/O interrupts are disabled.

Each memory mezzanine is controlled by an memc040. Each of these has the bus clock register initialized based on the MPU speed and has parity detection and parity interrupts enabled.

FILES

/dev/conctl
/dev/console
/dev/contty
/dev/contty??
/dev/dsk/m167_c0d?s?
/dev/e1x7_c0d0
/dev/generic/m167_c0d?
/dev/nvr*
/dev/printer/lp167_c0d0
/dev/rdsk/m167_c0d?s?
/dev/rmt/m167_c0d?
/dev/xedt/lp1x7_c0
/dev/xedt/scsi1x7_c0

SEE ALSO

dcon(1M), mvmecpu(1M), scsi1x7(1M), console(7), cons1x7(7), e1x7(7), enet1x7(7), lp1x7(7), nvram(7), scsi1x7(7).

NAME

`mvme181` - MVME181 CPU

DESCRIPTION

The `mvme181` is a CPU platform with an MC88100 MPU, two MC88200 CMMUs, two RS-232C serial communications ports driven by a 68692 DUART, a battery backup real-time clock/calendar, 8 MB of dual-ported onboard DRAM, and 512 KB of firmware containing the MVME181BUG Debugger and Diagnostic Package.

SPECIAL CONSIDERATIONS

The timer on the 68682 DUART is used as the system time base.

FILES

`/dev/conctl`
`/dev/console`
`/dev/contty`
`/dev/contty??`

SEE ALSO

`dcon`(1M), `mvmecpu`(1M), `console`(7)
MVME181BUG Debugging Package User's Manual
MVME181 VMEmodule RISC Microcomputer User's Manual

NAME

mvme187 - MVME187 CPU

DESCRIPTION

The mvme187 is a CPU platform with an MC88100 MPU, two MC88200 CMMUs, 32, 40, 48, or 64 MB of dual-ported onboard (mezzanine) memory, 8 KB of battery backup static RAM, 128 Kb of volatile static RAM, a time-of-day clock/calendar, an Ethernet transceiver interface (Intel 82596CA), four EIA-232-D serial communication ports (Cirrus Logic CD2400/2401), a SCSI-2 bus interface (NCR 53C710), a Centronics-compatible parallel printer port, configurable local and VMEbus address maps, four tick timers, and four ROM sockets of which two contain the MVME187BUG Debugger and Diagnostic Package.

SPECIAL CONSIDERATIONS

The mvme187 uses three integrated circuits for controlling the VMEbus interface (vmechip2), peripheral interrupts (pccchip2), and local memory (memc040). Unless otherwise specified, the configurable registers which control the memory, peripheral, or VMEbus interfaces are unchanged from what is described in the *MVME187BUG User's Manual*. This section describes those registers which are different from the ROM debugger settings.

The vmechip2 provides a mechanism for mapping onboard memory to the VMEbus (VMEbus accesses to this memory are issued on the local bus) and it provides mechanisms for mapping VMEbus addresses to the local bus (local bus accesses are issued on the VMEbus). All mappings are mapped one-to-one (a local bus access of 0xB0000000 is always converted to a VMEbus access of 0xB0000000 and vice versa). The following two tables describe how these mappings are set.

Local to VMEbus Mappings:

Memory Description	Attributes
Local Memory (0 .. DRAMSIZE - 1)	A32, A24, Write Posting
Local SRAM (0xFFFE0000 .. 0xFFE1FFFF)	A32

VMEbus to Local Mappings:

Memory Description	Attributes
General A32 VMEbus Memory (DRAMSIZE .. 0xEDFFFFFF)	A32, D32
General A24 VMEbus Memory (0xEE000000 .. 0xEEFFFFFF)	A24, D32
General A32 VMEbus Memory (0xEF000000 .. 0xEFFFFFFF)	A32, D32
A24 F-Page Memory (0xF0000000 .. 0xF0FFFFFF)	A24, D32
A32 F-Page Memory (0xF1000000 .. 0xFF7FFFFF)	A32, D32
VMEbus Short I/O (0xFFFF0000 .. 0xFFFFFFFF)	A16, D16

Both the F-Page and the Short I/O map decoders are enabled.

The vmechip2 controls the local bus to VMEbus requester. It is set so that VMEbus FAIR mode arbitration is used, the VMEbus is released when the transaction is completed, and the VMEbus request level has the value configured in the mvmecpu master.d file. The bus grant timeout timer is enabled, VMEbus access timeout value is set to 32 milliseconds, the VMEbus global timeout value is set to 256 microseconds, and the local bus timeout value is set to 8 microseconds.

The vmechip2 also controls various I/O related operations including DMA, a set of general purpose timers, and various local and VMEbus interrupts. All DMA registers are set to zero. Both timers' registers on the vmechip2 are initialized to zero and timer 1 is set up as a free running clock. The board control register is cleared, and the VMEbus control register word (0xFFF40048) has the MCLR bit (bit 11) set to 1 and all other bits reset to zero. The RESET button, ABORT, ACFAIL, write posting, parity, and all VMEbus interrupt levels are enabled. VMEbus interrupt request levels 1 through 7 are mapped to local interrupt request levels 1 through 7. The VMEX and VMEY interrupt vectors (used for interrupts generated by the vmechip2 itself) are set based on the interrupt vector values in the VMEX and VMEY entries of the edt_data file.

The pccchip2 controls all onboard peripherals. The high order 4 bits of the interrupt vector used by each of the onboard devices is set based on the interrupt vector level specified for the PCC2 module in the edt_data file. The two timers on the pcchip2 are initialized to an OFF state. Timer 1 is used by the operating system as a time base and is reinitialized when the system clock is started. General purpose I/O interrupts are disabled.

Each memory mezzanine is controlled by an memc040. Each of these has the bus clock register initialized based on the MPU speed and has parity detection and parity interrupts enabled.

FILES

/dev/conctl
/dev/console
/dev/contty
/dev/contty??
/dev/dsk/m187_c0d?s?
/dev/e1x7_c0d0
/dev/generic/m187_c0d?
/dev/nvr*
/dev/printer/lp187_c0d0
/dev/rdsk/m187_c0d?s?
/dev/rmt/m187_c0d?
/dev/xedt/lp1x7_c0
/dev/xedt/scsi1x7_c0

SEE ALSO

dcon(1M), mvmecpu(1M), scsi1x7(1M), console(7), cons1x7(7), e1x7(7), enet1x7(7), lp1x7(7), nvram(7), scsi1x7(7).

NAME

mvme188 - MVME188 CPU

DESCRIPTION

The mvme188 is a CPU platform which consists of: one, two, or four MC88100 MPUs, two, four, or eight MC88200 CMMUs, between 16 MB and 128 MB of dual-ported onboard DRAM, 2 KB of battery backup RAM, configurable local and VMEbus address maps, two RS-232C serial communications ports driven by a 68692 DUART, four programmable timers, a battery backup real-time clock/calendar, and 512 KB of firmware containing the MVME188BUG Debugger and Diagnostic Package.

SPECIAL CONSIDERATIONS

The bus snooper(s) and data/code CMMU parity detection are enabled. The timer on the 68682 DUART is used as the system time base.

FILES

/dev/conctl
/dev/console
/dev/contty
/dev/contty??
/dev/nvr*

SEE ALSO

dcon(1M), mvmecpu(1M), console(7), nvram(7)
MVME188BUG Debugging Package User's Manual
MVME188 VMEmodule RISC Microcomputer User's Manual

NAME

mvme323 - MVME323 disk controller (For M68K only)

DESCRIPTION

mvme323 is a driver that provides a general interface to the MVME323 VMEbus disk controller module. The MVME323 controller supports up to four ESDI disks. The mvme323 driver supports up to eight MVME323 controllers per system.

Each disk connected to the MVME323 has the same major device number. Disks with up to 16 slices are supported.

MVME323 IOCTLS

The following `ioctl` commands are supported:

`M323FMTT`	format track; *arg* must be a pointer to a `struct m323ctl`
`M323GET`	get configuration; *arg* must be a pointer to a `struct config`
`M323SET`	set configuration; *arg* must be a pointer to a `struct config`
`M323RST`	restore drive
`M323CLRF`	clear fault
`M323VRFY`	verify track
`M323COFF`	cache off
`M323CON`	cache on
`M323MPT`	map alternate track; *arg* must be a pointer to a `struct m323ctl`
`M323MPS`	map track with sector slip
`M323RFMT`	reformat track, saving alternates
`RDMFRLIST`	read manufacturer's defect list from disk; *arg* must be a pointer to a `struct m323mlargs`

FILES

/usr/include/sys/m323.h
/usr/include/sys/m323drv.h
/dev/dsk/m323_*
/dev/rdsk/m323_*

ERRORS

The mvme323 driver generates many different error messages, which are displayed on the console to help the operator diagnose problems.

SEE ALSO

mvme323(1M) (For M68K only), intro(7)

NAME

mvme328 - MVME328 SCSI Host Adapter

DESCRIPTION

The MVME328 driver controls up to a total of 8 MVME328 SCSI host adapters. Each MVME328 SCSI host adapter can have one or two SCSI buses, with each SCSI bus supporting up to seven SCSI devices.

Assuming the necessary system resources are available, the MVME328 driver will send each command to the controller as soon as it receives the command from an application.

The MVME328 driver does not have to wait for a command to complete before sending a command for another device.

SUPPORT DEVICES

Disk Drives

Disk drives currently supported are:

DESCRIPTION	ddefs(1M) FILE	TYPE
150MB CDC 94161 Wren III	mcdcIII	Hard
300MB CDC 94171 Wren IV	mcdcIV	Hard
600MB CDC 94181 Wren V	mcdcV	Hard
1.2GB CDC 94601 Wren VII	mcdcVII	Hard
135MB FUJITSU M2613S	mfuj2613	Hard
180MB FUJITSU M2614S	mfuj2614	Hard
330MB FUJITSU M2622S	mfuj2622	Hard
525MB FUJITSU M2624S	mfuj2624	Hard
1.75GB FUJITSU M2652S	mfuj2652	Hard
Toshiba XM3201B CDROM	none	CDROM
1.2MB TEAC 5¼ inch FC-1	see next table	Floppy
2.88MB TEAC 3½ inch FC-1	see next table	Floppy

Note that in all tables, each entry in the ddefs(1M) FILE column is the name of a file that defines the characteristics of the disk in the /etc/dskdefs directory. Each entry in the BLOCKS column is the number of specified blocks when making a file system with mkfs(1M).

The types of floppy diskettes currently supported are listed in the following two tables.

5¼ INCH DISKETTES				
DESCRIPTION	ddefs(1M) FILE	BLOCKS	MEDIA TYPE	SLICE
Double density Motorola format	mdsdd5	1276	MFD-2DD	0
Single density PC/XT 8 sect./track	mpcxt8	640	MFD-2DD	12
Single density PC/XT 9 sect./track	mpcxt9	720	MFD-2DD	9
Double density PC/AT	mpcat	2400	MF2-HD	8

3½ INCH DISKETTES				
DESCRIPTION	ddefs(1M) FILE	BLOCKS	MEDIA TYPE	SLICE
Double density PC/XT 9 sect./track	mpcxt9_3	1440	MFD-2DD	13
Double density PS/2	mps2	2880	MF2-HD	10
Super High Density (2.88MB formatted)	mshd	5760	PMF2-ED	11

Tape Drives

Tape drives currently supported by the MVME328 host adapter are:

DESCRIPTION	FORMAT	TYPE
Archive 2150S	QIC24, QIC120, QIC150	Streaming
Archive 2525	QIC24, QIC120, QIC150	Streaming
Archive Python	DAT	Streaming
Exabyte EXB-8200	8mm	Streaming
Kennedy 9660	9-track	Start/Stop
M4 Data 9914	9-track	Start/Stop

MINOR NUMBERS

The MVME328 device driver interprets the minor number of a device using the standard SCSI-1 minor mapping.

DISK SUPPORT

During system initialization, the MVME328 device driver will spin-up any disks that are strapped to spin-up.

The hard disk drives supported by the MVME328 handle all defects internally. A list of known defective locations is recorded on the medium. During format, any data that would normally be loaded into these locations are automatically assigned alternate locations. Also during format, the drive is checked for defects in addition to those on the known list. If any additional defective locations are found, any data that would be stored there are assigned alternate locations.

The MVME328 device driver complies with the disk support standard specified on the `disk`(7) man page with the following exceptions:

`DKGETCFG` `ioctl` command
The MVME328 driver returns only the parameters that are relevant to the MVME328 driver and controller.

`DKGETINFO` `ioctl` command
The MVME328 driver returns only the parameters that are relevant to the MVME328 driver and controller.

`DKSETCFG` `ioctl` command
The MVME328 driver sets only the parameters that are relevant to the MVME328 driver and controller.

`DKSETINFO` `ioctl` command
The MVME328 driver sets only the parameters that are relevant to the MVME328 driver and controller.

`DKFORMAT` `ioctl` command
The SCSI FORMAT command is used to format the device. The argument *arg* is not used. Because the bad block strategy is perfect, no defect list is passed to the drive. By turning on a bit in the controller attribute word of the disk definition file passed to `dinit`, the drive can be told to ignore the grown defect list on the disk. Refer to the description of the controller attribute word on the `disk`(7) man page for more information.

TAPE SUPPORT

The MVME328 device driver complies with the tape support standard specified on the `tape`(7) man page with no exceptions.

FLOPPY DISK SUPPORT

The MVME328 supported floppy drives provide level one support as defined by the *88open PC Floppy Emulation Supplement* to the *Binary Compatibility Standard*.

The MVME328 device driver complies with the floppy disk support standard specified on the `floppy`(7) manual page with the following exceptions:

`DKFIXBADSPOT` `ioctl` command
This command always returns `EINVAL`.

`DKGETCFG` `ioctl` command
The MVME328 driver returns only the parameters that are relevant to the MVME328 driver and controller.

`DKGETINFO` `ioctl` command
The MVME328 driver returns only the parameters that are relevant to the MVME328 driver and controller.

`DKSETCFG` `ioctl` command
This command performs no operation; it returns with no effect and no error.

`DKSETINFO` `ioctl` command
This command performs no operation; it returns with no effect and no error.

`DKSETSLC` `ioctl` command
This command performs no operation; it returns with no effect and no error.

`FL_PC_LEVEL` `ioctl` command
The MVME328 driver currently only supports level 1, so the integer pointed to by *arg* is always set to 1 by this call.

Slicing
Floppy diskettes do not have volume ID blocks or Volume Table of Contents (VTOC). A floppy drive can be thought of as a hard disk with a single slice. The slice bits of the *minor number* select the drive geometry as described later in this manual page.

`V_PDREAD` `ioctl` command
This command always returns `EINVAL`.

`V_PDWRITE` `ioctl` command
This command always returns `EINVAL`.

`V_RVTOC` `ioctl` command
This command always returns `EINVAL`.

V_WVTOC ioctl command
This command always returns EINVAL.

dinit/ddef
The ddef files for floppy disks are treated as placeholders. Although they are required for dinit(1M) to work, the information is not used. The flormat of the diskette is determined via the slice number of the device. Please refer to the supported floppy tables at the beginning of this man page for more information.

Bad blocks may not be mapped out on a floppy disk. A bad block on a floppy disk make the entire floppy unacceptable.

CDROM SUPPORT

The MVME328 device driver will not spin-up CDROM devices at system initialization time.

The MVME328 device driver complies with the CDROM support standard specified on the cdrom(7) manual page with the following exceptions:

DKGETCFG ioctl command
The MVME328 driver returns only the parameters that are relevant to the MVME328 driver and controller.

DKGETINFO ioctl command
The MVME328 driver returns only the parameters that are relevant to the MVME328 driver and controller.

DKGETCFG ioctl command
The MVME328 driver returns only the parameters that are relevant to the MVME328 driver and controller.

PASSTHRU SUPPORT

The MVME328 device driver complies with the passthru support standard specified on the passthru(7) man page with no exceptions.

ERROR MESSAGES

The MVME328 device driver prints error messages to the system console. Many of these messages print a unit number to indicate which device was being accessed at the time of the error. The following table can help to interpret the unit number.

BRD	BUS	DEVICE	LUN	UNIT #	BRD	BUS	DEVICE	LUN	UNIT #
0	0	0	0-7	0-7	1	0	0	0-7	128-135
0	0	1	0-7	8-15	1	0	1	0-7	136-143
0	0	2	0-7	16-23	1	0	2	0-7	144-151
0	0	3	0-7	24-31	1	0	3	0-7	152-159
0	0	4	0-7	32-39	1	0	4	0-7	160-167
0	0	5	0-7	40-47	1	0	5	0-7	168-175
0	0	6	0-7	48-55	1	0	6	0-7	176-183
0	0	7	0-7	56-63	1	0	7	0-7	184-191
0	1	0	0-7	64-71	1	1	0	0-7	192-199
0	1	1	0-7	72-79	1	1	1	0-7	200-207
0	1	2	0-7	80-87	1	1	2	0-7	208-215
0	1	3	0-7	87-95	1	1	3	0-7	216-223
0	1	4	0-7	96-103	1	1	4	0-7	224-231
0	1	5	0-7	104-111	1	1	5	0-7	232-239
0	1	6	0-7	112-119	1	1	6	0-7	240-247
0	1	7	0-7	120-127	1	1	7	0-7	248-255

The MVME328 driver will print the following messages on the system console if an error occurs during system initialization:

`mvme328: Board failed powerup diagnostics`
There may be a problem with the MVME328 or its firmware.

`mvme328: Unable to Initialize Controller`
The MVME328 failed to initialize properly. Devices on the MVME328 are inaccessible.

`Unable to Start Queued Mode`
The MVME328 failed to initialize queued (or interrupt) mode of operation. Devices on the MVME328 are inaccessible.

`mvme328: cache inhibited SG pages not allocated`
The driver failed to allocate cache-inhibited memory for internal data structures. Devices on the MVME328 are inaccessible.

`mvme328: Unit` *unit_number* `not ready`
A device is present but not ready.

`mvme328: Unknown SCSI device type on unit` *unit_number*
Unit *unit_number* is an unrecognized SCSI device.

The MVME328 driver will print an error message of the following format to the system console whenever a disk device returns fatal error status:

`FATAL ERROR (`*mvme328_error_message*`) on mvme328 unit`
unit_number `blk` *blkno*
`mvme328: Unit=`*unit_number* `Cmd=`*cmd* `SCSI Cmd=`*scsi*
`Status=`*status*
`mvme328: Unit=`*unit_number* `sense key=`*key*
`(`*sense_msg*`)`

The MVME328 driver will print an error message of the following format to the system console whenever a tape device returns fatal error status:

FATAL ERROR (*mvme328_error_message*) on mvme328 tape unit *unit_number*
mvme328: Unit=*unit_number* Cmd=*cmd* SCSI Cmd=*scsi* Status=*status*
mvme328: Unit=*unit_number* sense key=*key* (*sense_msg*)

Fatal error status means that the drive was not able to complete the command successfully.

Recovered errors are printed in the same format, but begin with RECOVERED ERROR. Recovered error status means that the drive was able to complete the command successfully after some recovery action.

Two of the more useful values from these error messages are the SCSI command and the sense key. The following tables list some of the more common SCSI commands and sense keys.

SCSI COMMAND CODES	
Code	Description
0x00	Test Unit Ready
0x01	Rewind
0x04	Format Unit
0x06	Format Track
0x08	Read
0x0A	Write
0x10	Write Filemarks
0x11	Space
0x12	Inquiry
0x15	Mode Select
0x1A	Mode Sense
0x2F	Verify

SCSI SENSE KEYS	
Code	Description
0x0	Good Status
0x1	Recovered Error
0x2	Unit Not Ready
0x3	Media Error
0x4	Hardware Error
0x5	Illegal command
0x6	Unit Attention
0x7	Write-protected Media
0x8	Read Blank Media
0xE	Data Miscompare

Refer to the ANSI SCSI specification for a complete list of SCSI command codes and sense keys.

MASTER.D PARAMETERS

The following parameters affect the operation of the MVME328 device driver. The following are parameters listed under the MVME328 description:

`m328_max_spl`
: This parameter sets the maximum number of concurrent special commands. The default value is 8. Special commands are all SCSI commands except reads or writes. Most `ioctl()` commands are special commands, and special commands are used during `open()` and `close()` processing. If this number is too low, some processes will sleep waiting for resources when doing special commands.

`m328_max_raw_bufs`
: This parameter specifies the number of 64K byte buffers that will be allocated if any MVME328 host adapters have revision XAM firmware. The default number is 1. These buffers are used to work around a problem in the firmware that affect raw I/O. Tuning this parameter higher when XAM firmware is present will result in improved raw I/O performance, however, the tuning is no replacement for obtaining a firmware upgrade.

`m328_max_sglists`
: This parameter specifies the number of special scatter/gather lists that are available for use within the driver. It should be set to at least the number of processors plus 2; the default number is 8.

`m328_starve_size`
: This parameter specifies the maximum length of a disk, floppy, or CDROM I/O queue that will be sorted before beginning another queue.

`m328_vme_to`
: This parameter specifies the VMEBUS transfer time out in 32 millisecond ticks.

`m328_vme_cnt`
: This parameter specifies the VMEBUS burst transfer count. On systems with a large number of disks and/or MVME328 host adapters this number may have to be lowered to avoid DMA problems.

`m328_noisy_disk_open`
: This parameter controls the printing of error messages on the console when disk devices do not have valid Motorola identification in them. If this parameter is non-zero, messages will be printed; zero, no messages will be printed.

SPECIAL CONSIDERATIONS

When an error occurs while writing or reading a tape, the best course of action in this case is to rewind the tape and repeat the operation.

Removing a cartridge tape during an `MTBSF` operation hangs the tape drive.

An incorrect transfer count may be returned by the MVME328 device driver when using variable mode tape devices (e.g. 9-tracks, EXABYTE) in variable mode. This is due to a BUG in the XAM firmware and it is not found in any later firmware.

The problem shows itself when an odd length read is used to read a tape that contains even length records. The returned transfer count will be one less than it should be. The work-around is to read tapes with even length reads equal to or larger than the maximum size of the records found on the tape.

The longest I/O operation which MVME328 host adapters can allow to occur on a tape device operating in variable mode depends on two factors. If the MVME328 host adapter is using revision XAM firmware, the maximum length is 65535 bytes. For all other boards and firmware combinations, the maximum length will vary from a minimum of 252K bytes (worst case page alignment) to 256K bytes (page aligned). The actual maximum length may be either larger or smaller than the MVME328 host adapter may support. Refer to the device's documentation for more information.

FILES

/dev/dsk/m328_*
/dev/rdsk/m328_*
/dev/rmt/m328_*
/dev/generic/m328_*
/etc/dskdefs/m*
/usr/include/sys/dk.h
/usr/include/sys/mtio.h
/usr/include/sys/m328scsi.h
/usr/include/sys/m328sio.h
/usr/include/sys/m328space.h
/usr/include/sys/mvme328.h
/usr/include/sys/pcflio.h

SEE ALSO

mt(1), ddefs(1M), dinit(1M), close(2), ioctl(2), open(2), read(2), write(2), cdrom(7), disk(7), floppy(7), intro(7), mvme323(7) (For M68K only), mvme350(7) (For M68K only), tape(7) passthru(7)

NAME

mvme332xt - MVME332XT communication controller STREAMS driver

DESCRIPTION

mvme332xt is a STREAMS-based driver that provides a general interface to the MVME332XT VMEbus communication controller module. The MVME332XT controller supports up to eight asynchronous serial communication ports and one Centronics-compatible printer port. The mvme332xt driver supports up to eight MVME332XT controllers per system.

Each peripheral device connected to the MVME332XT has the same major device number. The MVME332XT firmware presents a generic serial and printer device interface to the driver, which distinguishes a serial device from the printer device by its device unit number. Device numbers 0-7 are allocated for the eight serial devices, and the printer is designated unit 8. The least significant 4 bits in the minor device field are interpreted as the device unit number. Therefore, 16 minor device numbers are required per MVME332XT controller. The next highest four bits of the minor device number are interpreted as the controller number.

When the mvme332xt driver is used with the STREAMS line discipline module - ldterm(7), behavior on all communications ports is as described in UNIX System V/68 or V/88 Release 4 termio(7).

MVME332XT IOCTLS

In addition to supporting the standard ioctl(2) commands as specified by termio(7), the mvme332xt supports hardware flow control and downloading of object code and data to the MVME332XT.

The following MVME332XT-specific ioctl system calls have the form:

```
ioctl(fildes, command, arg)
int fildes, command;
struct dl_info *arg;
```

The dl_info structure is defined in /usr/include/sys/mvme332xt.h and has the following format:

```
struct   dl_info {
         unsigned long    hostaddr;    /* host (user) address */
         unsigned long    ipcaddr;
         unsigned long    count;       /* to be transferred */
         unsigned long    wrk0;
         unsigned short   wrk1;
};
```

TCGETDL

Get download information from the MVME332XT. *arg* is a pointer to a user buffer large enough to contain a dl_info structure. The base address of the downloadable area is returned in the ipcaddr field of this structure, and the size in bytes of the downloadable area is returned in the count field.

TCDLOAD

Download object code or data to the MVME332XT. *arg* is a pointer to a user buffer containing a dl_info structure. The hostaddr field points to a user buffer containing the object code or data to be downloaded. The ipcaddr field points to the base address of the downloadable area in MVME332XT local RAM. The `count` field specifies the number of bytes to be downloaded.

`TCGETSYM`
Get symbol table from the MVME332XT. *arg* is a pointer to a user buffer containing a `dl_info` structure. The `hostaddr` field points to a user buffer into which the symbol information will be copied. The size of this buffer in bytes is specified by the `count` field. The `ipcaddr` field should be set to `0` for the first call to `TCGETSYM` to indicate the beginning of the symbol table. It is updated by the MVME332XT for subsequent `TCGETSYM` commands. At the end of the symbol table, the MVME332XT returns `EOF` in the `ipcaddr` field. On completion, the `count` field specifies the number of bytes returned by the MVME332XT.

`TCWHAT`
This command performs exactly the same function as the `TCGETSYM` command, except that it returns a list of the firmware files with SCCS version numbers. *arg* is a pointer to a user buffer containing a `dl_info` structure. The `hostaddr` field points to a user buffer into which the SCCS information will be copied. The size of this buffer in bytes is specified by the `count` field. The `ipcaddr` field should be set to `0` for the first call to indicate the start of the `TCWHAT` command. It is updated by the MVME332XT for subsequent `TCWHAT` commands. At the end of the SCCS information, the MVME332XT returns `EOF` in the `ipcaddr` field. On completion, the `count` field specifies the number of bytes returned by the MVME332XT.

`TCLINE`
Load line discipline table, previously downloaded by `TCDLOAD`, into the MVME332XT's internal table. *arg* points to a user buffer containing a `dl_info` structure. The `ipcaddr` field points to a user buffer containing the `linesw` table. The `count` field specifies the number of lines in the `linesw` table. The MVME332XT `linesw` table is defined as follows:

```
struct   linesw
{
         int      (*l_open)();
         int      (*l_read)();
         int      (*l_write)();
         int      (*l_close)();
         int      (*l_ctl)();
         int      (*l_gate)();
};
```

`TCEXEC`
Execute a user function that has been downloaded by a previous `TCDLOAD` command. *arg* points to a user buffer containing a `dl_info` structure. The `ipcaddr` field specifies the execution function address.

The following MVME332XT-specific `ioctl` system call has the form:

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

TCSETHW
: Set hardware flow control option. If *arg* is `1`, enable hardware flow control using the RTS/CTS signal pairs; if *arg* is `0`, disable hardware flow control.

The following MVME332XT-specific *ioctl* system calls have the form:

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

TCGETHW
: Return hardware flow control status. If the specified serial port has hardware flow control enabled, `1` is returned to the *arg* integer location; otherwise, `0` is returned.

TCGETVR
: Return MVME332XT firmware and driver version and revision numbers in the integer pointed to by *arg*. The driver version number is returned in the most significant byte, the driver revision number is in the second most significant byte, the firmware revision number is in the third byte, and the firmware revision number is in the least significant byte.

TCGETDS
: Return the current status of a device's hardware signals, such as DCD, CTS, DSR, PR_FAULT, PR_POUT and PR_SELECT, in the integer pointed to by *arg*. The following status values are defined in `/usr/include/sys/mvme332xt.h`:

E_DCD, E_LOST_CDC
E_DSR, E_LOST_DSR
E_CTS, E_LOST_CTS
E_PR_FAULT, E_PR_POUT, E_PR_SELECT

The following MVME332XT-specific `ioctl` system calls have the form:

```
ioctl(fildes, command, arg)
int fildes, command;
struct termios *arg;
```

TCSETDF
: Set the default `termios` parameters. *arg* is a pointer to a user-supplied `termios` structure.

TCGETDF
: Get the default `termios` parameters. *arg* is a pointer to a user buffer large enough to contain a `termios` structure.

CONFIGURATION ISSUES

Currently, the MVME332XT operates in a canonical state which handles only the most basic of features (breaks and interrupts). Remaining functionality is left to the `ldterm`(7) module. The `ldterm`(7) module may be pushed on the STREAM via the `autopush`(1M) or when beginning a `ttymon`(1M) directly from the `/etc/inittab` file. [See init(1M)].

FILES

/usr/include/sys/mvme332xt.h
/dev/term/??, /dev/printer/lp?, /dev/port/m332_c?d?

ERRORS

The mvme332xt driver generates many different error messages, which are displayed on the console in order to help the operator to diagnose problems. The error messages displayed have the following format:

MVME332xt: controller *X*, unit *Y* - *MESSAGE*

where *X* is the controller number, *Y* is the unit number, and *MESSAGE* is one of the following:

Create channel error - disabled
: The driver must establish a communication channel with the MVME332XT before any commands can be dispatched. This error indicates that the channel between the driver and the MVME332XT was not successfully created, and typically indicates a configuration problem or malfunction. The controller is marked as bad by the driver and further access attempts are disallowed.

Initialization error, disabled
: An error was reported by the MVME332XT controller when the driver sent an initialization command to it. This condition will result if the driver attempts to size one of the MVME332XT read/write rings to a non-base-2 value.

Unknown interrupt
: An interrupt occurred from a MVME332XT controller that was marked nonexistent or bad.

Corrupt envelopes - disabled
: This indicates channel corruption in the MVME332XT shared RAM.

PRINTER is de-selected
: This message indicates that the printer is de-selected. Check the printer select switch.

PRINTER is out of paper
: This indicates that the printer is out of paper. Check the printer paper supply.

PRINTER fault for unknown reason
: This indicates a printer error other than the paper out or the de-selected error conditions. Check the printer connections or refer to the printer manufacturer's user manual.

SEE ALSO

autopush(1M), mvme332xt(1M), ttymon(1M), termio(7), ldterm(7).

NAME

mvme350 - MVME350 cartridge tape controller (For M68K only)

DESCRIPTION

mvme350 is a driver that provides a general interface to the MVME350 VMEbus tape controller module. The MVME350 controller supports one cartridge tape. The mvme350 driver supports up to eight MVME350 controllers per system.

Each tape connected to the MVME350 has the same major device number.

MVME350 IOCTLS

The following ioctl commands are supported:

M350REWIND	rewind tape
M350ERASE	erase tape
M350RETENSION	retension tape
M350WRTFM	write filemark
M350RDFM	read filemark
M350SETDMA	set DMA buffer size
M350GETDMA	get DMA buffer size
M350BYTESWAP	set/reset byteswapping

FILES

/usr/include/sys/mvme350.h
/dev/rmt/m350_*

ERRORS

The mvme350 driver generates many different error messages, which are displayed on the console to help the operator diagnose problems.

SEE ALSO

mvme350(1M) (For M68K only), intro(7)

NAME

netconfig - network configuration database

SYNOPSIS

```
#include <netconfig.h>
```

DESCRIPTION

The network configuration database, /etc/netconfig, is a system file used to store information about networks connected to the system and available for use. The netconfig database and the routines that access it [see getnetconfig(3N)] are part of the UNIX System V Network Selection component. The Network Selection component also includes the environment variable NETPATH and a group of routines that access the netconfig database using NETPATH components as links to the netconfig entries. NETPATH is described in sh(1); the NETPATH access routines are discussed in getnetpath(3N).

netconfig contains an entry for each network available on the system. Entries are separated by newlines. Fields are separated by whitespace and occur in the order in which they are described below. Whitespace can be embedded as "*blank*" or "*tab*." Backslashes may be embedded as "\\\\". Each field corresponds to an element in the struct netconfig structure. struct netconfig and the identifiers described on this manual page are defined in /usr/include/netconfig.h.

network ID

A string used to uniquely identify a network. *network ID* consists of non-null characters, and has a length of at least 1. No maximum length is specified. This namespace is locally significant and the local system administrator is the naming authority. All *network ID*s on a system must be unique.

semantics

The *semantics* field is a string identifying the "semantics" of the network, that is, the set of services it supports, by identifying the service interface it provides. The *semantics* field is mandatory. The following semantics are recognized.

tpi_clts	Transport Provider Interface, connectionless
tpi_cots	Transport Provider Interface, connection oriented
tpi_cots_ord	Transport Provider Interface, connection oriented, supports orderly release.
tpi_raw	Transport Provider Interface, raw

flag The *flag* field records certain two-valued ("true" and "false") attributes of networks. *flag* is a string composed of a combination of characters, each of which indicates the value of the corresponding attribute. If the character is present, the attribute is "true." If the character is absent, the attribute is "false." "-" indicates that none of the attributes is present. Only one character is currently recognized:

v Visible ("default") network. Used when the environment variable NETPATH is unset.

`b` Enable RPC broadcast.

protocol family

The *protocol family* and *protocol name* fields are provided for protocol-specific applications.

The *protocol family* field contains a string that identifies a protocol family. The *protocol family* identifier follows the same rules as those for *network IDs*, that is, the string consists of non-null characters; it has a length of at least `1`; and there is no maximum length specified. A "`-`" in the *protocol family* field indicates that no protocol family identifier applies, that is, the network is experimental. The following are examples:

`loopback`	Loopback (local to host).
`inet`	Internetwork: UDP, TCP, and so on
`implink`	ARPANET imp addresses
`pup`	PUP protocols: for example, BSP
`chaos`	MIT CHAOS protocols
`ns`	XEROX NS protocols
`nbs`	NBS protocols
`ecma`	European Computer Manufacturers Association
`datakit`	DATAKIT protocols
`ccitt`	CCITT protocols, X.25, and so on
`sna`	IBM SNA
`decnet`	DECNET
`dli`	Direct data link interface
`lat`	LAT
`hylink`	NSC Hyperchannel
`appletalk`	Apple Talk
`nit`	Network Interface Tap
`ieee802`	IEEE 802.2; also ISO 8802
`osi`	Umbrella for all families used by OSI (for example, protosw lookup)
`x25`	CCITT X.25 in particular
`osinet`	AFI = 47, IDI = 4
`gosip`	U.S. Government OSI

protocol name

The *protocol name* field contains a string that identifies a protocol. The *protocol name* identifier follows the same rules as those for *network IDs*, that is, the string consists of non-NULL characters; it has a length of at least `1`; and there is no maximum length specified. The following protocol names are recognized. A "`-`" indicates that none of the names listed applies.

`tcp` Transmission Control Protocol

`udp` User Datagram Protocol

`icmp` Internet Control Message Protocol

network device

The *network device* is the full pathname of the device used to connect to the transport provider. Typically, this device will be in the `/dev` directory. The *network device* must be specified.

directory lookup libraries

The *directory lookup libraries* support a "directory service" (a name-to-address mapping service) for the network. This service is implemented by the UNIX System V Name-to-Address Mapping feature. If a network is not provided with such a library, the *netdir* feature will not work. A "-" in this field indicates the absence of any lookup libraries, in which case name-to-address mapping for the network is non-functional. The directory lookup library field consists of a comma-separated list of full pathnames to dynamically linked libraries. Commas may be embedded as "\,"; backslashs as "\\".

Lines in /etc/netconfig that begin with a sharp sign (#) in column 1 are treated as comments.

The struct netconfig structure includes the following members corresponding to the fields in in the netconfig database entries:

char * nc_netid	Network ID, including NULL terminator
unsigned long nc_semantics	Semantics
unsigned long nc_flag	Flags
char * nc_protofmly	Protocol family
char * nc_proto	Protocol name
char * nc_device	Full pathname of the network device
unsigned long nc_nlookups	Number of directory lookup libraries
char ** nc_lookups	Full pathnames of the directory lookup libraries themselves
unsigned long nc_unused[9]	Reserved for future expansion (not advertised to user level)

The nc_semantics field takes the following values, corresponding to the semantics identified above:

NC_TPI_CLTS
NC_TPI_COTS
NC_TPI_COTS_ORD
NC_TPI_RAW

The nc_flag field is a bitfield. The following bit, corresponding to the attribute identified above, is currently recognized. NC_NOFLAG indicates the absence of any attributes.

NC_VISIBLE

FILES

/etc/netconfig
/usr/include/netconfig.h

SEE ALSO

getnetconfig(3N), getnetpath(3N), icmp(7), ip(7), netconfig(4), netdir_getbyname() [see netdir(3N)]

NAME

netmasks - network mask data base

DESCRIPTION

The netmasks file contains network masks used to implement IP standard subnetting. For each network that is subnetted, a single line should exist in this file with the network number, any number of SPACE or TAB characters, and the network mask to use on that network. Network numbers and masks may be specified in the conventional IP '.' notation (like IP host addresses, but with zeroes for the host part). For example,

```
128.32.0.0 255.255.255.0
```

can be used to specify that the Class B network 128.32.0.0 should have eight bits of subnet field and eight bits of host field, in addition to the standard sixteen bits in the network field.

FILES

/etc/netmasks

SEE ALSO

ifconfig(1M)

Postel, Jon, and Mogul, Jeff, *Internet Standard Subnetting Procedure*, RFC 950, Network Information Center, SRI International, Menlo Park, Calif., August 1985

NAME

netrc - file for ftp remote login data

DESCRIPTION

The .netrc file contains data for logging in to a remote host over the network for file transfers by ftp(1). This file resides in the user's home directory on the machine initiating the file transfer. Its permissions should be set to disallow read access by group and others [see chmod(1)].

The following tokens are recognized; they may be separated by SPACE, TAB, or NEWLINE characters:

machine *name*
: Identify a remote machine name. The auto-login process searches the .netrc file for a machine token that matches the remote machine specified on the ftp command line or as an open command argument. Once a match is made, the subsequent .netrc tokens are processed, stopping when the EOF is reached or another machine token is encountered.

login *name*
: Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

password *string*
: Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note: if this token is present in the .netrc file, ftp will abort the auto-login process if the .netrc is readable by anyone besides the user.

account *string*
: Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an ACCT command if it does not.

macdef *name*
: Define a macro. This token functions as the ftp macdef command functions. A macro is defined with the specified name; its contents begin with the next .netrc line and continue until a NULL line (consecutive NEWLINE characters) is encountered. If a macro named init is defined, it is automatically executed as the last step in the auto-login process.

EXAMPLE

A .netrc file containing the following line:

```
machine ray login demo password mypassword
```

allows an autologin to the machine ray using the login name demo with password mypassword.

FILES

~/.netrc

SEE ALSO

chmod(1), ftp(1), ftpd(1M)

NAME

networks - network name data base

DESCRIPTION

The networks file contains information regarding the known networks which comprise the DARPA Internet. For each network a single line should be present with the following information:

official-network-name network-number aliases

Items are separated by any number of SPACE and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network number may be specified in the conventional '.' notation using the inet_network routine from the Internet address manipulation library, inet(7). Network names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

FILES

/etc/networks

SEE ALSO

getnetent(3N), inet(7)

NOTES

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

`null` - the null file

DESCRIPTION

Data written on the null special file, `/dev/null`, is discarded.

Reads from a null special file always return 0 bytes.

FILES

`/dev/null`

NAME

nvram - general non-volatile RAM driver for SYSTEM V

DESCRIPTION

The nvram driver provides an interface from SYSTEM V to the non-volatile RAM device and to character devices.

The non-volatile RAM is a collection of eight slices. Each slice is associated with a minor device number and a size. The nvram slice sizes are static and cannot be changed by the user. The following tables show the 2 KB and 8 KB slice configurations for nvram.

2 KB Slice Configuration for SYSTEM V/68			
Minor Device (slice) Number	Functionality	Size (in bytes)	Device Name
0	available to user	1024	/dev/nvr/user
1	networking	64	/dev/nvr/net
2	unused	0	
3	operating system	440	/dev/nvr/os
4	unused	0	
5	BUG	512	/dev/nvr/bug
6	unused	0	
7	total nvram	2040	/dev/nvr/nvr

2 KB Slice Configuration for SYSTEM V/88			
Minor Device (slice) Number	Functionality	Size (in bytes)	Device Name
0	available to user	1024	/dev/nvr/user
1	networking	64	/dev/nvr/net
2	unused	0	
3	operating system	440	/dev/nvr/os
4	unused	0	
5	BUG	512	/dev/nvr/bug
6	CONFIG	256	/dev/nvr/config
7	total nvram	2040	/dev/nvr/nvr

8 KB Slice Configuration for SYSTEM V/68 and V/88			
Minor Device (slice) Number	Functionality	Size (in bytes)	Device Name
0	available to user	4096	/dev/nvr/user
1	networking	256	/dev/nvr/net
2	unused	0	
3	operating system	1528	/dev/nvr/os
4	unused	0	
5	BUG	2048	/dev/nvr/bug
6	CONFIG	256	/dev/nvr/config
7	total nvram	8184	/dev/nvr/nvr

Superuser privileges are required to write nvram slices having a minor device number greater than 0. Read access on slices 1 through 7 (inclusive) and read/write access on slice 0 are defined by the file permissions on the associated device file.

NVRAM BASE ADDRESS

MVME187 - 0xfffc0000
MVME167 - 0xfffc0000
MVME188 - 0xfff80000
MVME197 - 0xfffc0000

ERRORS

If failure occurs, the NVRAM driver generates the following error messages:

ENXIO invalid device minor number

EPERM invalid access permission

EFAULT data transfer failed or an illegal accesss to memory occurred

EINVAL boundary violation

FILES

/dev/nvr/bug
/dev/nvr/config
/dev/nvr/net
/dev/nvr/nvr
/dev/nvr/os
/dev/nvr/user

SEE ALSO

close(2), lseek(2), open(2), read(2), and write(2)

NAME

.ott - FACE object architecture information

DESCRIPTION

The FACE object architecture stores information about object-types in an ASCII file named .ott (object type table) that is contained in each directory. This file describes all of the objects in that directory. Each line of the .ott file contains information about one object in pipe-separated fields. The fields are (in order):

name — the name of the actual UNIX System file.

dname — the name that should be displayed to the user, or a dot if it is the same as the name of the file.

description — the description of the object, or a dot if the description is the default (the same as object-type).

object-type — the FACE internal object type name.

flags — object specific flags.

mod time — the time that FACE last modified the object. The time is given as number of seconds since 1/1/1970, and is in hexadecimal notation.

object information — an optional field, contains a set of semi-colon separated *name=value* fields that can be used by FACE to store any other information necessary to describe this object.

FILES

.ott is created in any directory opened by FACE.

NAME

`passthru` - passthru support

DESCRIPTION

All Motorola SCSI controllers provide passthru support via the DKPASSTHRU ioctl command. This function permits any scsi command specified by a device manufacturer to be passed directly to the device for processing. This command requires superuser permissions.

IOCTL COMMANDS

All DKPASSTHRU `ioctl`(2) operations take the form `ioctl` (*fildes, DKPASSTHRU, *arg)*, where *arg is a pointer to a scsi_pass structure. The scsi_pass structure is defined in <sys/dk.h>.

You must set up the scsi_pass structure before issuing this ioctl. The following is a list of the fields in the scsi_pass structure and their functions:

flags

This field contains the size of the SCSI command descriptor block (CDB) in bits 4-7 (bit four is the low order bit). These bits are defined by the mask SPT_CDB_LEN in <sys/dk.h>. The only valid values for this sub-field are 6, 10, and 12. If this sub-field contains any other value, the ioctl fails, returning -1 and setting errno to ERANGE. Another bit is defined by the SPT_READ mask defined in <sys/dk.h>. This bit must be set if the direction of data transfer for this CDB is from the device to the host system. If this bit is set incorrectly, the ioctl fails, returning -1 and setting errno to EIO and error_info to SPTERR_CTLR (controller error).

Only one other bit is currently defined. This last bit is defined by the SPT_LONG_TIMEOUT mask defined in <sys/dk.h>. If this bit is set, it tells the driver that the SCSI command takes a long time (e.g., FORMAT UNIT), and so the command timeout should be long enough to compensate. The MVME328 device driver currently does not specify a timeout for commands (the timeout is infinite), so it ignores this bit.

All other flags bits are currently reserved and should be zero. If any reserved bit is set, the ioctl fails, returning EINVAL.

xfer_len

This field contains the number of bytes that are to be transferred to or from the device. The direction of transfer is determined by the SPT_READ bit in the flags field. If this field is zero, no data transfer is attempted. Note that the setting of this field depends on the SCSI command. The xfer_len count must be an even number. If the transfer length in the CDB is an odd number, xfer_len must be rounded up to be even. The buffer must of course be large enough to allow this adjustment. If the transfer count is odd, the ioctl fails, returning -1 and setting errno to EINVAL.

Note that this restriction only applies to MVME328 thru-hole boards. If the firmware revision number for the MVME328 is XAM, the residual is -1 as a result of this adjustment. Later revisions of the firmware have the correct residual count (with respect to the transfer count in the CDB). Surface mount versions of the MVME328 will not have this restriction. Also note that for

MVME328 controllers with firmware revision XAM, the transfer length is limited to MACSI_SG_RSIZE (65535) bytes. This is the size of the buffers in the MVME328 driver because this is the maximum value for each scatter/gather register.

If xfer_len is zero when the CDB is set up to transfer data, the ioctl fails, returning -1 and setting errno to EIO and error_info to SPTERR_CTLR (controller error). If xfer_len is not equal to the number of bytes the SCSI command defined by the CDB transfers, it could cause a SCSI bus hang.

data

This field points to a buffer of size xfer_len in the caller's address space. The buffer must be page-aligned (use the NBPP define in <sys/immu.h>). If the SPT_READ bit in the flags field is clear (0), the buffer contains data to be sent as part of the command (for example, a defect list sent as part of a FORMAT UNIT command). If this bit is set (1), it indicates that the buffer will receive the data returned from the device as a result of executing the command.

resid

This field points to an integer in the caller's address space. This integer is set to the number of bytes that were not transferred as a result of the SCSI command. This is the difference between the value of xfer_len and the number of bytes that were successfully transferred to or from the device. If this integer is set to zero after a command completes, xfer_len bytes were successfully transferred. If it is equal to xfer_len, no bytes were successfully transferred. This field may not be valid if errno is EFAULT.

If this field contains a bad pointer (e.g., NULL), the ioctl fails when it attempts to set this field, returning -1 and setting errno to EFAULT.

sense_data

This field is a pointer to a structure of type struct ext_sense in the caller's address space that is used to accept SCSI sense data in the event of a SCSI error. This structure is defined in <sys/dk.h>. If there is a SCSI error while executing the command, and the status is 0x02 (SCSI Check Condition), the error_info field is set to SPTERR_SCSI and the sense data is copied to this buffer. Note that this buffer is only modified by this command in the event of a SCSI error with Check Condition status.. Therefore, the caller should clear this buffer before executing this ioctl.

If this field contains a bad pointer (e.g., NULL) and there is a SCSI Check Condition status while executing the command, the ioctl fails, returning -1 and setting errno to EFAULT.

cdb

This field is an array of 12 bytes that contains the SCSI CDB that is to be passed to the device. Only the number of bytes specified in bits 4-7 of the flags field are actually copied out of this array into the IOPB that is passed to the device. The device driver does no checking of the contents of the CDB. It simply passes it to the device.

status

This field is a pointer to a byte that is used to accept the SCSI status byte in the case of a SCSI error. This field is valid in the case of a SCSI or controller error (when error_info is set to SPTERR_SCSI or SPTERR_CTLR), and it is not updated if the command completes successfully or with a driver error.

If this field contains a bad pointer (e.g., NULL) and there is a SCSI error or a controller error while executing the command, the ioctl fails when it tries to copy the status out to the user's address space, returning -1 and setting errno to EFAULT. Note that some status values are never returned by the MVME328 device driver because they are handled by the controller (e.g., busy). See the ANSI SCSI specification for a list of status codes.

error_info

This field is a pointer to an unsigned integer in the caller's address space that is used to indicate the resulting status of the ioctl. This field is always updated, even if no error occurs. If the command fails (ioctl returns -1), this field indicates what type of error occured. The valid values for this field are defined in <sys/dk.h>.

If the unsigned integer pointed to by error_info is set to SPTERR_DRIVER, it means that an error occured while setting up the command before sending it to the device. In this case, errno should be examined to determine the cause of the failure. Note that some values of errno can be caused by one of several different error conditions (EINVAL and EFAULT, for instance). If it is set to SPTERR_SCSI or SPTERR_CTLR, then the byte pointed to by the status field contains the SCSI status byte. If this status byte is set to 0x02 (SCSI Check Condition), the sense data for the device is copied into the buffer pointed to by the sense_data field. For this case, errno is set to EIO.

If this field contains a bad pointer (e.g., NULL), the ioctl fails, returning -1 and setting errno to EFAULT.

ctlr_code

This field is used to return the controller-specific error code in the case of a SCSI or driver error. This field is only modified if an error occurs and the error_info field is either SPTERR_SCSI or SPTERR_CTLR.

If this field contains a bad pointer (e.g., NULL), the ioctl fails, returning -1 and setting errno to EFAULT. See the MVME328 SCSI Host Adapter User's Guide for a list of valid controller codes.

ERRORS

If the passthru command can access too may memory regions the command will be terminated and ENXIO will be returned.

If the passthru command fails for any other reason the error results returned by the driver will be returned to the calling program.

FILES

`/usr/include/sys/dk.h`

SEE ALSO

ioctl(2), mvme328(7), scsi1x7(7)

NAME

passwd - password file

SYNOPSIS

/etc/passwd

DESCRIPTION

/etc/passwd is an ASCII file that contains basic information about each user's account. This file contains a one-line entry for each authorized user, of the form:

username : password : uid : gid : comment : home-dir : login-shell

where:

username is the user's login name. This field contains no uppercase characters, and must not be more than eight characters in length.

password contains the character x. This field remains only for compatibility reasons. Password information is contained in the file /etc/shadow; see shadow(4). If this field is empty, login(1) does not request a password before logging the user in.

uid is the user's numerical ID for the system, which must be unique. *uid* is generally a value between 0 and 32767.

gid is the numerical ID of the group that the user belongs to. *gid* is generally a value between 0 an 32767.

comment is the user's real name, along with information to pass along in a mail-message heading. An ampersand (&) in this field stands for the login name (in cases where the login name appears in a user's real name).

home-dir is the pathname to the directory in which the user is initially positioned upon logging in.

login-shell is the user's initial shell program. If this field is empty, the default shell is /usr/bin/sh.

Fields are separated by a colon, and each user from the next by a NEWLINE. Comment lines (lines preceded by the pound character (#) are not allowed in the /etc/passwd file. passwd also contains information used by the NIS package. These options are available only if the NIS package is installed.

/etc/passwd has general read permission on all systems, and can be used by routines that map numerical user IDs to names. The passwd file can also have lines beginning with a plus sign (+) which means to incorporate entries from the Network Information Service (NIS). There are three styles of + entries in this file: by itself, + means to insert the entire contents of the NIS password file at that point; + *name* means to insert the entry (if any) for *name* from the NIS service at that point; +@ *netgroup* means to insert the entries for all members of the network group netgroup at that point. If a + *name* entry has a non-NULL *password, comment, home-dir*, or *login-shell* field, the value of that field overrides what is contained in the NIS service. The *uid* and *gid* fields cannot be overridden.

The passwd file can also have lines beginning with a minus sign (-) which means to disallow entries from the NIS service. There are two styles of - entries in this file: - *name* means to disallow any subsequent entries (if any) for *name* (in this file or in the NIS service); -@ *netgroup* means to disallow any subsequent entries for all members of the network group *netgroup*.

EXAMPLES

Here is a sample passwd file:

```
root:x:0:10:God:/:/bin/csh
fred:x:508:10:& Fredericks:/usr2/fred:/bin/csh
+john:
+@documentation:no-login:
+::::Guest
```

In this example, there are specific entries for users root and fred, to assure that they can log in even when the system is running standalone. The user john will have his password entry in the NIS service incorporated without change; anyone in the netgroup documentation will have their password field disabled, and anyone else will be able to log in with their usual password, shell, and home directory, but with a comment field of Guest.

FILES

/etc/passwd
/etc/shadow

SEE ALSO

login(1), passwd(1), pwconv(1M), useradd(1M), usermod(1M), userdel(1M), a641(3C), getpwent(3C), putpwent(3C), shadow(4), group(4), and unistd(4).

NAME

pathalias - alias file for FACE

DESCRIPTION

The pathalias files contain lines of the form alias=*path* where *path* can be one or more colon-separated directories. Whenever a FACE user references a path not beginning with a "/", this file is checked. If the first component of the pathname matches the left-hand side of the equals sign, the right-hand side is searched much like $PATH variable in the UNIX System. This allows users to reference the folder $HOME/FILECABINET by typing filecabinet.

There is a system-wide pathalias file called $VMSYS/pathalias, and each user can also have local alias file called $HOME/pref/pathalias. Settings in the user alias file override settings in the system-wide file. The system-wide file is shipped with several standard FACE aliases, such as filecabinet, wastebasket, preferences, other_users, and so on.

NOTES

Unlike command keywords, partial matching of a path alias is not permitted, however, path aliases are case insensitive. The name of an alias should be alphabetic, and in no case can it contain special characters like "/", "\", or "=". There is no particular limit on the number of aliases allowed. Alias files are read once, at login, and are held in core until logout. Thus, if an alias file is modified during a session, the change will not take effect until the next session.

FILES

$HOME/pref/pathalias
$VMSYS/pathalias

NAME

pckt - STREAMS Packet Mode module

DESCRIPTION

pckt is a STREAMS module that may be used with a pseudo terminal to packetize certain messages. The pckt module should be pushed [see I_PUSH, streamio(7)] onto the master side of a pseudo terminal.

Packetizing is performed by prefixing a message with an M_PROTO message. The original message type is stored in the 4 byte data portion of the M_PROTO message.

On the read-side, only the M_PROTO, M_PCPROTO, M_STOP, M_START, M_STOPI, M_STARTI, M_IOCTL, M_DATA, M_FLUSH, and M_READ messages are packetized. All other message types are passed upstream unmodified.

Since all unread state information is held in the master's stream head read queue, flushing of this queue is disabled.

On the write-side, all messages are sent down unmodified.

With this module in place, all reads from the master side of the pseudo terminal should be performed with the getmsg(2) or getpmsg() system call. The control part of the message contains the message type. The data part contains the actual data associated with that message type. The onus is on the application to separate the data into its component parts.

SEE ALSO

crash(1M), getmsg(2), ioctl(2), ldterm(7), ptem(7), streamio(7), termio(7).

NAME

pkginfo - package characteristics file

DESCRIPTION

pkginfo is an ASCII file that describes the characteristics of the package along with information that helps control the flow of installation. It is created by the software package developer.

Each entry in the pkginfo file is a line that establishes the value of a parameter in the following form:

PARAM="*value*"

There is no required order in which the parameters must be specified within the file. Each parameter is described below. Only fields marked with an asterisk are mandatory.

PKG*

PKG is the parameter to which you assign an abbreviation for the name of the package being installed. The abbreviation must be a short string (no more than nine characters long) and it must conform to file naming rules. All characters in the abbreviation must be alphanumeric and the first may not be numeric. install, new, and all are reserved abbreviations.

The package name you assign to PKG is also used in the instance name (*pkginst*) for the package in question. *pkginst* is composed of one or two parts: *pkg* (the same string you assigned to PKG) and, if more than one instance of that package exists, *pkg* plus *inst* (an instance identifier). (The term "package instance" is used loosely: it refers to all instantiations of *pkginst*, even those that do not include instance identifiers.)

The package name abbreviation (*pkg*) is the mandatory part of *pkginst*. To create such an abbreviation, assign it with the PKG parameter. For example, to assign the abbreviation sds to the Software Distribution Service package, enter PKG=sds.

The second part (*inst*), which is required only if you have more than one instance of the package in question, is a suffix that identifies the instance. This suffix is either a number (preceded by a period) or any short mnemonic string you choose. If you don't assign your own instance identifier when one is required, the system assigns a numeric one by default. For example, if you have three instances of the Software Distribution Service package and you don't create your own mnemonic identifiers (such as old and beta), the system adds the suffixes .2 and .3 to the second and third packages, automatically.

To indicate all instances of a package, specify inst.*. (When using this format, enclose the command line in single quotes to prevent the shell from interpreting the * character.) Use the token all to refer to all packages available on the source medium.

NAME*	Text that specifies the package name.
ARCH	A comma-separated list of alphanumeric tokens that indicate the architecture (for example, ARCH=m68k,m88k) associated with the package. The pkgmk tool may be used to create or modify this value when actually building the package. The maximum length of a token is 16 characters and it cannot include a comma. ARCH is not a mandatory field. Therefore, if it is not specified or if it is specified as NULL, it is ignored.
VERSION*	Text that specifies the current version associated with the software package. The maximum length is 256 ASCII characters and the first character cannot be a left parenthesis. The pkgmk tool may be used to create or modify this value when actually building the package.
CATEGORY*	A comma-separated list of categories under which a package may be displayed. A package must at least belong to the system or application category. Categories are case-insensitive and may contain only alphanumerics. Each category is limited in length to 16 characters.
DESC	Text that describes the package.
VENDOR	Used to identify the vendor that holds the software copyright (maximum length of 256 ASCII characters).
HOTLINE	Phone number and/or mailing address where further information may be received or bugs may be reported (maximum length of 256 ASCII characters).
EMAIL	An electronic address where further information is available or bugs may be reported (maximum length of 256 ASCII characters).
VSTOCK	The vendor stock number, if any, that identifies this product (maximum length of 256 ASCII characters).
CLASSES	A space-separated list of classes defined for a package. The order of the list determines the order in which the classes are installed. Classes listed first will be installed first (on a media by media basis). This parameter may be modified by the request script.
ISTATES	A list of allowable run states for package installation (for example, "S s 1").
RSTATES	A list of allowable run states for package removal (for example, "S s 1").
BASEDIR	The pathname to a default directory where "relocatable" files may be installed. If blank, the package is not relocatable and any files that have relative pathnames will not be installed. An administrator can override the default directory.
ULIMIT	If set, this parameter is passed as an argument to the ulimit command, which establishes the maximum size of a file during installation.

ORDER
: A list of classes defining the order in which they should be put on the medium. Used by pkgmk in creating the package. Classes not defined in this field are placed on the medium using the standard ordering procedures.

MAXINST
: The maximum number of package instances that should be allowed on a machine at the same time. By default, only one instance of a package is allowed. This parameter must be set in order to have multiple instances of a package.

PSTAMP
: Production stamp used to mark the pkgmap file on the output volumes. Provides a means for distinguishing between production copies of a version if more than one is in use at a time. If PSTAMP is not defined, the default is used. The default consists of the UNIX system machine name followed by the string "*YYMMDDHHMM*" (year, month, date, hour, minutes).

INTONLY
: Indicates that the package should only be installed interactively when set to any non-NULL value.

PREDEPEND
: Used to maintain compatibility with dependency checking on packages delivered earlier than System V Release 4. Pre-Release 4 dependency checks were based on whether or not the name file for the required package existed in the /usr/options directory. This directory is not maintained for Release 4 packages because the depend file is used for checking dependencies. However, entries can be created in this directory to maintain compatibility. Setting the PREDEPEND parameter to y or yes creates a /usr/options entry for the package. (Packages new for Release 4 do not need to use this parameter.)

EXAMPLES

Here is a sample pkginfo:

```
PKG="oam"
NAME="OAM Installation Utilities"
VERSION="3"
VENDOR="AT&T"
HOTLINE="1-800-ATT-BUGS"
EMAIL="attunix!olsen"
VSTOCK="0122c3f5566"
CATEGORY="system.essential"
ISTATES="S 2"
RSTATES="S 2"
```

NOTES

Developers may define their own installation parameters by adding a definition to this file. A developer-defined parameter must begin with a capital letter, followed by lowercase letters.

NAME

pkgmap - package contents description file

DESCRIPTION

pkgmap is an ASCII file that provides a complete listing of the package contents. It is automatically generated by pkgmk(1) using the information in the prototype file.

Each entry in pkgmap describes a single "deliverable object file." A deliverable object file includes shell scripts, executable objects, data files, directories, and so on. The entry consists of several fields of information, each field separated by a space. The fields are described below and must appear in the order shown.

part An optional field designating the part number in which the object resides. A part is a collection of files, and is the atomic unit by which a package is processed. A developer can choose the criteria for grouping files into a part (for example, based on class). If no value is defined in this field, part 1 is assumed.

ftype A one-character field that indicates the file type. Valid values are:

- f a standard executable or data file
- e a file to be edited upon installation or removal
- v volatile file (one whose contents are expected to change)
- d directory
- x an exclusive directory
- l linked file
- p named pipe
- c character special device
- b block special device
- i installation script or information file
- s symbolic link

class The installation class to which the file belongs. This name must contain only alphanumeric characters and be no longer than 12 characters. It is not specified if the ftype is i (information file).

pathname The pathname where the object will reside on the target machine, such as /usr/bin/mail. Relative pathnames (those that do not begin with a slash) indicate that the file is relocatable.

For linked files (ftype is either l or s), pathname must be in the form of *path1=path2*, with *path1* specifying the destination of the link and *path2* specifying the source of the link.

pathname may contain variables which support relocation of the file. A $*parameter* may be embedded in the pathname structure. $BASEDIR can be used to identify the parent directories of the path hierarchy, making the entire package easily relocatable. Default values for *parameter* and BASEDIR must be supplied in the pkginfo file and may be overridden at installation.

major The major device number. The field is only specified for block or character special devices.

minor The minor device number. The field is only specified for block or character special devices.

mode The octal mode of the file (for example, 0664). A question mark (?) indicates that the mode will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files, packaging information files or non-installable files.

owner The owner of the file (for example, `bin` or `root`). The field is limited to 14 characters in length. A question mark (`?`) indicates that the owner will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or non-installable files. It is used optionally with a package information file. If used, it indicates with what owner an installation script will be executed.

Can be a variable specification in the form of $[A-Z]. Will be resolved at installation time.

group The group to which the file belongs (for example, "bin" or "sys"). The field is limited to 14 characters in length. A question mark (?) indicates that the group will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or non-installable files. It is used optionally with a package information file. If used, it indicates with what group an installation script will be executed.

Can be a variable assignment in the form of `$[A-Z]`. Will be resolved at installation time.

size The actual size of the file in bytes. This field is not specified for named pipes, special devices, directories or linked files.

cksum The checksum of the file contents. This field is not specified for named pipes, special devices, directories or linked files.

modtime The time of last modification, as reported by the `stat`(2) function call. This field is not specified for named pipes, special devices, directories or linked files.

Each `pkgmap` must have one line that provides information about the number and maximum size (in 512-byte blocks) of parts that make up the package. This line is in the following format:

: *number_of_parts maximum_part_size*

Lines that begin with "#" are comment lines and are ignored.

When files are saved during installation before they are overwritten, they are normally just copied to a temporary pathname. However, for files whose mode includes execute permission (but which are not editable), the existing version is linked to a temporary pathname and the original file is removed. This allows processes which are executing during installation to be overwritten.

EXAMPLES

The following is an example of a `pkgmap` file.

```
:2 500
1 i pkginfo 237 1179 541296672
1 b class1 /dev/rmt/ctape 17 134 0644 root other
1 c class1 /dev/rmt/ctape 17 134 0644 root other
1 d none bin 0755 root bin
1 f none bin/INSTALL 0755 root bin 11103 17954 541295535
1 f none bin/REMOVE 0755 root bin 3214 50237 541295541
1 l none bin/UNINSTALL=bin/REMOVE
1 f none bin/cmda 0755 root bin 3580 60325 541295567
1 f none bin/cmdb 0755 root bin 49107 51255 541438368
1 f class1 bin/cmdc 0755 root bin 45599 26048 541295599
1 f class1 bin/cmdd 0755 root bin 4648 8473 541461238
1 f none bin/cmde 0755 root bin 40501 1264 541295622
1 f class2 bin/cmdf 0755 root bin 2345 35889 541295574
1 f none bin/cmdg 0755 root bin 41185 47653 541461242
2 d class2 data 0755 root bin
2 p class1 data/apipe 0755 root other
2 d none log 0755 root bin
2 v none log/logfile 0755 root bin 41815 47563 541461333
2 d none save 0755 root bin
2 d none spool 0755 root bin
2 d none tmp 0755 root bin
```

NOTES

The pkgmap file may contain only one entry per unique pathname.

NAME

pkgquest - package question file

DESCRIPTION

pkgquest is an ASCII file that defines questions (and resulting parameters) for packages which require user input during an installation or upgrade. It is created by the software package developer.

Each entry in the pkgquest file is a series of lines that define a prompt for the user to provide a parameter input. The following are the definitions of the lines:

N *parameter_name*
H *header_line*
B *body_line*
F *footer_line*
? *help_message_line*
RI [*lower_bound upper_bound*]
RS [*regular_expression*]
RC
RY

One each and only one each of the N and R lines is required and allowed; however, at least one of the others is necessary to give some indication of what is requested. The lines must be arranged in the order listed. The R lines have a response type specified by I, S, C or Y which indicate that an integer, a string, a character or yes/no, respectively, are expected. Integer responses have an optional range specified, helpful for menus as well as parameters requiring integer values. String response may have a regular expression used to verify correct responses. The following defines how the lines appear on the terminal:

package_name Package Query *#1*
zero to ten header lines
zero or more body lines (split into pages if all won't fit on display)
zero to ten footer lines

The help lines are displayed only if requested and appear on the terminal as follows:

package_name Package Query *#1* Help
one or more help lines, as many as necessary
Press RETURN to return to the *package_name* Package Query *#1* screen.

If no help lines are specified, a default message stating that no help is available is presented. If the help lines cannot all be displayed on the screen at once, they will be split into pages.

A package cannot request input containing ASCII codes 0x00 to 0x1f or 0x7f to 0xff, since those are reserved for pkgquest(1). In addition, when the user enters ? followed by a newline, the help message will be displayed.

There is no required order in which the questions must be specified within the file, except they will be displayed as ordered.

EXAMPLES

Here is a sample pkgquest (for the package nsu):

```
NPTNUM
F Enter the number of pseudo-terminal devices
F        to configure on your system.
? NOTE: since each pseudo-terminal device configured
?       allocates memory and streams buffers, choose only
?       the number of terminals you really require.
RI 0 256
```

NOTES

The header and body sections are provided for those packages wishing to provide long messages to the user relevant to the question at hand. It is probably better to put such information into the help section with a statement noting that help is available.

FILES

`/var/sadm/pkg/*/install/questions` location of pkgquest file

SEE ALSO

pkgask(1), pkgquest(1).

NAME

pnch - file format for card images

DESCRIPTION

The PNCH format is a convenient representation for files consisting of card images in an arbitrary code.

A PNCH file is a simple concatenation of card records. A card record consists of a single control byte followed by a variable number of data bytes. The control byte specifies the number (which must lie in the range 0-80) of data bytes that follow. The data bytes are 8-bit codes that constitute the card image. If there are fewer than 80 data bytes, it is understood that the remainder of the card image consists of trailing blanks.

NAME

ppp - Point-to-Point Protocol (PPP)

SYNOPSIS

ppp

DESCRIPTION

The Point-to-Point protocol (PPP) is a method for transmitting datagrams over point-to-point serial links. The protocol and configuration information is described in RFC 1171 and RFC 1172. PPP is not IP specific like SLIP, but the current implementation only supports transmission of IP datagrams over serial links. The Point-to-Point protocol is implemented as a multiplexing STREAMS driver (PPPSM) that is linked beneath IP when internetworking is started. The PPPSM manages the routing of IP datagrams between the interfaces presented to IP and the physical links to remote hosts (PPC). It also performs PPP specific operations concerned with negotiating PPP operating parameters when PPCs are established and tearing down PPCs when they are no longer needed.

The interfaces presented to IP are specified in `/etc/strcf` and are created and marked `up` when `slink`(1M) is started. The PPC links are NOT established to remote hosts until a pending datagram intended for a known remote host is detected by the PPPSM. The interfaces presented to IP are marked as point-to-point interfaces and as such have a known destination IP address. There may be a number of different physical links available that can be used to reach the destination host. The PPC links are described in the PPP and UUCP configuration files.

When a PPP data request (IP datagram) is detected, the PPSM will notify the Point-to-Point Connection Information Daemon (PPCID), `in.pppd` [see `pppd`(1M)] that a pending datagram exists for a specific destination IP address. `in.pppd` will then check it's configuration files for information on how to reach the remote host. Using that information, `in.pppd` performs a `uucp`(1) style login to the remote host and negotiates the line characteristics at both the local and remote hosts. Once the negotiation has finished and the PPC is established, the tty representing the link is linked beneath the PPPSM and the PPPSM is given information about the link. The PPPSM now uses the link for its IP datagram traffic. The PPC will continue to exist under the PPPSM until a pre-set count-down timer measuring continuous link inactivity has expired, or the link is broken by administrator command, that is, using `ifconfig`(1M) to mark the interface `down`.

SEE ALSO

`ifconfig`(1M), `slink`(1M), `ppp`(1), `pppd`(1M), `strcf`(4), `hosts`(5), `ppphosts`(5)
RFC 1171, RFC 1172

NAME

ppphosts - Point-to-Point Protocol Host name database

SYNOPSIS

/etc/inet/ppphosts

DESCRIPTION

The /etc/inet/ppphosts file contains information about known PPP hosts. This file contains a single-line entry for each PPP host with the following information:

Remote host name or alias

Inactivity timeout in minutes (optional, default = "forever")

Tty name for direct connection (optional)

Uucp system name for this remote host

Timeout per PPP protocol request (optional, default = 10 seconds)

Maximum number of retries per PPP protocol request (optional, default = 3)

These data items should be separated by "white space". A '#' indicates the beginning of a comment; characters appearing after '#' are ignored.

This file should be created and maintained by the Network Administrator. These guidelines should be followed in creating /etc/inet/ppphosts:

The host name should have a corresponding entry in /etc/hosts [see hosts(4)]

Optional parameters may be defaulted by using the '-' place-holder

The tty name (if other than '-') should have a corresponding entry in /usr/lib/uucp/Devices

The uucp system name should have a corresponding entry in /usr/lib/uucp/Systems

The contents of this file will be used by the pppd daemon [see pppd(1M)].

EXAMPLES

Example 1 - Typical /usr/inet/ppphosts File

```
#
#            Inactivity   Tty name   UUCP     ACK
#Host        timeout      (direct    system   timeout     ACK
#name        (minutes)    connect)   name     (seconds)   retries
#
homer_ppp    5            -          homer
bart_ppp     -            tty01      bart     -           5
```

The guidelines shown in *Network File System Administration*, show how typical data in the /etc/inet/ppphosts, /usr/lib/uucp/Systems, and /usr/lib/uucp/Devices files could be used for reaching a PPP host.

Example 2 - A Direct Line between PPP Hosts

```
ppphosts:     bart_ppp   5   tty01   bart
hosts:        128.2.129.2   bart_ppp
Devices:      Direct   tty01   -   9600   direct
Systems:      bart   Any   Direct   9600   -   login:
Password:     PPP_password
```

The special user_name nppp will initiate the remote login session; also note that following four network-dependent data items in the above table entries must match: bart_ppp, bart, tty01, and Direct.

Example 3 - A Dial-up Line between PPP Hosts

```
ppphosts:     homer_ppp   5   -   homer
hosts:        28.2.129.5   homer_ppp
Systems:      homer   Any   ACU   2400   555-1234   login: nppp
Password:     PPP_password
```

USER CONSIDERATIONS

The Network Administrator should ensure consistent entries in the /etc/inet/ppphosts , /usr/lib/uucp/Systems , and /usr/lib/uucp/Devices for the PPP hosts. The remote login request needs to specify \&nppp as its user_name. PPP creates /usr/lib/ppp/named_ppp with uid nppp. The remote login must use this uid to communicate through the named pipe to pppd.

FILES

/etc/hosts
/etc/inet/ppphosts
/usr/lib/uucp/Systems
/usr/lib/uucp/Devices

SEE ALSO

uucp(1), pppd(1M), host(4).

NAME

prf - operating system profiler

DESCRIPTION

The special file /dev/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 /dev/prf is a pseudo-device with no associated hardware.

FILES

/dev/prf

NOTES

By default, the prf device is not configured into the kernel for Motorola processors. To turn it on, you must edit the /stand/system file, and add the prf modules to the list of included modules.

SEE ALSO

profiler(1M)

NAME

/proc - process file system

DESCRIPTION

/proc is a file system that provides access to the image of each active process in the system. The name of each entry in the /proc directory is a decimal number corresponding to the process ID. The owner of each "file" is determined by the process's user-ID.

Standard system call interfaces are used to access /proc files: open, close, read, write, and ioctl. An open for reading and writing enables process control; a read-only open allows inspection but not control. As with ordinary files, more than one process can open the same /proc file at the same time. Exclusive open is provided to allow controlling processes to avoid collisions: an open for writing that specifies O_EXCL fails if the file is already open for writing; if such an exclusive open succeeds, subsequent attempts to open the file for writing, with or without the O_EXCL flag, fail until the exclusively-opened file descriptor is closed. (Exception: a super-user open that does not specify O_EXCL succeeds even if the file is exclusively opened.) There can be any number of read-only opens, even when an exclusive write open is in effect on the file.

Data may be transferred from or to any locations in the traced process's address space by applying lseek to position the file at the virtual address of interest followed by read or write. The PIOCMAP operation can be applied to determine the accessible areas (mappings) of the address space. A contiguous area of the address space may appear as multiple mappings due to varying read/write/execute permissions. I/O transfers may span contiguous mappings. An I/O request extending into an unmapped area is truncated at the boundary.

Information and control operations are provided through ioctl. These have the form:

```
#include <sys/types.h>
#include <sys/signal.h>
#include <sys/fault.h>
#include <sys/syscall.h>
#include <sys/procfs.h>
void *p;
retval = ioctl(fildes, code, p);
```

The argument *p* is a generic pointer whose type depends on the specific ioctl code. Where not specifically mentioned below, its value should be zero. sys/procfs.h contains definitions of ioctl codes and data structures used by the operations. Certain operations can be performed only if the process file is open for writing; these include all operations that affect process control.

Process information and control operations involve the use of sets of flags. The set types sigset_t, fltset_t, and sysset_t correspond, respectively, to signal, fault, and system call enumerations defined in sys/signal.h, sys/fault.h, and sys/syscall.h. Each set type is large enough to hold flags for its own enumeration. Although they are of different sizes, they have a common structure and can be manipulated by these macros:

```
prfillset(&set);              /* turn on all flags in set */
premptyset(&set);             /* turn off all flags in set */
praddset(&set, flag);         /* turn on the specified flag */
prdelset(&set, flag);         /* turn off the specified flag */
r = prismember(&set, flag);   /* != 0 iff flag is turned on */
```

One of `prfillset` or `premptyset` must be used to initialize `set` before it is used in any other operation. `flag` must be a member of the enumeration corresponding to `set`.

The allowable `ioctl` codes follow. Those requiring write access are marked with an asterisk (*). Except where noted, an `ioctl` to a process that has terminated elicits the error ENOENT.

PIOCSTATUS

This returns status information for the process; *p* is a pointer to a `prstatus` structure which is defined in the header <`sys/procfs.h`>. The `pr_status` structure contains at least the following fields, but not necessarily in this order.

```
typedef struct prstatus {
  long      pr_flags;           /* Process flags */
  short     pr_why;             /* Reason for process stop (if stopped) */
  short     pr_what;            /* More detailed reason */
  struct siginfo pr_info;       /* Info associated with signal or fault */
  exblk_t   pr_exblks[NDMPBL];  /* Exception blks for machine exceptions */
                                   (For m88k only)
  short     pr_cursig;          /* Current signal */
  sigset_t  pr_sigpend;         /* Set of other pending signals */
  sigset_t  pr_sighold;         /* Set of held signals */
  struct sigaltstack pr_altstack;/* Alternate signal stack info */
  struct sigaction pr_action;   /* Signal action for current signal */
  pid_t     pr_pid;             /* Process id */
  pid_t     pr_ppid;            /* Parent process id */
  pid_t     pr_pgrp;            /* Process group id */
  pid_t     pr_sid;             /* Session id */
  timestruc_t pr_utime;         /* Process user cpu time */
  timestruc_t pr_stime;         /* Process system cpu time */
  timestruc_t pr_cutime;        /* Sum of children's user times */
  timestruc_t pr_cstime;        /* Sum of children's system times */
  char      pr_clname[8];       /* Scheduling class name */
  long      pr_filler[20];      /* Filler area for future expansion */
  long      pr_instr;           /* Current instruction */
  gregset_t pr_reg;             /* General registers */
} prstatus_t;
```

`pr_flags` is a bit-mask holding these flags:

PR_STOPPED	process is stopped
PR_ISTOP	process is stopped on an event of interest (see PIOCSTOP)
PR_DSTOP	process has a stop directive in effect (see PIOCSTOP)
PR_ASLEEP	process is in an interruptible sleep within a system call

PR_FORK	process has its inherit-on-fork flag set (see PIOCSFORK)
PR_RLC	process has its run-on-last-close flag set (see PIOCSRLC)
PR_PTRACE	process is being traced via ptrace
PR_PCINVAL	process program counter refers to an invalid address
PR_ISSYS	process is a system process (see PIOCSTOP)

pr_why and pr_what together describe, for a stopped process, the reason that the process is stopped. Possible values of pr_why are:

PR_REQUESTED
: indicates that the process stopped because PIOCSTOP was applied; pr_what is unused in this case.

PR_SIGNALLED
: indicates that the process stopped on receipt of a signal (see PIOCSTRACE); pr_what holds the signal number that caused the stop (for a newly-stopped process, the same value is in pr_cursig).

PR_FAULTED
: indicates that the process stopped on incurring a hardware fault (see PIOCSFAULT); pr_what holds the fault number that caused the stop.

PR_SYSENTRY and PR_SYSEXIT
: indicate a stop on entry to or exit from a system call (see PIOCSENTRY and PIOCSEXIT); pr_what holds the system call number.

PR_JOBCONTROL
: indicates that the process stopped due to the default action of a job control stop signal (see sigaction); pr_what holds the stopping signal number.

pr_info, when the process is in a PR_SIGNALLED or PR_FAULTED stop, contains additional information pertinent to the particular signal or fault (see sys/siginfo.h).

pr_exblks exists only for the M88000 family of processors. This field contains the exception blocks generated when the machine exception occurred. The exception blocks will be valid **only** when the signal in pr_info is the result of a machine exception. The SI_MACHINEXCEP macro (found in sys/siginfo.h) detects whether a given pr_info is the result of a machine exception. The number of valid exception blocks is contained in the _ncodes field of the pr_info structure. Note that the _exblks pointer in the pr_info structure will not be valid.

pr_cursig names the current signal—that is, the next signal to be delivered to the process. pr_sigpend identifies any other pending signals. pr_sighold identifies those signals whose delivery is being delayed if sent to the process.

pr_altstack contains the alternate signal stack information for the process (see sigaltstack). pr_action contains the signal action information pertaining to the current signal (see sigaction); it is undefined if pr_cursig is zero.

pr_pid, pr_ppid, pr_pgrp, and pr_sid are, respectively, the process id, the id of the process's parent, the process's process group id, and the process's session id.

pr_utime, pr_stime, pr_cutime, and pr_cstime are, respectively, the user and system time consumed by the process, and the cumulative user and system time consumed by the process's children, in seconds and nanoseconds.

pr_clname contains the name of the process's scheduling class.

The pr_filler area is reserved for future use.

pr_instr contains the machine instruction to which the program counter refers. The amount of data retrieved from the process is machine-dependent. On the M88000 family of processors it is 4 bytes. In general, the size is that of the machine's smallest instruction. If the program counter refers to an invalid address, PR_PCINVAL is set and pr_instr is undefined.

pr_reg is an array holding the contents of the general registers. On the M88000 family of processors the predefined constants R_R31 R_PSR R_XIP, R_NIP, and R_FIP can be used as indices to refer to the corresponding registers.

PIOCSTOP*, PIOCWSTOP

PIOCSTOP directs the process to stop and waits until it has stopped; PIOCWSTOP simply waits for the process to stop. These operations complete when the process stops on an event of interest, immediately if already so stopped. If *p* is non-zero it points to an instance of prstatus_t to be filled with status information for the stopped process.

An "event of interest" is either a PR_REQUESTED stop or a stop that has been specified in the process's tracing flags (set by PIOCSTRACE, PIOCSFAULT, PIOCSENTRY, and PIOCSEXIT). A PR_JOBCONTROL stop is specifically not an event of interest. (A process may stop twice due to a stop signal, first showing PR_SIGNALLED if the signal is traced and again showing PR_JOBCONTROL if the process is set running without clearing the signal.) If the process is controlled by ptrace, it comes to a PR_SIGNALLED stop on receipt of any signal; this is an event of interest only if the signal is in the traced signal set. If PIOCSTOP is applied to a process that is stopped, but not on an event of interest, the stop directive takes effect when the process is restarted by the competing mechanism; at that time the process enters a PR_REQUESTED stop before executing any user-level code.

ioctls are interruptible by signals so that, for example, an alarm can be set to avoid waiting forever for a process that may never stop on an event of interest. If PIOCSTOP is interrupted, the stop directive remains in effect even though the ioctl returns an error.

A system process (indicated by the PR_ISSYS flag) never executes at user level, has no user-level address space visible through /proc, and cannot be stopped. Applying PIOCSTOP or PIOCWSTOP to a system process elicits the error EBUSY.

PIOCRUN*

The traced process is made runnable again after a stop. If *p* is non-zero it points to a prrun structure describing additional actions to be performed:

```
typedef struct prrun {
  long      pr_flags;       /* Flags */
  sigset_t  pr_trace;       /* Set of signals to be traced */
  sigset_t  pr_sighold;     /* Set of signals to be held */
  fltset_t  pr_fault;       /* Set of faults to be traced */
  caddr_t   pr_vaddr;       /* Virtual address at which to resume */
  long      pr_filler[8];   /* Filler area for future expansion */
} prrun_t;
```

pr_flags is a bit-mask describing optional actions; the remainder of the entries are meaningful only if the appropriate bits are set in pr_flags. pr_filler is reserved for future use; this area must be filled with zeros by the user's program. Flag definitions:

PRCSIG clears the current signal, if any (see PIOCSSIG)

PRCFAULT clears the current fault, if any (see PIOCCFAULT)

PRSTRACE sets the traced signal set to pr_trace (see PIOCSTRACE)

PRSHOLD sets the held signal set to pr_sighold (see PIOCSHOLD)

PRSFAULT sets the traced fault set to pr_fault (see PIOCSFAULT)

PRSVADDR sets the address at which execution resumes to pr_vaddr

PRSTEP directs the process to single-step — that is, to run and to execute a single machine instruction. On completion of the instruction, a hardware trace trap occurs. If FLTTRACE is being traced, the process stops, otherwise it is sent SIGTRAP; if SIGTRAP is being traced and not held, the process stops. This operation requires hardware support and may not be implemented on all processors.

PRSABORT is meaningful only if the process is in a PR_SYSENTRY stop or is marked PR_ASLEEP; it instructs the process to abort execution of the system call (see PIOCSENTRY, PIOCSEXIT).

PRSTOP directs the process to stop again as soon as possible after resuming execution (see PIOCSTOP). In particular if the process is stopped on PR_SIGNALLED or PR_FAULTED, the next stop will show PR_REQUESTED, no other stop will have intervened, and the process will not have executed any user-level code.

PIOCRUN fails (EBUSY) if applied to a process that is not stopped on an event of interest. Once PIOCRUN has been applied, the process is no longer stopped on an event of interest even if, due to a competing mechanism, it remains stopped.

PIOCSTRACE*

This defines a set of signals to be traced: the receipt of one of these signals causes the traced process to stop. The set of signals is defined via an instance of sigset_t addressed by *p*. Receipt of SIGKILL cannot be traced.

If a signal that is included in the held signal set is sent to the traced process, the signal is not received and does not cause a process stop until it is removed from the held signal set, either by the process itself or by setting the held signal set with PIOCSHOLD or the PRSHOLD option of PIOCRUN.

PIOCGTRACE

The current traced signal set is returned in an instance of sigset_t addressed by *p*.

PIOCSSIG*

The current signal and its associated signal information are set according to the contents of the siginfo structure addressed by *p* (see sys/siginfo.h). If the specified signal number is zero or if *p* is zero, the current signal is cleared. The semantics of this operation are different from those of kill or PIOCKILL in that the signal is delivered to the process immediately after execution is resumed (even if it is being held) and an additional PR_SIGNALLED stop does not intervene even if the signal is traced. Setting the current signal to SIGKILL terminates the process immediately, even if it is stopped.

PIOCKILL*

A signal is sent to the process with semantics identical to those of kill; *p* points to an int naming the signal. Sending SIGKILL terminates the process immediately.

PIOCUNKILL*

A signal is deleted, that is, it is removed from the set of pending signals; the current signal (if any) is unaffected. *p* points to an int naming the signal. It is an error to attempt to delete SIGKILL.

PIOCGHOLD, PIOCSHOLD*

PIOCGHOLD returns the set of held signals (signals whose delivery will be delayed if sent to the process) in an instance of sigset_t addressed by *p*. PIOCSHOLD correspondingly sets the held signal set but does not allow SIGKILL or SIGSTOP to be held.

PIOCMAXSIG, PIOCACTION

These operations provide information about the signal actions associated with the traced process (see sigaction). PIOCMAXSIG returns, in the int addressed by *p*, the maximum signal number understood by the system. This can be used to allocate storage for use with the PIOCACTION operation, which returns the traced process's signal actions in an array of sigaction structures addressed by *p*. Signal numbers are displaced by 1 from array indices, so that the action for signal number *n* appears in position *n*-1 of the array.

PIOCSFAULT*

This defines a set of hardware faults to be traced: on incurring one of these faults the traced process stops. The set is defined via an instance of fltset_t addressed by *p*. Fault names are defined in sys/fault.h and include the following. Some of these may not occur on all processors; there may be processor-specific faults in addition to these.

FLTILL	illegal instruction
FLTPRIV	privileged instruction
FLTBPT	breakpoint trap
FLTTRACE	trace trap
FLTACCESS	memory access fault
FLTBOUNDS	memory bounds violation
FLTIOVF	integer overflow

FLTIZDIV	integer zero divide
FLTFPE	floating-point exception
FLTSTACK	unrecoverable stack fault
FLTPAGE	recoverable page fault

When not traced, a fault normally results in the posting of a signal to the process that incurred the fault. If the process stops on a fault, the signal is posted to the process when execution is resumed unless the fault is cleared by PIOCCFAULT or by the PRCFAULT option of PIOCRUN. FLTPAGE is an exception; no signal is posted. There may be additional processor-specific faults like this. pr_info in the prstatus structure identifies the signal to be sent and contains machine-specific information about the fault.

PIOCGFAULT
: The current traced fault set is returned in an instance of fltset_t addressed by *p*.

PIOCCFAULT*
: The current fault (if any) is cleared; the associated signal is not sent to the process.

PIOCSENTRY*, PIOCSEXIT*
: These operations instruct the process to stop on entry to or exit from specified system calls. The set of syscalls to be traced is defined via an instance of sysset_t addressed by *p*.

When entry to a system call is being traced, the traced process stops after having begun the call to the system but before the system call arguments have been fetched from the process. When exit from a system call is being traced, the traced process stops on completion of the system call just prior to checking for signals and returning to user level. At this point all return values have been stored into the traced process's saved registers.

If the traced process is stopped on entry to a system call (PR_SYSENTRY) or when sleeping in an interruptible system call (PR_ASLEEP is set), it may be instructed to go directly to system call exit by specifying the PRSABORT flag in a PIOCRUN request. Unless exit from the system call is being traced the process returns to user level showing error EINTR.

PIOCGENTRY, PIOCGEXIT
: These return the current traced system call entry or exit set in an instance of sysset_t addressed by *p*.

PIOCSFORK*, PIOCRFORK*
: PIOCSFORK sets the inherit-on-fork flag in the traced process: the process's tracing flags are inherited by the child of a fork. PIOCRFORK turns this flag off: child processes start with all tracing flags cleared.

PIOCSRLC*, PIOCRRLC*
: PIOCSRLC sets the run-on-last-close flag in the traced process: when the last writable /proc file descriptor referring to the traced process is closed, all of the process's tracing flags are cleared, any outstanding stop directive is canceled, and if the process is stopped, it is set running as though PIOCRUN had been applied to it. PIOCRRLC turns this flag off: the process's tracing flags are retained and the process is not set running when the process file is closed.

PIOCGREG, PIOCSREG*
These operations respectively get and set the saved process registers into or out of an array addressed by *p*; the array has type gregset_t. Register contents are accessible using a set of predefined indices (see PIOCSTATUS). Only certain bits of the processor-status word (PSW) can be modified by PIOCSREG. On the M88000 family of processors these include the Serial Mode, Carry, Byte Order and Misaligned Access Enable bits. Other privileged registers cannot be modified at all. PIOCSREG fails (EBUSY) if applied to a process that is not stopped on an event of interest. Currently on the M88000 family of processors no floating point registers are available via this ioctl.

PIOCGFPREG, PIOCSFPREG*
These operations respectively get and set the saved process floating-point registers into or out of a structure addressed by *p*; the structure has type fpregset_t. An error (EINVAL) is returned if there is no floating-point hardware on the machine. PIOCSFPREG fails (EBUSY) if applied to a process that is not stopped on an event of interest.

PIOCNICE*
The traced process's nice priority is incremented by the amount contained in the int addressed by *p*. Only the super-user may better a process's priority in this way, but any user may make the priority worse.

PIOCPSINFO
This returns miscellaneous process information such as that reported by ps(1). *p* is a pointer to a prpsinfo structure containing at least the following fields:

```
typedef struct prpsinfo {
  char    pr_state;      /* numeric process state (see pr_sname) */
  char    pr_sname;      /* printable character representing pr_state */
  char    pr_zomb;       /* !=0: process terminated but not waited for */
  char    pr_nice;       /* nice for cpu usage */
  u_long  pr_flag;       /* process flags */
  uid_t   pr_uid;        /* real user id */
  gid_t   pr_gid;        /* real group id */
  pid_t   pr_pid;        /* unique process id */
  pid_t   pr_ppid;       /* process id of parent */
  pid_t   pr_pgrp;       /* pid of process group leader */
  pid_t   pr_sid;        /* session id */
  caddr_t pr_addr;       /* physical address of process */
  long    pr_size;       /* size of process image in pages */
  long    pr_rssize;     /* resident set size in pages */
  caddr_t pr_wchan;      /* wait addr for sleeping process */
  timestruc_t pr_start;  /* process start time, sec+nsec since epoch */
  timestruc_t pr_time;        /* usr+sys cpu time for this process */
  long    pr_pri;             /* priority, high value is high priority */
  char    pr_oldpri; /* pre-System V Release 4.0, low value is high priority */
  char    pr_cpu;    /* pre-System V release 4.0, cpu usage for scheduling */
  dev_t   pr_ttydev;          /* controlling tty device (PRNODEV if none) */
  char    pr_clname[8];       /* Scheduling class name */
  char    pr_fname[16];       /* last component of execed pathname */
  char    pr_psargs[PRARGSZ]; /* initial characters of arg list */
  long    pr_filler[20];      /* for future expansion */
```

```
} prpsinfo_t;
```

Some of the entries in prpsinfo, such as pr_state and pr_flag, are system-specific and should not be expected to retain their meanings across different versions of the operating system. pr_addr is a vestige of the past and has no real meaning in current systems.

PIOCPSINFO can be applied to a zombie process (one that has terminated but whose parent has not yet performed a wait on it).

PIOCNMAP, PIOCMAP

These operations provide information about the memory mappings (virtual address ranges) associated with the traced process. PIOCNMAP returns, in the int addressed by *p*, the number of mappings that are currently active. This can be used to allocate storage for use with the PIOCMAP operation, which returns the list of currently active mappings. For PIOCMAP, *p* addresses an array of elements of type prmap_t; one array element (one structure) is returned for each mapping, plus an additional element containing all zeros to mark the end of the list.

```
typedef struct prmap {
  caddr_t  pr_vaddr;       /* Virtual address base */
  u_long   pr_size;        /* Size of mapping in bytes */
  off_t    pr_off;         /* Offset into mapped object, if any */
  long     pr_mflags;      /* Protection and attribute flags */
  long     pr_filler[4];   /* Filler for future expansion */
} prmap_t;
```

pr_vaddr is the virtual address base (the lower limit) of the mapping within the traced process and pr_size is its size in bytes. pr_off is the offset within the mapped object (if any) to which the address base is mapped.

pr_mflags is a bit-mask of protection and attribute flags:

MA_READ	mapping is readable by the traced process
MA_WRITE	mapping is writable by the traced process
MA_EXEC	mapping is executable by the traced process
MA_SHARED	mapping changes are shared by the mapped object
MA_BREAK	mapping is grown by the brk system call
MA_STACK	mapping is grown automatically on stack faults

PIOCOPENM

The return value *retval* provides a read-only file descriptor for a mapped object associated with the traced process. If *p* is zero the traced process's execed file (its a.out file) is found. This enables a debugger to find the object file symbol table without having to know the path name of the executable file. If *p* is non-zero it points to a caddr_t containing a virtual address within the traced process and the mapped object, if any, associated with that address is found; this can be used to get a file descriptor for a shared library that is attached to the process. On error (invalid address or no mapped object for the designated address), -1 is returned.

PIOCCRED

Fetch the set of credentials associated with the process. *p* points to an instance of prcred_t, which is filled by the operation:

```
typedef struct prcred {
    uid_t    pr_euid;      /* Effective user id */
    uid_t    pr_ruid;      /* Real user id */
    uid_t    pr_suid;      /* Saved user id (from exec) */
    uid_t    pr_egid;      /* Effective group id */
    uid_t    pr_rgid;      /* Real group id */
    uid_t    pr_sgid;      /* Saved group id (from exec) */
    u_int    pr_ngroups;   /* Number of supplementary groups */
} prcred_t;
```

PIOCGROUPS
Fetch the set of supplementary group IDs associated with the process. *p* points to an array of elements of type uid_t, which will be filled by the operation. PIOCCRED can be applied beforehand to determine the number of groups (pr_ngroups) that will be returned and the amount of storage that should be allocated to hold them.

PIOCGETPR, PIOCGETU
These operations copy, respectively, the traced process's proc structure and user area into the buffer addressed by *p*. They are provided for completeness but it should be unnecessary to access either of these structures directly since relevant status information is available through other control operations. Their use is discouraged because a program making use of them is tied to a particular version of the operating system.

PIOCGETPR can be applied to a zombie process (see PIOCPSINFO).

NOTES

Each operation (ioctl or I/O) is guaranteed to be atomic with respect to the traced process, except when applied to a system process.

For security reasons, except for the super-user, an open of a /proc file fails unless both the user-ID and group-ID of the caller match those of the traced process and the process's object file is readable by the caller. Files corresponding to setuid and setgid processes can be opened only by the super-user. Even if held by the super-user, an open process file descriptor becomes invalid if the traced process performs an exec of a setuid/setgid object file or an object file that it cannot read. Any operation performed on an invalid file descriptor, except close, fails with EAGAIN. In this situation, if any tracing flags are set and the process file is open for writing, the process will have been directed to stop and its run-on-last-close flag will have been set (see PIOCSRLC). This enables a controlling process (if it has permission) to reopen the process file to get a new valid file descriptor, close the invalid file descriptor, and proceed. Just closing the invalid file descriptor causes the traced process to resume execution with no tracing flags set. Any process not currently open for writing via /proc but that has left-over tracing flags from a previous open and that execs a setuid/setgid or unreadable object file will not be stopped but will have all its tracing flags cleared.

For reasons of symmetry and efficiency there are more control operations than strictly necessary. On the M88000 family of processors reference platform which support the Binary Compatible Standard, BCS, the ioctl operations described here may not work with programs compiled and linked on non-UNIX System V/68 or V/88 Release 4 systems.

FILES

/proc	directory (list of active processes)
/proc/*nnnnn*	process image

DIAGNOSTICS

Errors that can occur in addition to the errors normally associated with file system access:

ENOENT — the traced process has exited after being opened

EIO — I/O was attempted at an illegal address in the traced process

EBADF — an I/O or ioctl operation requiring write access was attempted on a file descriptor not open for writing

EBUSY — PIOCSTOP or PIOCWSTOP was applied to a system process; an exclusive open was attempted on a process file already already open for writing; an open for writing was attempted and an exclusive open is in effect on the process file; PIOCRUN, PIOCSREG or PIOCSFPREG was applied to a process not stopped on an event of interest; an attempt was made to mount /proc when it is already mounted.

EPERM — someone other than the super-user attempted to better a process's priority by issuing PIOCNICE

ENOSYS — an attempt was made to perform an unsupported operation (such as create, remove, link, or unlink) on an entry in /proc

EFAULT — an I/O or ioctl request referred to an invalid address in the controlling process

EINVAL — in general this means that some invalid argument was supplied to a system call. The list of conditions eliciting this error includes: the ioctl code is undefined; an ioctl operation was issued on a file descriptor referring to the /proc directory; an out-of-range signal number was specified with PIOCSSIG, PIOCKILL, or PIOCUNKILL; SIGKILL was specified with PIOCUNKILL; an illegal virtual address was specified in a PIOCOPENM request; PIOCGFPREG or PIOCSFPREG was issued on a machine without floating-point hardware.

EINTR — a signal was received by the controlling process while waiting for the traced process to stop via PIOCSTOP or PIOCWSTOP

EAGAIN — the traced process has performed an exec of a setuid/setgid object file or of an object file that it cannot read; all further operations on the process file descriptor (except close) elicit this error.

SEE ALSO

open(2), ptrace(2), sigaction(2), signal(2), sigset(2).

NAME

profile - setting up an environment at login time

SYNOPSIS

/etc/profile
$HOME/.profile

DESCRIPTION

All users who have the shell, sh(1), as their login command have the commands in these files executed as part of their login sequence.

/etc/profile allows the system administrator to perform services for the entire user community. Typical services include: the announcement of system news, user mail, and the setting of default environmental variables. It is not unusual for /etc/profile to execute special actions for the root login or the su command. Computers running outside the U.S. Eastern time zone should have the line

```
. /etc/TIMEZONE
```

included early in /etc/profile [see timezone(4)].

The file $HOME/.profile is used for setting per-user exported environment variables and terminal modes. The following example is typical (except for the comments):

```
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 022
# Tell me when new mail comes in
MAIL=/var/mail/$LOGNAME
# Add my /usr/usr/bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Set terminal type
TERM=$ {L0:-u/n/k/n/o/w/n} # gnar.invalid
while :
do
      if [ -f ${TERMINFO:-/usr/share/lib/terminfo}/?/$TERM ]
      then break
      elif [ -f /usr/share/lib/terminfo/?/$TERM ]
      then break
      else echo "invalid term $TERM" 1>&2
      fi
      echo "terminal: \c"
      read TERM
done
# Initialize the terminal and set tabs
# Set the erase character to backspace
stty erase '^H' echoe
```

FILES

/etc/TIMEZONE	timezone environment
$HOME/.profile	user-specific environment
/etc/profile1	

SEE ALSO

env(1), login(1), mail(1), sh(1), stty(1), su(1M), tput(1), terminfo(4), timezone(4), environ(5), term(5).

NOTES

Care must be taken in providing system-wide services in /etc/profile. Personal .profile files are better for serving all but the most global needs.

NAME

protocols - protocol name data base

SYNOPSIS

/etc/protocols

DESCRIPTION

The protocols file contains information regarding the known protocols used in the DARPA Internet. For each protocol a single line should be present with the following information:

official-protocol-name protocol-number aliases

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Protocol names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

EXAMPLE

The following is a sample database:

```
#
# Internet (IP) protocols
#
ip      0       IP      # internet protocol, pseudo protocol number
icmp    1       ICMP    # internet control message protocol
ggp     3       GGP     # gateway-gateway protocol
tcp     6       TCP     # transmission control protocol
pup     12      PUP     # PARC universal packet protocol
udp     17      UDP     # user datagram protocol
```

FILES

/etc/protocols

SEE ALSO

getprotoent(3N)

NOTES

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

prototype - package information file

DESCRIPTION

prototype is an ASCII file used to specify package information. Each entry in the file describes a single deliverable object. An object may be a data file, directory, source file, executable object, and so on. This file is generated by the package developer.

Entries in a prototype file consist of several fields of information separated by white space. Comment lines begin with a "#" and are ignored. The fields are described below and must appear in the order shown.

part An optional field designating the part number in which the object resides. A part is a collection of files, and is the atomic unit by which a package is processed. A developer can choose criteria for groupig files into a part (for example, based on class). If this field is not used, part 1 is assumed.

ftype A one-character field which indicates the file type. Valid values are:

- f a standard executable or data file
- e a file to be edited upon installation or removal
- v volatile file (one whose contents are expected to change)
- d directory
- x an exclusive directory
- l linked file
- p named pipe
- c character special device
- b block special device
- i installation script or information file
- s symbolic link

class The installation class to which the file belongs. This name must contain only alphanumeric characters and be no longer than 12 characters. The field is not specified for installation scripts. (admin and all classes beginning with capital letters are reserved class names.)

pathname The pathname where the file will reside on the target machine, for example, /usr/bin/mail or bin/ras_proc. Relative pathnames (those that do not begin with a slash) indicate that the file is relocatable. The form

path1=path2

may be used for two purposes: to define a link and to define local pathnames.

For linked files, *path1* indicates the destination of the link and *path2* indicates the source file. (This format is mandatory for linked files.)

For symbolically linked files, *path2* can be a relative pathname, such as ./ or ../. For example, if you enter a line such as

```
s /foo/bar/etc/mount=../usr/sbin/mount
```

path2 (/foo/bar/etc/mount) will be a symbolic link to

../usr/sbin/mount.

For local pathnames, *path1* indicates the pathname an object should have on the machine where the entry is to be installed and *path2* indicates either a relative or fixed pathname to a file on the host machine which contains the actual contents.

A pathname may contain a variable specification, which will be resolved at the time of installation. This specification should have the form $[A-Z].

major The major device number. The field is only specified for block or character special devices.

minor The minor device number. The field is only specified for block or character special devices.

mode The octal mode of the file (for example, 0664). A question mark (`?`) indicates that the mode will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or packaging information files.

owner The owner of the file (for example, `bin` or `root`). The field is limited to 14 characters in length. A question mark (`?`) indicates that the owner will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or packaging information files.

Can be a variable specification in the form of `$[A-Z]`. Will be resolved at installation time.

group The group to which the file belongs (for example, `bin` or `sys`). The field is limited to 14 characters in length. A question mark (`?`) indicates that the group will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or packaging information files.

Can be a variable specification in the form of $[A-Z]. Will be resolved at installation time.

An exclamation point (`!`) at the beginning of a line indicates that the line contains a command. These commands are used to incorporate files in other directories, to locate objects on a host machine, and to set permanent defaults. The following commands are available:

`search` Specifies a list of directories (separated by white space) to search for when looking for file contents on the host machine. The basename of the *path* field is appended to each directory in the ordered list until the file is located.

`include` Specifies a pathname which points to another prototype file to include. Note that `search` requests do not span `include` files.

`default` Specifies a list of attributes (mode, owner, and group) to be used by default if attribute information is not provided for prototype entries which require the information. The defaults do not apply to entries in `include` prototype files.

param=value Places the indicated parameter in the current environment.

The above commands may have variable substitutions embedded within them, as demonstrated in the two example prototype files below.

Before files are overwritten during installation, they are copied to a temporary pathname. The exception to this rule is files whose mode includes execute permission, unless the file is editable (that is, *ftype* is e). For files which meet this exception, the existing version is linked to a temporary pathname, and the original file is removed. This allows processes which are executing during installation to be overwritten.

EXAMPLES

Example 1:

```
!PROJDIR=/usr/proj
!BIN=$PROJDIR/bin
!CFG=$PROJDIR/cfg
!LIB=$PROJDIR/lib
!HDRS=$PROJDIR/hdrs
!search /usr/myname/usr/bin /usr/myname/src /usr/myname/hdrs
i pkginfo=/usr/myname/wrap/pkginfo
i depend=/usr/myname/wrap/depend
i version=/usr/myname/wrap/version
d none /usr/wrap 0755 root bin
d none /usr/wrap/usr/bin 0755 root bin
! search $BIN
f none /usr/wrap/bin/INSTALL 0755 root bin
f none /usr/wrap/bin/REMOVE 0755 root bin
f none /usr/wrap/bin/addpkg 0755 root bin
!default 755 root bin
f none /usr/wrap/bin/audit
f none /usr/wrap/bin/listpkg
f none /usr/wrap/bin/pkgmk
# The logfile starts as a zero length file, since the source
# file has zero length. Later, the size of logfile grows.
v none /usr/wrap/logfile=/usr/wrap/log/zero_length 0644 root bin
# the following specifies a link (dest=src)
l none /usr/wrap/src/addpkg=/usr/wrap/bin/rmpkg
! search $SRC
!default 644 root other
f src /usr/wrap/src/INSTALL.sh
f src /usr/wrap/src/REMOVE.sh
f src /usr/wrap/src/addpkg.c
f src /usr/wrap/src/audit.c
f src /usr/wrap/src/listpkg.c
f src /usr/wrap/src/pkgmk.c
d none /usr/wrap/data 0755 root bin
d none /usr/wrap/save 0755 root bin
d none /usr/wrap/spool 0755 root bin
d none /usr/wrap/tmp 0755 root bin
d src /usr/wrap/src 0755 root bin
```

Example 2:

```
# this prototype is generated by 'pkgproto' to refer
# to all prototypes in my src directory
!PROJDIR=/usr/dew/projx
!include $PROJDIR/src/cmd/prototype
!include $PROJDIR/src/cmd/audmerg/protofile
!include $PROJDIR/src/lib/proto
```

SEE ALSO

pkginfo(4), pkgmk(1)

NOTES

Normally, if a file is defined in the prototype file but does not exist, that file is created at the time of package installation. However, if the file pathname includes a directory that does not exist, the file will not be created. For example, if the prototype file has the following entry:

```
f none /usr/dev/bin/command
```

and that file does not exist, it will be created if the directory /usr/dev/bin already exists or if the prototype also has an entry defining the directory:

```
d none /usr/dev/bin
```

NAME

ptem - STREAMS Pseudo Terminal Emulation module

DESCRIPTION

ptem is a STREAMS module that when used in conjunction with a line discipline and pseudo terminal driver emulates a terminal.

The ptem module must be pushed [see I_PUSH, streamio(7)] onto the slave side of a pseudo terminal STREAM, before the ldterm module is pushed.

On the write-side, the TCSETA, TCSETAF, TCSETAW, TCGETA, TCSETS, TCSETSW, TCSETSF, TCGETS, TCSBRK, JWINSIZE, TIOCGWINSZ, and TIOCSWINSZ termio ioctl(2) messages are processed and acknowledged. A hang up (such as stty 0) is converted to a zero length M_DATA message and passed downstream. Termio cflags and window row and column information are stored locally one per stream. M_DELAY messages are discarded. All other messages are passed downstream unmodified.

On the read-side all messages are passed upstream unmodified with the following exceptions. All M_READ and M_DELAY messages are freed in both directions. An ioctl TCSBRK is converted to an M_BREAK message and passed upstream and an acknowledgement is returned downstream. An ioctl TIOCSIGNAL is converted into an M_PCSIG message, and passed upstream and an acknowledgement is returned downstream.

Finally an ioctl TIOCREMOTE is converted into an M_CTL message, acknowledged, and passed upstream. The argument is a pointer to an int. If the value of the int is non-zero, remote mode is enabled; if the value of the int is zero, remote mode is disabled. This mode can be enabled or disabled independently of packet mode. When a pseudo-terminal is in remote mode, input to the slave device of the pseudo-terminal is flow controlled and not input edited (regardless of the mode of the slave side of the pseudo-terminal). Each write to the master device produces a record boundary for the process reading the slave device. In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an EOF character. This means that a process writing to a pseudo-terminal master in remote mode must keep track of line boundaries, and write only one line at a time to the master. For example, if a process were to buffer up several newline characters and write them to the master with one write, it would appear to a process reading from the slave as if a single line containing several newline characters had been typed (as if, for example, a user had typed the LNEXT character before typing all but the last of those newline characters). Remote mode can be used when doing remote line editing in a window manager, or whenever flow controlled input is required.

FILES

<sys/ptem.h>

SEE ALSO

crash(1M), stty(1), ioctl(2), ldterm(7), pckt(7), pty(7), streamio(7), termio(7).

NAME

pty - pseudo-terminal driver

SYNOPSIS

cc [*flags*] *files* -lsocket -lnsl

```
#include <fcntl.h>
#include <sys/stropts.h>
#include <sys/termios.h>

char *slavename; /* name of slave pseudo-tty */

grantpt(master);            /* change perms of slave */
unlockpt(master);           /* unlock slave */
slavename = ptsname(master);     /* get name of slave */
if ((slave = open(slavename, O_RDWR)) < 0) {
      perror(slavename);
      exit(-1);
}
ioctl(slave, I_PUSH, "ptem");    /* pty hware emul module */
ioctl(slave, I_PUSH, "ldterm");  /* line discipline module */
ioctl(slave, I_PUSH, "ttcompat");     /* BSD/XENIX compat module */
```

DESCRIPTION

The pty driver provides support for a pair of devices collectively known as a pseudo-terminal. The two devices comprising a pseudo-terminal are known as a master and a slave. The slave device distinguishes between the B0 baud rate and other baud rates specified in the c_flag word of the termios structure, and the CLOCAL flag in that word. It does not support any of the other termio(7) device control functions specified by flags in the c_flag word of the termios structure and by the IGNBRK, IGNPAR, PARMRK, or INPCK flags in the c_iflag word of the termios structure, as these functions apply only to asynchronous serial ports. All other termio(7) functions must be performed by STREAMS modules pushed atop the driver; when a slave device is opened, the ldterm(7) and ttcompat(7) STREAMS modules are automatically pushed on top of the stream, providing the standard termio(7) interface.

Instead of having a hardware interface and associated hardware that supports the terminal functions, the functions are implemented by another process manipulating the master device of the pseudo-terminal.

The master and slave devices of the pseudo-terminal are tightly connected. Any data written on the master device is given to the slave device as input, as though it had been received from a hardware interface. Any data written on the slave terminal can be read from the master device (rather than being transmitted from a UART).

In configuring, the default count is given in the system(4) file with the lines:

```
INCLUDE:PTM(256)
INCLUDE:PTS
INCLUDE:PTEM(256)
```

which means that 256 pseudo-terminal pairs are configured. The maximum allowed during installation of the Networking Support Utilities (nsu) package is 1024 pseudo-terminal pairs. For the M88000 architecture, the pty driver supports pseudo-terminal access via the 88/Open Binary Compatability Standard (BCS). BCS pseudo-terminals are configured with INCLUDE:BCSPTS in the system(4) file. The number of BCS pseudo-terminals configured is the same as the number indicated by the PTM entry. If more than 256 pairs are given, BCS pseudo-terminal pairs will be limited to 256 pairs.

ioctls

The standard set of termio ioctl commands are supported by the slave device. None of the bits in the c_cflag word have any effect on the pseudo-terminal, except that if the baud rate is set to B0, it appears to the process on the master device as if the last process on the slave device had closed the line; thus, setting the baud rate to B0 has the effect of "hanging up" the pseudo-terminal, just as it has the effect of hanging up a real terminal.

There is no notion of parity on a pseudo-terminal, so none of the flags in the c_iflag word that control the processing of parity errors have any effect. Similarly, there is no notion of a "break," so none of the flags that control the processing of breaks and none of the ioctls that generate breaks have any effect.

Input flow control is automatically performed; a process that attempts to write to the master device is blocked if too much unconsumed data is buffered on the slave device. The input flow control provided by the IXOFF flag in the c_iflag word is not supported.

The delays specified in the c_oflag word are not supported.

Because pseudo-terminals cannot use modems, the ioctls that return or alter the state of modem control lines are silently ignored.

A few special ioctls are provided on the master devices of pseudo-terminals to provide functionality needed by application programs to emulate real hardware interfaces:

ISPTM A successful return identifies the device as a pseudo-terminal.

UNLKPT Changes the internal state of the corresponding slave pseudo-terminal so that it can be opened.

The ioctls TIOCGWINSZ and TIOCSWINSZ can be performed on the master device of a pseudo-terminal; they have the same effect as when performed on the slave device.

FILES

/dev/ptmx pseudo-terminal master clone device
/dev/pts[0-1023] pseudo-terminal slave devices
/dev/pty[p-za-l][0-9a-f] BCS pseudo-terminal master devices
/dev/tty[p-za-l][0-9a-f] BCS pseudo-terminal slave devices

SEE ALSO

rlogin(1), grantpt(3C), ptsname(3C), unlockpt(3C), ldterm(7), pckt(7), ptem(7), termio(7), ttcompat(7).

NAME

publickey - public key database

SYNOPSIS

/etc/publickey

DESCRIPTION

/etc/publickey is the public key database used for secure RPC. Each entry in the database consists of a network user name (which may either refer to a user or a hostname), followed by the user's public key (in hex notation), a colon, and then the user's secret key encrypted with a password (also in hex notation).

This file is altered either by the user through the chkey(1) command or by the system administrator through the newkey(1) command.

SEE ALSO

chkey(1), newkey(1), publickey(3N)

NAME

resolv.conf - configuration file for name server

SYNOPSIS

/etc/resolv.conf

DESCRIPTION

The resolver is a set of routines in the C library [see resolver(3)] that provide access to the Internet Domain Name System. The resolver configuration file contains information that will be read by the resolver routines at the first instance when they are invoked by a process. The file is designed to be human readable and will contain a list of keywords with values that provide various types of resolver information.

On a normally configured system this file should not be necessary. The only name server to be queried will be on the local machine; then the domain name will be determined from the host name and the domain search path will be constructed from the domain name.

The different configuration options are:

nameserver

The Internet address (in dot notation) of a name server that the resolver should query: Up to MAXNS (currently 3) name servers may be listed, one per keyword. If there are multiple servers, the resolver library will query them in the order listed. If no nameserver entries are present, the default will be to use the name server on the local machine. (The algorithm used is to try a name server; if the query times out, try the next one until you are out of name servers, then repeat trying all the name servers until a maximum number of retries have been performed).

domain

Local domain name: Most queries for names within this domain can use short names relative to the local domain. If no domain entry is present, the domain will be determined from the local host name returned by gethostname(3); the domain part will be taken to be everything after the first '.'. Finally, if the host name does not contain a domain part, the "root domain" will be assumed.

search

Search the list for host name lookup: Normally, the search list will be determined from the local domain name; by default, it will begin with the local domain name, then with successive parent domains that have at least two components in their names. This may be changed by listing the desired domain search path following the search keyword with spaces or tabs separating the names.

Most resolver queries will be attempted using each component of the search path in turn until a match is found.

FILES

/etc/resolv.conf

SEE ALSO

gethostbyname(3N), resolver(3), named(1M).

NOTES

The `search` process may be slow and will generate a lot of network traffic if the servers for the listed domains are not local and that queries will time out if no server is available for one of the domains.

The search list is currently limited to six domains with a total of 256 characters.

The `domain` and `search` keywords are mutually exclusive. If more than one instance of these keywords is present, the last instance will override the earlier one(s).

The keyword and its value must appear on a single line; the keyword (e.g., `nameserver`) must start the line. The value should follow the keyword, separated by white space.

It is possible for `rlogind` and `telnetd` to respond slowly when Domain Name Service is in place and the primary nameserver is unreachable or slow to respond. If your nameserver or network is heavily loaded, you should consider configuring a slave name server on your system. This will allow the nameserver database to be cached locally, doing away with the need for potentially slow resolver requests over the network on each and every login attempt. Four steps must be carried out to set up a slave nameserver:

1) The entry `nameserver 127.1` should be placed at the top of the nameserver list in `/etc/resolv.conf`.
2) The address of the primary nameserver should be listed on the `forwarders` line in `/etc/named.boot`.
3) The nameserver should be placed into slave mode by uncommenting the keyword `slave` in `/etc/named.boot`.
4) The SOA information in `/etc/named.data/localhost.rev` should be filled in according to the comments listed there.

NAME

`rfmaster` - Remote File Sharing name server master file

DESCRIPTION

Each transport provider used by Remote File Sharing has an associated `rfmaster` file that identifies the primary and secondary name servers for that transport provider. The `rfmaster` file ASCII contains a series of records, each terminated by a newline; a record may be extended over more than one line by escaping the newline character with a backslash ("\"). The fields in each record are separated by one or more tabs or spaces. Each record has three fields:

name type data

The *type* field, which defines the meaning of the *name* and *data* fields, has three possible values. These values can appear in upper case or lower case:

`p` The `p` type defines the primary domain name server. For this type, *name* is the domain name and *data* is the full host name of the machine that is the primary name server. The full host name is specified as *domain.nodename.* There can be only one primary name server per domain.

`s` The `s` type defines a secondary name server for a domain. *name* and *data* are the same as for the `p` type. The order of the `s` entries in the `rfmaster` file determines the order in which secondary name servers take over when the current domain name server fails.

`a` The `a` type defines a network address for a machine. *name* is the full domain name for the machine and *data* is the network address of the machine. The network address can be in plain ASCII text or it can be preceded by a `\x` or `\X` to be interpreted as hexadecimal notation. (See the documentation for the particular network you are using to determine the network addresses you need.

If a line in the `rfmaster` file begins with a # character, the entire line is treated as a comment.

There are at least two lines in the `rfmaster` file per domain name server: one `p` and one `a` line, to define the primary and its network address.

This file is created and maintained on the primary domain name server. When a machine other than the primary tries to start Remote File Sharing, this file is read to determine the address of the primary. If the associated `rfmaster` for a transport provider is missing, use `rfstart -p` to identify the primary for that transport provider. After that, a copy of the primary's `rfmaster` file is automatically placed on the machine.

Domains not served by the primary can also be listed in the `rfmaster` file. By adding primary, secondary, and address information for other domains on a network, machines served by the primary will be able to share resources with machines in other domains.

A primary name server may be a primary for more than one domain. However, the secondaries must then also be the same for each domain served by the primary. There is an `rfmaster` file for each transport provider.

EXAMPLES

An example of an `rfmaster` file is shown below. (The network address examples, `comp1.serve` and `comp2.serve`, are TCP/IP network addresses.)

```
rfsdomain  P     rfsdomain.pri_nameserve
rfsdomain.pri_nameserve     A     \x00020ace980a011f0000000000000000
```

FILES

`/etc/rfs/`<*transport*>`/rfmaster`

SEE ALSO

`rfstart`(1M).

NAME

routing - system support for packet network routing

SYNOPSIS

#include <net/route.h>

DESCRIPTION

The network facilities provide general packet routing. Routing table maintenance may be implemented in applications processes.

A simple set of data structures compose a routing table used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. The routing table was designed to support routing for the Internet Protocol (IP), but its implementation is protocol independent and thus it may serve other protocols as well. User programs may manipulate this data base with the aid of two ioctl(2) commands, SIOCADDRT and SIOCDELRT. These commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by privileged user.

A routing table entry has the following form, as defined in /usr/include/net/route.h:

```
struct rtentry {
    u_long  rt_hash;                  /* to speed lookups */
    struct  sockaddr rt_dst;          /* key */
    struct  sockaddr rt_gateway;      /* value */
    short   rt_flags;                 /* up/down?, host/net */
    short   rt_refcnt;                /* # held references */
    u_long  rt_use;                   /* raw # packets forwarded */
#ifdef STRNET
    struct  ip_provider *rt_prov;     /* the answer: provider to use */
#else
    struct  ifnet *rt_ifp;            /* the answer: interface to use */
#endif /* STRNET */
    int rt_metric;                    /* metric for route provider */
    int rt_proto;                     /* protocol route was learned */
    time_t rt_age;                    /* time of last update */
    rwlock_t *rt_lck;                 /* ptr to rthost_lck or rtnet_lck */
};
```

with *rt_flags* defined from:

```
#define   RTF_UP        0x1           /* route usable */
#define   RTF_GATEWAY   0x2           /* destination is a gateway */
#define   RTF_HOST      0x4           /* host entry (net otherwise) */
```

Routing table entries come in three flavors: for a specific host, for all hosts on a specific network, for any destination not matched by entries of the first two types (a wildcard route). Each network interface installs a routing table entry when it it is initialized. Normally the interface specifies the route through it is a direct connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface may be requested to address the packet to an entity different from the eventual recipient (that is, the packet is forwarded).

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (rt_refcnt is non-zero), the resources associated with it will not be reclaimed until all references to it are removed.

User processes read the routing tables through the /dev/kmem device.

The *rt_use* field contains the number of packets sent along the route. This value is used to select among multiple routes to the same destination. When multiple routes to the same destination exist, the least used route is selected.

A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

FILES

/dev/kmem

DIAGNOSTICS

EEXIST	A request was made to duplicate an existing entry.
ESRCH	A request was made to delete a non-existent entry.
ENOBUFS	Insufficient resources were available to install a new route.

SEE ALSO

route(1M), routed(1M), ioctl(2).

NAME

rpc - rpc program number data base

SYNOPSIS

rpc

DESCRIPTION

The rpc program number database contains user readable names that can be used in place of RPC program numbers. Each line has the following information:

name of server for the RPC program
RPC program number
aliases

Items are separated by any number of blanks and/or tab characters. A # indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Below is an example of an RPC database:

```
#
#            rpc
#

rpcbind      100000        portmap sunrpc portmapper
rusersd      100002        rusers
nfs          100003        nfsprog
mountd       100005        mount showmount
walld        100008        rwall shutdown
sprayd       100012        spray
llockmgr     100020
nlockmgr     100021
status       100024
bootparam    100026
keyserv      100029        keyserver
```

NAME

rt_dptbl - real-time dispatcher parameter table

DESCRIPTION

The process scheduler (or dispatcher) is the portion of the kernel that controls allocation of the CPU to processes. The scheduler supports the notion of scheduling classes where each class defines a scheduling policy, used to schedule processes within that class. Associated with each scheduling class is a set of priority queues on which ready to run processes are linked. These priority queues are mapped by the system configuration into a set of global scheduling priorities which are available to processes within the class. (The dispatcher always selects for execution the process with the highest global scheduling priority in the system.) The priority queues associated with a given class are viewed by that class as a contiguous set of priority levels numbered from 0 (lowest priority) to *n* (highest priority—a configuration dependent value). The set of global scheduling priorities that the queues for a given class are mapped into might not start at zero and might not be contiguous (depending on the configuration).

The real-time class maintains an in-core table, with an entry for each priority level, giving the properties of that level. This table is called the real-time dispatcher parameter table (rt_dptbl). The rt_dptbl consists of an array of parameter structures (struct rt_dpent), one for each of the *n* priority levels. The properties of a given priority level *i* are specified by the *i*th parameter structure in this array (rt_dptbl*i*).

A parameter structure consists of the following members. These are also described in the /usr/include/sys/rt.h header file.

rt_globpri — The global scheduling priority associated with this priority level. The mapping between real-time priority levels and global scheduling priorities is determined at boot time by the system configuration. The rt_globpri values cannot be changed with dispadmin(1M).

rt_quantum — The length of the time quantum allocated to processes at this level in ticks (HZ). The time quantum value is only a default or starting value for processes at a particular level as the time quantum of a real-time process can be changed by the user with the priocntl command or the priocntl system call.

An administrator can affect the behavior of the real-time portion of the scheduler by reconfiguring the rt_dptbl. There are two methods available for doing this.

MASTER FILE

The rt_dptbl can be reconfigured at boot time by specifying the desired values in the rt master file and reconfiguring the system using the auto-configuration boot procedure; see mkboot(1M) and master(4). This is the only method that can be used to change the number of real-time priority levels or the set of global scheduling priorities used by the real-time class.

DISPADMIN CONFIGURATION FILE

The rt_quantum values in the rt_dptbl can be examined and modified on a running system using the dispadmin(1M) command. Invoking dispadmin for the real-time class allows the administrator to retrieve the current rt_dptbl configuration from the kernel's in-core table, or overwrite the in-core table with values from a configuration file. The configuration file used for input to `dispadmin` must conform to the specific format described below.

Blank lines are ignored and any part of a line to the right of a # symbol is treated as a comment. The first non-blank, non-comment line must indicate the resolution to be used for interpreting the time quantum values. The resolution is specified as

`RES=`*res*

where *res* is a positive integer between 1 and 1,000,000,000 inclusive and the resolution used is the reciprocal of *res* in seconds. (For example, `RES=1000` specifies millisecond resolution.) Although very fine (nanosecond) resolution may be specified, the time quantum lengths are rounded up to the next integral multiple of the system clock's resolution.

The remaining lines in the file are used to specify the `rt_quantum` values for each of the real-time priority levels. The first line specifies the quantum for real-time level 0, the second line specifies the quantum for real-time level 1, etc. There must be exactly one line for each configured real-time priority level. Each `rt_quantum` entry must be either a positive integer specifying the desired time quantum (in the resolution given by *res*), or the symbol `RT_TQINF` indicating an infinite time quantum for that level.

EXAMPLE

The following excerpt from a `dispadmin` configuration file illustrates the format. Note that for each line specifying a time quantum there is a comment indicating the corresponding priority level. These level numbers indicate priority within the real-time class, and the mapping between these real-time priorities and the corresponding global scheduling priorities is determined by the configuration specified in the `rt` master file. The level numbers are strictly for the convenience of the administrator reading the file and, as with any comment, they are ignored by `dispadmin` on input. `dispadmin` assumes that the lines in the file are ordered by consecutive, increasing priority level (from 0 to the maximum configured real-time priority). The level numbers in the comments should normally agree with this ordering; if for some reason they don't, however, `dispadmin` is unaffected.

```
# Real-Time Dispatcher Configuration File
RES=1000

#     TIME QUANTUM                    PRIORITY
#     (rt_quantum)                      LEVEL
           100                   #        0
           100                   #        1
           100                   #        2
           100                   #        3
           100                   #        4
           100                   #        5
            90                   #        6
            90                   #        7
           .                     .        .
           .                     .        .
           .                     .        .
            10                   #       58
            10                   #       59
```

FILES

/usr/include/sys/rt.h

SEE ALSO

dispadmin(1M), priocntl(1), priocntl(2), master(4), mkboot(1M).

NAME

SA - devices administered by System Administration

DESCRIPTION

The files in the directories /dev/SA (for block devices) and the /dev/rSA (for raw devices) are used by System Administration to access the devices on which it operates. For devices that support more than one slice (like disks) the /dev/(r)SA entry is linked to the slice that spans the entire device. Not all /dev/(r)SA entries are used by all System Administration commands.

FILES

/dev/SA
/dev/rSA

SEE ALSO

sysadm(1)

NAME

sad - STREAMS Administrative Driver

SYNOPSIS

```
#include <sys/types.h>
#include <sys/conf.h>
#include <sys/sad.h>
#include <sys/stropts.h>

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

DESCRIPTION

The STREAMS Administrative Driver provides an interface for applications to perform administrative operations on STREAMS modules and drivers. The interface is provided through `ioctl`(2) commands. Privileged operations may access the `sad` driver via `/dev/sad/admin`. Unprivileged operations may access the `sad` driver via `/dev/sad/user`.

fildes is an open file descriptor that refers to the `sad` driver. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

COMMAND FUNCTIONS

The autopush facility [see `autopush`(1M)] allows one to configure a list of modules to be automatically pushed on a stream when a driver is first opened. Autopush is controlled by the next commands.

`SAD_SAP` Allows the administrator to configure the autopush information for the given device. *arg* points to a `strapush` structure which contains the following members:

```
uint   sap_cmd;
long   sap_major;
long   sap_minor;
long   sap_lastminor;
long   sap_npush;
uint   sap_list[MAXAPUSH] [FMNAMESZ + 1];
```

The `sap_cmd` field indicates the type of configuration being done. It may take on one of the following values:

`SAP_ONE` Configure one minor device of a driver.

`SAP_RANGE` Configure a range of minor devices of a driver.

`SAP_ALL` Configure all minor devices of a driver.

`SAP_CLEAR` Undo configuration information for a driver.

The `sap_major` field is the major device number of the device to be configured. The `sap_minor` field is the minor device number of the device to be configured. The `sap_lastminor` field is used only with the `SAP_RANGE` command, with which a range of minor devices between `sap_minor` and `sap_lastminor`, inclusive, are to be configured. The minor fields have no meaning for the SAP_ALL command. The sap_npush field indicates the number of modules to be automatically pushed when the device is opened. It must be less than or equal to MAXAPUSH, defined in sad.h. It must also be less than or equal to NSTRPUSH, the maximum number of modules that can be pushed on a stream, defined in the kernel master file. The field sap_list is an array of module names to be pushed in the order in which they appear in the list.

When using the SAP_CLEAR command, the user sets only sap_major and sap_minor. This will undo the configuration information for any of the other commands. If a previous entry was configured as SAP_ALL, sap_minor should be set to zero. If a previous entry was configured as SAP_RANGE, sap_minor should be set to the lowest minor device number in the range configured.

On failure, errno is set to the following value:

EFAULT — *arg* points outside the allocated address space.

EINVAL — The major device number is invalid, the number of modules is invalid, or the list of module names is invalid.

ENOSTR — The major device number does not represent a STREAMS driver.

EEXIST — The major-minor device pair is already configured.

ERANGE — The command is SAP_RANGE and sap_lastminor is not greater than sap_minor, or the command is SAP_CLEAR and sap_minor is not equal to the first minor in the range.

ENODEV — The command is SAP_CLEAR and the device is not configured for autopush.

ENOSR — An internal autopush data structure cannot be allocated.

SAD_GAP — Allows any user to query the sad driver to get the autopush configuration information for a given device. *arg* points to a strapush structure as described in the previous command.

The user should set the sap_major and sap_minor fields of the strapush structure to the major and minor device numbers, respectively, of the device in question. On return, the strapush structure will be filled in with the entire information used to configure the device. Unused entries in the module list will be zero-filled.

On failure, errno is set to one of the following values:

EFAULT — *arg* points outside the allocated address space.

EINVAL — The major device number is invalid.

ENOSTR The major device number does not represent a STREAMS driver.

ENODEV The device is not configured for autopush.

SAD_VML Allows any user to validate a list of modules (such as, to see if they are installed on the system.) *arg* is a pointer to a str_list structure with the following members:

```
int                 sl_nmods;
struct str_mlist    *sl_modlist;
```

The str_mlist structure has the following member:

```
char                l_name[FMNAMESZ+1];
```

sl_nmods indicates the number of entries the user has allocated in the array and sl_modlist points to the array of module names. The return value is 0 if the list is valid, 1 if the list contains an invalid module name, or -1 on failure. On failure, errno is set to one of the following values:

EFAULT *arg* points outside the allocated address space.

EINVAL The sl_nmods field of the str_list structure is less than or equal to zero.

SEE ALSO

intro(2), ioctl(2), open(2).

DIAGNOSTICS

Unless specified otherwise above, the return value from ioctl is 0 upon success and -1 upon failure with errno set as indicated.

NAME

`sccsfile` - format of SCCS file

DESCRIPTION

An SCCS (Source Code Control System) file is an ASCII file. It consists of six logical parts: the checksum, the delta table (contains information about each delta), user names (contains login names and/or numerical group IDs of users who may add deltas), flags (contains definitions of internal keywords), comments (contains arbitrary descriptive information about the file), and the body (contains the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as the control character and will be represented graphically as `@`. Any line described below that is not depicted as beginning with the control character is prevented from beginning with the control character.

Entries of the form *DDDDD* represent a five-digit string (a number between 00000 and 99999).

Each logical part of an SCCS file is described in detail below.

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

`@h`*DDDDD*

The value of the checksum is the sum of all characters, except those of the first line. The `@h` provides a magic number of (octal) 064001, depending on byte order.

Delta Table

The delta table consists of a variable number of entries of one of the following forms:

`@s` *DDDDD/DDDDD/DDDDD*
`@d` *<type> <SCCS ID> yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD*
`@i` *DDDDD* . . .
`@x` *DDDDD* . . .
`@g` *DDDDD* . . .
`@m` <MR *number*>
. . .
`@c` *<comments>* . . .
. . .
`@e`

The first line (`@s`) contains the number of lines inserted/deleted/unchanged, respectively. The second line (`@d`) contains the type of the delta (normal: `D` or removed: `R`), the SCCS ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The `@i`, `@x`, and `@g` lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

The @m lines (optional) each contain one MR number associated with the delta; the @c lines contain comments associated with the delta. The @e line ends the delta table entry.

User Names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines @u and @U. An empty list allows anyone to make a delta. Any line starting with a ! prohibits the succeeding group or user from making deltas.

Flags

Keywords used internally. See admin(1) for more information on their use. Each flag line takes the form:

@f *<flag>* *<optional text>*

The following flags are defined:

@f t *<type of program>*
@f v *<program name>*
@f i *<keyword string>*
@f b
@f m *<module name>*
@f f *<floor>*
@f c *<ceiling>*
@f d *<default-sid>*
@f n
@f j
@f l *<lock-releases>*
@f q *<user defined>*
@f z *<reserved for use in interfaces>*

The t flag defines the replacement for the %Y% identification keyword. The v flag controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program. The i flag controls the warning/error aspect of the "No id keywords" message. When the i flag is not present, this message is only a warning; when the i flag is present, this message causes a fatal error (the file will not be "gotten", or the delta will not be made). When the b flag is present the -b keyletter may be used on the get command to cause a branch in the delta tree. The m flag defines the first choice for the replacement text of the %M% identification keyword. The f flag defines the floor release; the release below which no deltas may be added. The c flag defines the ceiling release; the release above which no deltas may be added. The d flag defines the default SID to be used when none is specified on a get command. The n flag causes delta to insert a null delta (a delta that applies no changes) in those releases that are skipped when a delta is made in a new release (for example, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the n flag causes skipped releases to be completely empty. The j flag causes get to allow concurrent edits of the same base SID. The l flag defines a *list* of releases that are locked against editing. The q flag defines the replacement for the `%Q%` identification keyword. The `z` flag is used in specialized interface programs.

Comments

Arbitrary text is surrounded by the bracketing lines `@t` and `@T`. The comments section typically will contain a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines do not begin with the control character, control lines do. There are three kinds of control lines: insert, delete, and end, represented by:

`@I` *DDDDD*
`@D` *DDDDD*
`@E` *DDDDD*

respectively. The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

`admin`(1), `delta`(1), `get`(1), `prs`(1)

NAME

scsi1x7 - SCSI1x7 SCSI host adapter

DESCRIPTION

The SCSI1x7 driver controls a SCSI host adapter with one SCSI bus, supporting up to seven SCSI devices. Each SCSI device can have up to eight sub devices.

Assuming the necessary system resources are available, the SCSI1x7 driver sends each command to the controller as soon as it receives the command from an application.

The SCSI1x7 driver does not have to wait for a command to complete before sending a command for another device.

SUPPORT DEVICES

Disk Drives

Disk drives currently supported are:

DESCRIPTION	ddefs(1M) FILE	TYPE
150MB CDC 94161 Wren III	mcdcIII	Hard
300MB CDC 94171 Wren IV	mcdcIV	Hard
600MB CDC 94181 Wren V	mcdcV	Hard
1.2GB CDC 94601 Wren VII	mcdcVII	Hard
135MB FUJITSU M2613S	mfuj2613	Hard
180MB FUJITSU M2614S	mfuj2614	Hard
330MB FUJITSU M2622S	mfuj2622	Hard
525MB FUJITSU M2624S	mfuj2624	Hard
1.75GB FUJITSU M2652S	mfuj2652	Hard
Toshiba XM3201B CDROM	none	CDROM
1.2MB TEAC 5¼ inch FC-1	see next table	Floppy
2.88MB TEAC 3½ inch FC-1	see next table	Floppy

Note that in all tables, each entry in the ddefs(1M) FILE column is the name of a file that defines the characteristics of the disk in the /etc/dskdefs directory. Each entry in the BLOCKS column is the number of specified blocks when making a file system with mkfs(1M).

The types of floppy diskettes currently supported are listed in the following two tables.

5¼ INCH DISKETTES				
DESCRIPTION	ddefs(1M) FILE	BLOCKS	MEDIA TYPE	SLICE
Double density Motorola format	mdsdd5	1276	MFD-2DD	0
Single density PC/XT 8 sect./track	mpcxt8	640	MFD-2DD	12
Single density PC/XT 9 sect./track	mpcxt9	720	MFD-2DD	9
Double density PC/AT	mpcat	2400	MF2-HD	8

3½ INCH DISKETTES				
DESCRIPTION	ddefs(1M) FILE	BLOCKS	MEDIA TYPE	SLICE
Double density PC/XT 9 sect./track	mpcxt9_3	1440	MFD-2DD	13
Double density PS/2	mps2	2880	MF2-HD	10
Super High Density (2.88MB formatted)	mshd	5760	PMF2-ED	11

Tape Drives

Tape drives currently supported by the SCSI1x7 host adapter are:

DESCRIPTION	FORMAT	TYPE
Archive 2150S	QIC24, QIC120, QIC150	Streaming
Archive 2525	QIC24, QIC120, QIC150	Streaming
Archive Python	DAT	Streaming
Exabyte EXB-8200	8mm	Streaming
Kennedy 9660	9-track	Start/Stop
M4 Data 9914	9-track	Start/Stop

MINOR NUMBERS

The SCSI1x7 device driver interprets the minor number of a device using the standard SCSI-2 minor mapping.

DISK SUPPORT

During system initialization, the SCSI1x7 device driver will spin-up any disks that are strapped to spin-up.

The hard disk drives supported by the SCSI1x7 handle all defects internally. A list of known defective locations is recorded on the medium. During format, any data that would normally be loaded into these locations are automatically assigned alternate locations. Also during format, the drive is checked for defects in addition to those on the known list. If any additional defective locations are found, any data that would be stored there are assigned alternate locations.

The SCSI1x7 device driver complies with the disk support standard specified on the `disk`(7) man page with the following `ioctl` command exceptions.

`DKGETCFG ioctl` command
: The disk is accessed in order to set the parameters associated with the disk. The driver does not keep this information internally.

`DKGETINFO ioctl` command
: The disk is accessed in order to set the parameters associated with the disk. The driver does not keep this information internally.

`DKSETCFG ioctl` command
: The disk is accessed in order to set the parameters associated with the disk. The driver does not keep this information internally.

`DKSETINFO ioctl` command
: The disk is accessed in order to set the parameters associated with the disk. The driver does not keep this information internally.

DKFORMAT ioctl command

The scsiformat command is used to format the device. By turning on a bit in the controller attribute word of the disk definition file passed to dinit, the drive can be told to ignore the grown defect list on the disk. See the description of the controller attribute word on the disk(7) manual page for more information.

TAPE SUPPORT

The SCSI1x7 device driver complies with the tape support standard specified on the tape(7) manual page with no exceptions.

FLOPPY DISK SUPPORT

The SCSI1x7 supported floppy drives provide level one support as defined by the *88open PC Floppy Emulation Supplement* to the *Binary Compatibility Standard*.

The SCSI1x7 device driver complies with the floppy disk support standard specified on the floppy(7) manual page with the following exceptions:

DKFIXBADSPOT ioctl command

This command is not supported; it returns an EINVAL error.

DKGETCFG ioctl command

This command performs no operation; it returns with no effect and no error.

DKGETINFO ioctl command

This command performs no operation; it returns with no effect and no error.

DKSETCFG ioctl command

This command performs no operation; it returns with no effect and no error.

DKSETINFO ioctl command

This command performs no operation; it returns with no effect and no error.

DKGETSLC ioctl command

This command performs no operation; it returns with no effect and no error.

DKSETSLC ioctl command

This command performs no operation; it returns with no effect and no error.

FL_PC_LEVEL ioctl command

The SCSI1x7 driver currently only supports level 1, so the integer pointed to by *arg* is always set to 1 by this call.

Slicing

Floppy diskettes do not have volume ID blocks or Volume Table of Contents (VTOC). A floppy drive can be thought of as a hard disk with a single slice. The slice bits of the *minor number* select the drive geometry as described later in this manual page.

V_PDREAD ioctl command

This command always returns EINVAL.

V_PDWRITE ioctl command

This command always returns EINVAL.

V_RVTOC ioctl command

This command always returns EINVAL.

V_WVTOC ioctl command
This command always returns EINVAL.

dinit/ddef
The ddef files for floppy disks are treated as placeholders. Although they are required for dinit(1M) to work, the information is not used. The format of the diskette is determined via the slice number of the device. See the supported floppy tables at the beginning of this manual page for more information.

CDROM SUPPORT

The SCSI1x7 device driver will not spin-up CDROM devices at system initialization time.

The SCSI1x7 device driver complies with the CDROM support standard specified on the cdrom(7) man page with the following exceptions:

DKGETCFG ioctl command
The disk is accessed in order to get the parameters associated with the disk. The driver does not keep this information internally.

PASSTHRU SUPPORT

The SCSI1X7 device driver complies with the passthru support standard specified on the passthru(7) man page with no exceptions.

ERROR MESSAGES

The SCSI1X7 driver prints error messages to the system console. The SCSI1X7 driver can generate several different error messages. These error messages attempt to provide enough information to permit the operator to diagnose the problem. Some of these messages print a unit number to indicate which device was being accessed at the time of the error. The following table can help to interpret the unit number.

BUS	DEVICE	LUN	UNIT #
0	0	0-7	0-7
0	1	0-7	8-15
0	2	0-7	16-23
0	3	0-7	24-31
0	4	0-7	32-39
0	5	0-7	40-47
0	6	0-7	48-55
0	7	0-7	56-63

Most error messages start with a line that prints out the drive, controller, and slice that has the error. If the error is non-recoverable (fatal), the following is the first line of the error message for disks:

`ERROR on` *device* `at MVME187 SCSI bus address` *x*`, slice` *y*

For tapes, the following is the first line:

`FATAL ERROR on MVME187 SCSI ctl` *x*`, Tape drive` *y*

The following is the next line of the error message:

`MVME187 SCSI error on` *device* `at SCSI address` *x*

where *device* is one of disk, floppy, tape, or CDROM.

The next line of the error message gives the SCSI Driver Library command that encountered the error. It is of the form:

`SDL` *cmd* `command failed`

There will be up to one additional line describing each of the four types of error codes described above: SCSI Sense Key, SCSI status, SIOP status, and SDL status. If any of these status codes indicate a non-error status, its line will be printed.

The following error messages are associated with streaming tape:

`Controller timeout`
: The MVME187 controller timed out while executing a command. This usually means that the SCSI controller attached to the MVME187 could not be accessed. Check cables and power.

`Tape not ready`
: There may be a problem with the streaming tape cartridge. Check to see whether the cartridge is defective or not in place.

`End of media`
: During write operation, ran off the end of the tape. The last file written to the tape is incomplete and needs to be written to another tape.

`End of data`
: During read operation, tried to read past the last filemark on the tape.

`Write protected`
: Attempted to write to a write protected tape. Remove the tape cartridge from the drive and check the cartridge.

`Illegal request`
: Attempted to execute commands that make no logical sense such as trying to erase the tape beginning in the middle.

Other error codes may indicate serious defects. Report the error code to Motorola Field Service Division/Customer Support.

Miscellaneous Error Messages

`Timeout on` *device* `at MVME187 SCSI bus address` *x*`, slice` *y*
: A request sent to SCSI bus address *x*, drive *y* was not returned to the driver within the allotted time. This could indicate a software or hardware problem that needs further attention.

Other error codes may indicate serious defects. Report them to Motorola Field Service Division/Customer Support.

There are four types of error codes returned by the MVME187 driver: SCSI Sense Key, SCSI status, SIOP status, and SDL status.

SCSI Sense Keys	
SCSI Sense	Description
0x00	No sense data available.
0x01	Recovered error: command was successfully retried.
0x02	Not ready: device had not spun up before command was issued.
0x03	Medium error, bad spot: incorrect or unformatted media used.
0x04	Hardware error: controller reporting a hardware problem.
0x05	Illegal request: command issued has illegal parameter.
0x06	Unit attention: removable media changed.
0x07	Data protect: device is write protected.
0x08	Blank check: blank spot encountered on tape.
0x09	Vendor-specfic error.
0x0A	Copy aborted.
0x0B	Aborted command.
0x0C	Equal.
0x0D	Volume overflow.
0x0E	Miscompare.
0x20	Illegal length.
0x40	End of Media: encountered end of tape media.
0x80	File Mark: encountered a tape file mark.

SCSI Status Byte Values	
SCSI Status	Explanation
0x00	Good completion.
0x02	Check condition.
0x04	Condition met good.
0x08	Busy.
0x10	Intermediate good.
0x14	Intermediate condition met good.
0x18	Reservation conflict.
0x22	Command terminated.
0x28	Queue full.

SIOP Status Values	
SIOP Status	Explanation
0x00	Good status.
0x01	No operation bits were set.
0x02	Command aborted due to SCSI bus reset.
0x03	Command aborted due to SCSI device reset.
0x04	Command aborted due to abort message.
0x05	Command aborted due to abort tag message.
0x06	Command aborted due to clear queue message.
0x07	Data overflow - too much data from device.
0x08	Data underrun - not enough data from device.
0x09	Clock faster than 50 MHz.
0x0A	Bad clock parameter.
0x0B	Queue depth too large.
0x0C	Selection timeout - device did not respond.
0x0D	Reselection timeout - device did not respond.
0x0E	Bus error during data phase.
0x0F	Bus error during non-data phase.
0x10	Illegal NCR script.
0x11	Command aborted due to unexpected disconnect.
0x12	Command aborted due to unexpected phase change.
0x13	SCSI bus hang during command.
0x14	Data phase not expected by user.
0x15	Data phase in wrong direction.
0x16	Incorrect phase following select.
0x17	Incorrect phase following msg-out.
0x18	Incorrect phase following data.
0x19	Incorrect phase following command.
0x1A	Incorrect phase following status.
0x1B	Incorrect phase following rptr message.
0x1C	Incorrect phase following sdptr message.
0x1D	No identify message after re-selection.
0x1E	SIOP failed during script patching.

SDL Status Values	
SDL Status	Explanation
0x00	Good status.
0x01	Early termination with good status.
0x02	Check condition on request sense command.
0x03	Illegal retry condition.
0x04	Unsupported status code.
0x05	Undefined sense key.
0x06	Illegal mode parameter page.
0x07	Attempted access with block mismatch.
0x08	Maximum block size of CDB exceeded.
0x09	Unsupported build function.
0x0A	Insufficient inquiry data count.
0x0B	Logical unit has not been attached.
0x0C	Variable block size maximum transfer count exceeded.

Refer to the ANSI SCSI specification for a complete list of SCSI command codes and sense keys.

MASTER.D PARAMETERS

The following parameters affect the operation of the SCSI1x7 device driver. The following are parameters listed under the SCSI1x7 description:

`scsi_host_address`
This parameter specifies the SCSI bus address occupied by the host (ncr53c710) SCSI chip.

`scsi_bus_reset_delay`
This parameter specifies the delay after a SCSI bus reset before issuing commands to any device.

`sd_max_cmd_queue_size`
This parameter specifies the number of `sdl_cmd` structures allocated per device. It places an upper limit on the number of simultaneous commands sent to the SCSI Driver Library for a disk device.

`sd_default_cmd_queue_size`
This parameter specifies the default maximum number of simultaneous commands sent to the SCSI Driver Library for a disk device.

`scsi_tape_maxbsize`
This parameter specifies the maximum double buffer size for tape transfers.

`scsi_len_sglists`
This parameter specifies the number of entries in the scatter/gather lists The maximum transfer size to the device is `scsi_len_sglists` pages when going through the raw I/O interface.

`scsi_rescan`
This parameter determines if a rescan of the device will be done at open time. If a 0, no rescan will be done; otherwise a rescan will be done.

scsi_max_spl
: This parameter sets the maximum number of concurrent special commands. The default value is 8. Special commands are all SCSI commands except reads or writes. Most ioctl() commands are special commands, and special commands are used during open() and close() processing. If this number is too low, some processes will sleep waiting for resources when doing special commands.

scsi_starvsize
: This parameter specifies the maximum length of a disk, floppy, or CDROM I/O queue that will be sorted before beginning another queue.

scsi_spdkeepsize
: This parameter specifies the maximum number of SCSI private areas that the driver keeps for each hard and cdrom disk device. Each SCSI private area is currently 7 bytes. This parameter is used to keep the driver from deadlocking the system when there is no free memory available.

SPECIAL CONSIDERATIONS

When an error occurs while writing or reading a tape, the best course of action in this case is to rewind the tape and repeat the operation.

Removing a cartridge tape during an MTBSF operation hangs the tape drive.

The longest I/O operation which SCSI1x7 host adapters can allow to occur on a tape device operating in variable mode depends on the master.d parameter scsi_len_sglists.

FILES

/dev/dsk/m187_*
/dev/rdsk/m187_*
/dev/rmt/m187_*
/dev/generic/m187_*
/etc/dskdefs/m*
/usr/include/sys/dk.h
/usr/include/sys/mtio.h
/usr/include/sys/dsk.h
/usr/include/sys/scsi.h
/usr/include/sys/scsi_cdisk.h
/usr/include/sys/scsi_disk.h
/usr/include/sys/scsi_fdisk.h
/usr/include/sys/scsi_hdisk.h
/usr/include/sys/scsi_tape.h
/usr/include/sys/scsi_space.h
/usr/include/sys/scd_space.h
/usr/include/sys/sfd_space.h
/usr/include/sys/shd_space.h
/usr/include/sys/sot_space.h
/usr/include/sys/st_space.h
/usr/include/sys/pcflio.h
/usr/include/sys/scsi/sbc_scsi/incl/ncr.h
/usr/include/sys/scsi/sbc_scsi/incl/ncr710.h
/usr/include/sys/scsi/sbc_scsi/incl/ncr710db.h
/usr/include/sys/scsi/sbc_scsi/incl/scsi.h

/usr/include/sys/scsi/sbc_scsi/incl/scsi_dbg.h
/usr/include/sys/scsi/sbc_scsi/incl/scsi_err.h
/usr/include/sys/scsi/sbc_scsi/incl/sdl.h
/usr/include/sys/scsi/sbc_scsi/incl/sdl_cnfg.h
/usr/include/sys/scsi/sbc_scsi/incl/sdldb.h

SEE ALSO

mt(1), ddefs(1M), dinit(1M), close(2), ioctl(2), open(2), read(2), write(2), cdrom(7), disk(7), floppy(7), intro(7), tape(7) passthru(7)

NAME

services - Internet services and aliases

DESCRIPTION

The services file contains an entry for each service available through the DARPA Internet. Each entry consists of a line of the form:

service-name port / protocol aliases

service-name	This is the official Internet service name.
port / protocol	This field is composed of the port number and protocol through which the service is provided (for instance, 512/tcp).
aliases	This is a list of alternate names by which the service might be requested.

Fields can be separated by any number of SPACE and/or TAB characters. A '#' (pound-sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Service names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

FILES

/etc/services

SEE ALSO

getservent(3N), inetd.conf(4)

NOTES

A name server should be used instead of a static file.

NAME

shadow - shadow password file

DESCRIPTION

/etc/shadow is an access-restricted ASCII system file. The fields for each user entry are separated by colons. Each user is separated from the next by a new-line. Unlike the /etc/passwd file, /etc/shadow does not have general read permission.

Here are the fields in /etc/shadow:

username	The user's login name (ID).
password	A 13-character encrypted password for the user, a *lock* string to indicate that the login is not accessible, or no string to show that there is no password for the login.
lastchanged	The number of days between January 1, 1970, and the date that the password was last modified.
minimum	The minimum number of days required between password changes.
maximum	The maximum number of days the password is valid.
warn	The number of days before password expires that the user is warned.
inactive	The number of days of inactivity allowed for that user.
expire	An absolute date specifying when the login may no longer be used.
flag	Reserved for future use, set to zero. Currently not used.

The encrypted password consists of 13 characters chosen from a 64-character alphabet (., /, 0-9, A-Z, a-z).

To update this file, use the passwd, useradd, usermod, or userdel commands.

FILES

/etc/shadow

NOTES

If the /etc/passwd file contains any + entries, similar entries should also exist in this file in order to allow logins for users in the NIS database.

SEE ALSO

login(1), passwd(1), useradd(1M), usermod(1M), userdel(1M), getspent(3C), putspent(3C), and passwd(4).

NAME

sharetab - shared file system table

DESCRIPTION

sharetab resides in directory /etc/dfs and contains a table of local resources shared by the share command.

Each line of the file consists of the following fields:

pathname resource fstype specific_options description

where

pathname	Indicates the pathname of the shared resource.
resource	Indicates the symbolic name by which remote systems can access the resource.
fstype	Indicates the file system type of the shared resource.
specific_options	Indicates file-system-type-specific options that were given to the share command when the resource was shared.
description	Is a description of the shared resource provided by the system administrator when the resource was shared.

SEE ALSO

share(1M)

NAME

SLIP - Serial Line IP (SLIP) Protocol

DESCRIPTION

The Serial Line IP (SLIP) protocol is a very simple protocol which allows two machines to communicate via TCP/IP over a serial line. This protocol simply defines the octets necessary for framing and escaping octets in an IP packet. At the sending end, all octets in the IP packet that should be preceded by an "escape" character, will be "escaped" before sending this packet; this packet transmission will end with a FRAME_END octet. At the receiving end, the octets will be gathered and any "escaped" octets will be transposed (as necessary), until a FRAME_END is received for this packet; then the resulting packet will be passed up to IP.

IOCTLS

The following *ioctl* calls can be used to adjust the behavior of the SLIP module.

NOTE: The S_MTU *ioctl* is the only *ioctl* call which needs a parameter value, an integer.

S_COMPRESSON	Turn on TCP/IP header compression.
S_COMPRESSOFF	Turn off TCP/IP header compression.
S_COMPRESSAON	Turn on automatic detection of TCP/IP header compression (start using compression when peer system does).
S_COMPRESSAOFF	Turn off automatic detection of TCP/IP header compression.
S_NOICMP	Don't allow ICMP packets out on the wire.
S_ICMP	Allow ICMP packets out on the wire.
S_MTU	Set the "maximum transmission unit" (MTU) value for this interface. This request requires an integer as a parameter value to indicate the new MTU size.

SEE ALSO

slattach(1M)
RFC 1144

NAME

snmpd.comm - SNMP communities file

SYNOPSIS

/etc/snmp.d/snmpd.comm

DESCRIPTION

/etc/snmp.d/snmpd.comm contains the definitions for the communities which will be supported by the SNMP agent/server daemon, snmpd(1M). The file contains lines which consist of three items: a session or community name, an IP address in dot notation, and the priviledges to be associated with that communitiy and IP address pair. The priviledges should be one of READ, WRITE, or NONE. NONE is used to lock out specific communities or hosts. Lines which begin with '#' are ignored.

EXAMPLE

```
test1 128.212.64.99 READ
test2 128.212.64.15 WRITE
test3 128.212.64.15 READ
test4 0.0.0.0 READ
public 0.0.0.0 READ
interop 0.0.0.0 READ
```

FILES

/etc/snmp.d/snmpd.comm

SEE ALSO

snmpd(1M)
RFC 1066, RFC 1067

NAME

snmpd.conf - SNMP configuration file

SYNOPSIS

/etc/snmp.d/snmpd.conf

DESCRIPTION

/etc/snmp.d/snmpd.conf is used to configure some portions of the MIB being supported by snmpd(1M). The file contains lines which consist of a keyword and a value to be associated with the MIB element corresponding to that keyword. The keywords are treated as case insensitive. Lines which begin with '#' are ignored.

Currently, two initializers are supported. They are used to initialize the sysDescr and sysObjectID elements of the *system* group of the MIB. The keywords associated with these elements are DESCR and OBJID, respectively.

EXAMPLE

descr=Generic SNMPD Version 1.1
objid=UTK_UNIX_agent.1.1

FILES

/etc/snmp.d/snmpd.conf

SEE ALSO

snmpd(1M)
RFC 1065, RFC 1066

NAME

snmpd.trap - SNMP trap communities file

SYNOPSIS

/etc/snmp.d/snmpd.trap

DESCRIPTION

/etc/snmp.d/snmpd.trap contains the definitions for the hosts which will be sent a TRAP PDU by the SNMP agent/server daemon, snmpd (1M). The file contains lines which consist of three items: a session or community name, an IP address in dot notation, and the IP port number to send the TRAP PDU to. Lines which begin with '#' are ignored.

Currently, two TRAP PDU's are generated by snmpd . They are the coldStart and authenticationFailure trap types. The coldStart trap is generated when snmpd is started. The authenticationFailure trap is generated when an authentication error occurs.

EXAMPLE

```
test2 192.9.200.99 162
test2 192.9.200.15 162
```

FILES

/etc/snmp.d/snmpd.trap

SEE ALSO

snmpd(1M)
RFC 1066, RFC 1067

NOTICE

The port number specified should always be equal to 162 according to RFC 1067.

NAME

sockio - ioctls that operate directly on sockets

SYNOPSIS

#include <sys/sockio.h>

DESCRIPTION

The ioctls listed in this manual page apply directly to sockets, independent of any underlying protocol. The setsockopt call (see getsockopt(3N)) is the primary method for operating on sockets, rather than on the underlying protocol or network interface. ioctls for a specific network interface or protocol are documented in the manual page for that interface or protocol.

SIOCSPGRP, FIOSETOWN
: The argument is a pointer to an int. Set the process-group ID that will subsequently receive SIGIO or SIGURG signals for the socket referred to by the descriptor passed to ioctl to the value of that int. For the M88000 architecture, BSD semantics are provided; if the int argument is less than zero then it refers to a process-group ID which is the absolute value of the argument. If the argument is greater than zero refers to a process ID.

SIOCGPGRP, FIOGETOWN
: The argument is a pointer to an int. Set the value of that int to the process-group ID that is receiving SIGIO or SIGURG signals for the socket referred to by the descriptor passed to ioctl. For the M88000 architecture, BSD semantics are provided; if the int argument is less than zero then it refers to a process-group ID which is the absolute value of the argument. If the argument is greater than zero refers to a process ID.

SIOCCATMARK
: The argument is a pointer to an int. Set the value of that int to 1 if the read pointer for the socket referred to by the descriptor passed to ioctl points to a mark in the data stream for an out-of-band message. Set the value of that int to 0 if the read pointer for the socket referred to by the descriptor passed to ioctl does not point to a mark in the data stream for an out-of-band message.

SEE ALSO

ioctl(2), getsockopt(2), filio(4)

NAME

space - disk space requirement file

DESCRIPTION

space is an ASCII file that gives information about disk space requirements for the target environment. It defines space needed beyond that which is used by objects defined in the prototype file—for example, files which will be installed with the installf command. It should define the maximum amount of additional space which a package will require.

The generic format of a line in this file is:

pathname blocks inodes

Definitions for the fields are as follows:

pathname Specifies a directory name which may or may not be the mount point for a filesystem. Names that do not begin with a slash (/) indicate relocatable directories. Components of the pathname may be installation parameters.

blocks Defines the number of disk blocks required for installation of the files and directory entries contained in the pathname (using a 512-byte block size).

inodes Defines the number of inodes required for installation of the files and directory entries contained in the pathname.

EXAMPLE

```
# extra space required by config data which is
# dynamically loaded onto the system
data 500  1
```

SEE ALSO

installf(1M), prototype(4)

NAME

`stat` - data returned by `stat` system call

SYNOPSIS

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

DESCRIPTION

The system calls `stat`, `lstat` and `fstat` return data in a `stat` structure, which is defined in `stat.h` for the M88000 family of processors reference platform:

```
struct   stat
{
         dev_t            st_dev;
         lang             st_pad1[3];
         ino_t            st_ino;
         mode_t           st_mode;
         nlink_t          st_nlink;
         uid_t            st_uid;
         gid_t            st_gid;
         dev_t            st_rdev;
         lang             st_pad2[2];
         off_t            st_size;
         lang             st_pad3;
         timestruct_t     st_atime;
         timestruct_t     st_mtime;
         timestruct_t     st_ctime;
         lang             st_blksize;
         lang             st_blocks;
         char             st_fstype [_ST_FSTYPSZ];
         mang             st_pad4[8];
};
```

The constants used in the `st_mode` field are also defined in this file:

```
#define S_IFMT     0xF000    /* type of file */
#define S_IAMB     0x1FF     /* access mode bits */
#define S_IFIFO    0x1000    /* fifo */
#define S_IFCHR    0x2000    /* character special */
#define S_IFDIR    0x4000    /* directory */
#define S_IFNAM    0x5000    /* XENIX special named file */
#define S_INSEM    0x1       /* XENIX semaphore subtype of IFNAM */
#define S_INSMD    0x2       /* XENIX shared data subtype of IFNAM */
#define S_IFBLK    0x6000    /* block special */
#define S_IFREG    0x8000    /* regular */
#define S_IFLNK    0xA000    /* symbolic link */
#define S_SFSOCK   0xC000    /* Socket*/
#define S_ISUID    0x800     /* set user id on execution */
```

```
#define S_ISGID   0x400    /* set group id on execution */
#define S_ISVTX   0x200    /* save swapped text even after use */
#define S_IREAD   00400    /* read permission, owner */
#define S_IWRITE  00200    /* write permission, owner */
#define S_IEXEC   00100    /* execute/search permission, owner */
#define S_ENFMT   S_ISGID  /* record locking enforcement flag */
#define S_IRWXU   00700    /* read, write, execute: owner */
#define S_IRUSR   00400    /* read permission: owner */
#define S_IWUSR   00200    /* write permission: owner */
#define S_IXUSR   00100    /* execute permission: owner */
#define S_IRWXG   00070    /* read, write, execute: group */
#define S_IRGRP   00040    /* read permission: group */
#define S_IWGRP   00020    /* write permission: group */
#define S_IXGRP   00010    /* execute permission: group */
#define S_IRWXO   00007    /* read, write, execute: other */
#define S_IROTH   00004    /* read permission: other */
#define S_IWOTH   00002    /* write permission: other */
#define S_IXOTH   00001    /* execute permission: other */
```

SEE ALSO

stat(2), types(5)

NAME

strcf - STREAMS Configuration File for STREAMS TCP/IP

DESCRIPTION

/etc/strcf contains the script that is executed by slink(1M) to perform the STREAMS configuration operations required for STREAMS TCP/IP.

The standard /etc/strcf file contains several functions that perform various configuration operations, along with a sample boot function. Normally, only the boot function must be modified to customize the configuration for a given installation. In some cases, however, it may be necessary to change existing functions or add new functions.

The following functions perform basic linking operations:

The tp function is used to set up the link between a transport provider, such as TCP, and IP.

```
#
# tp - configure transport provider (i.e. tcp, udp, icmp)
# usage: tp devname
#
tp {
      p = open $1
      ip = open /dev/ip
      link p ip
}
```

The linkint function links the specified streams and does a sifname operation with the given name.

```
#
# linkint - link interface to ip or arp
# usage: linkint top bottom ifname
#
linkint {
      x = link $1 $2
      sifname $1 x $3
}
```

The aplinkint function performs the same function as linkint for an interface that uses the app module.

```
#
# aplinkint - like linkint, but app is pushed on dev
# usage: aplinkint top bottom ifname
#
aplinkint {
      push $2 app
      linkint $1 $2 $3
}
```

The following functions are used to configure different types of Ethernet interfaces:

The uenet function is used to configure an Ethernet interface for a cloning device driver that uses the *unit select* ioctl to select the desired interface. The interface name is constructed by concatenating the supplied prefix and the unit number.

```
#
# uenet - configure ethernet-type interface for cloning
#         driver using unit select
# usage: uenet ip-fd devname ifprefix unit
#
uenet {
     ifname = strcat $3 $4
     dev = open $2
     unitsel dev $4
     aplinkint $1 dev ifname
     dev = open $2
     unitsel dev $4
     arp = open /dev/arp
     linkint arp dev ifname
}
```

The denet function performs the same function as uenet, except that DL_ATTACH is used instead of *unit select*.

```
#
# denet - configure ethernet-type interface for cloning
#         driver using DL_ATTACH
# usage: denet ip-fd devname ifprefix unit
#
denet {
     ifname = strcat $3 $4
     dev = open $2
     dlattach dev $4
     aplinkint $1 dev ifname
     dev = open $2
     dlattach dev $4
     arp = open /dev/arp
     linkint arp dev ifname
}
```

The cenet function is used to configure an Ethernet interface for a cloning device driver that uses a different major number for each interface. The device name is formed by concatenating the supplied device name prefix and the unit number. The interface name is formed in a similar manner using the interface name prefix.

```
#
# cenet - configure ethernet-type interface for cloning
#         driver with one major per interface
# usage: cenet ip-fd devprefix ifprefix unit
#
cenet {
     devname = strcat $2 $4
     ifname = strcat $3 $4
     dev = open devname
```

```
        aplinkint $1 dev ifname
        dev = open devname
        arp = open /dev/arp
        linkint arp dev ifname
}
```

The senet function is used to configure an Ethernet interface for a non-cloning device driver. Two different device nodes must be specified for IP and ARP.

```
#
# senet - configure ethernet-type interface for non-cloning
#         driver
# usage: senet ip-fd ipdevname arpdevname ifname
#
senet {
        dev = open $2
        aplinkint $1 dev $4
        dev = open $3
        arp = open /dev/arp
        linkint arp dev $4
}
```

The senetc function is like senet, except that it allows the specification of a convergence module to be used with the ethernet driver.

```
#
# senetc - configure ethernet-type interface for non-cloning
#          driver using convergence module
# usage: senetc ip-fd convergence ipdevname arpdevname ifname
#
senetc {
        dev = open $3
        push dev $2
        aplinkint $1 dev $5
        dev = open $4
        push dev $2
        arp = open /dev/arp
        linkint arp dev $5
}
```

The loopback function is used to configure the loopback interface.

```
#
# loopback - configure loopback device
# usage: loopback ip-fd
#
loopback {
        dev = open /dev/loop
        linkint $1 dev lo0
}
```

The slip function is used to configure a SLIP interface. This function is not normally executed at boot time. Rather, the slattach(1M) command runs slink specifying slip on the command line.

```
#
# slip - configure slip interface
# usage: slip unit
#
slip {
     ip = open /dev/ip
     s = open /dev/slip
     ifname = strcat sl $1
     unitsel s $1
     linkint ip s ifname
}
```

The `boot` function is called by default when `slink` is executed. Normally, only the *interfaces* section and possibly the *queue params* section will have to be customized for a given installation. Examples are provided for the various Ethernet driver types.

```
#
# boot - boot time configuration
#
boot {
     #
     # queue params
     #
     initqp /dev/udp rq 8192 40960
     initqp /dev/ip muxrq 8192 40960 rq 8192 40960
     #
     # transport
     #
     tp /dev/tcp
     tp /dev/udp
     tp /dev/icmp
     tp /dev/rawip
}
```

FILES

/etc/strcf

SEE ALSO

slattach(1M), slink(1M)

NAME

streamio - STREAMS ioctl commands

SYNOPSIS

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

int ioctl (int fildes, int command, . . . /* arg */);
```

DESCRIPTION

STREAMS [see intro(2)] ioctl commands are a subset of the ioctl(2) system calls which perform a variety of control functions on streams.

fildes is an open file descriptor that refers to a stream. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure. The *command* and *arg* are interpreted by the stream head. Certain combinations of these arguments may be passed to a module or driver in the stream.

Since these STREAMS commands are a subset of ioctl, they are subject to the errors described there. In addition to those errors, the call will fail with errno set to EINVAL, without processing a control function, if the stream referenced by *fildes* is linked below a multiplexor, or if *command* is not a valid value for a stream.

Also, as described in ioctl, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the stream head containing an error value. This causes subsequent system calls to fail with errno set to this value.

COMMAND FUNCTIONS

The following ioctl commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH — Pushes the module whose name is pointed to by *arg* onto the top of the current stream, just below the stream head. If the stream is a pipe, the module will be inserted between the stream heads of both ends of the pipe. It then calls the open routine of the newly-pushed module. On failure, errno is set to one of the following values:

- EINVAL — Invalid module name.
- EFAULT — *arg* points outside the allocated address space.
- ENXIO — Open routine of new module failed.
- ENXIO — Hangup received on *fildes*.

I_POP — Removes the module just below the stream head of the stream pointed to by *fildes*. To remove a module from a pipe requires that the module was pushed on the side it is being removed from. *arg* should be 0 in an I_POP request. On failure, errno is set to one of the following values:

- EINVAL — No module present in the stream.

ENXIO Hangup received on *fildes*.

I_LOOK Retrieves the name of the module just below the stream head of the stream pointed to by *fildes*, and places it in a null terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long. A #include <sys/conf.h> declaration is required. On failure, errno is set to one of the following values:

EFAULT *arg* points outside the allocated address space.

EINVAL No module present in stream.

I_FLUSH This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

FLUSHR Flush read queues.

FLUSHW Flush write queues.

FLUSHRW Flush read and write queues.

If a pipe or FIFO does not have any modules pushed, the read queue of the stream head on either end is flushed depending on the value of *arg*.

If FLUSHR is set and *fildes* is a pipe, the read queue for that end of the pipe is flushed and the write queue for the other end is flushed. If *fildes* is a FIFO, both queues are flushed.

If FLUSHW is set and *fildes* is a pipe and the other end of the pipe exists, the read queue for the other end of the pipe is flushed and the write queue for this end is flushed. If *fildes* is a FIFO, both queues of the FIFO are flushed.

If FLUSHRW is set, all read queues are flushed, that is, the read queue for the FIFO and the read queue on both ends of the pipe are flushed.

Correct flush handling of a pipe or FIFO with modules pushed is achieved via the pipemod module. This module should be the first module pushed onto a pipe so that it is at the midpoint of the pipe itself.

On failure, errno is set to one of the following values:

ENOSR Unable to allocate buffers for flush message due to insufficient STREAMS memory resources.

EINVAL Invalid *arg* value.

ENXIO Hangup received on *fildes*.

I_FLUSHBAND

Flushes a particular band of messages. *arg* points to a bandinfo structure that has the following members:

```
unsigned char   bi_pri;
int             bi_flag;
```

The bi_flag field may be one of FLUSHR, FLUSHW, or FLUSHRW as described earlier.

I_SETSIG Informs the stream head that the user wishes the kernel to issue the SIGPOLL signal [see signal(2)] when a particular event has occurred on the stream associated with *fildes*. I_SETSIG supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

S_INPUT Any message other than an M_PCPROTO has arrived on a stream head read queue. This event is maintained for compatibility with prior UNIX System V releases. This is set even if the message is of zero length.

S_RDNORM An ordinary (non-priority) message has arrived on a stream head read queue. This is set even if the message is of zero length.

S_RDBAND A priority band message (band > 0) has arrived on a stream head read queue. This is set even if the message is of zero length.

S_HIPRI A high priority message is present on the stream head read queue. This is set even if the message is of zero length.

S_OUTPUT The write queue just below the stream head is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.

S_WRNORM This event is the same as S_OUTPUT.

S_WRBAND A priority band greater than 0 of a queue downstream exists and is writable. This notifies the user that there is room on the queue for sending (or writing) priority data downstream.

S_MSG A STREAMS signal message that contains the SIGPOLL signal has reached the front of the stream head read queue.

S_ERROR An M_ERROR message has reached the stream head.

S_HANGUP An M_HANGUP message has reached the stream head.

S_BANDURG When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the stream head read queue.

A user process may choose to be signaled only of high priority messages by setting the *arg* bitmask to the value S_HIPRI.

Processes that wish to receive SIGPOLL signals must explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same stream, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further SIGPOLL signals. On failure, errno is set to one of the following values:

EINVAL *arg* value is invalid or *arg* is zero and process is not registered to receive the SIGPOLL signal.

EAGAIN Allocation of a data structure to store the signal request failed.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of I_SETSIG above. On failure, errno is set to one of the following values:

EINVAL Process not registered to receive the SIGPOLL signal.

EFAULT *arg* points outside the allocated address space.

I_FIND Compares the names of all modules currently present in the stream to the name pointed to by *arg*, and returns 1 if the named module is present in the stream. It returns 0 if the named module is not present. On failure, errno is set to one of the following values:

EFAULT *arg* points outside the allocated address space.

EINVAL *arg* does not contain a valid module name.

I_PEEK Allows a user to retrieve the information in the first message on the stream head read queue without taking the message off the queue. I_PEEK is analogous to getmsg(2) except that it does not remove the message from the queue. *arg* points to a strpeek structure which contains the following members:

```
struct strbuf    ctlbuf;
struct strbuf    databuf;
long             flags;
```

The maxlen field in the ctlbuf and databuf strbuf structures [see getmsg(2)] must be set to the number of bytes of control information and/or data information, respectively, to retrieve. flags may be set to RS_HIPRI or 0. If RS_HIPRI is set, I_PEEK will look for a high priority message on the stream head read queue. Otherwise, I_PEEK will look for the first message on the stream head read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the stream head read queue. It does not wait for a message to arrive. On return, ctlbuf specifies information in the control buffer, databuf specifies information in the data buffer, and flags contains the value RS_HIPRI or 0. On failure, errno is set to the following value:

EFAULT *arg* points, or the buffer area specified in ctlbuf or databuf is, outside the allocated address space.

EBADMSG Queued message to be read is not valid for I_PEEK

EINVAL Illegal value for flags.

I_SRDOPT Sets the read mode [see read(2)] using the value of the argument *arg*. Legal *arg* values are:

RNORM Byte-stream mode, the default.

RMSGD Message-discard mode.

RMSGN Message-nondiscard mode.

In addition, treatment of control messages by the stream head may be changed by setting the following flags in *arg*:

RPROTNORM Fail read() with EBADMSG if a control message is at the front of the stream head read queue. This is the default behavior.

RPROTDAT Deliver the control portion of a message as data when a user issues read().

RPROTDIS Discard the control portion of a message, delivering any data portion, when a user issues a read().

On failure, errno is set to the following value:

EINVAL *arg* is not one of the above legal values.

I_GRDOPT Returns the current read mode setting in an int pointed to by the argument *arg*. Read modes are described in read(2). On failure, errno is set to the following value:

EFAULT *arg* points outside the allocated address space.

I_NREAD Counts the number of data bytes in data blocks in the first message on the stream head read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the stream head read queue. For example, if zero is returned in *arg*, but the ioctl return value is greater than zero, this indicates that a zero-length message is next on the queue. On failure, errno is set to the following value:

EFAULT *arg* points outside the allocated address space.

I_FDINSERT Creates a message from user specified buffer(s), adds information about another stream and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

arg points to a strfdinsert structure which contains the following members:

```
struct strbuf     ctlbuf;
struct strbuf     databuf;
long              flags;
int               fildes;
int               offset;
```

The len field in the ctlbuf strbuf structure [see putmsg(2)] must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fildes* in the strfdinsert structure specifies the file descriptor of the other stream. offset, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where I_FDINSERT will store a pointer. This pointer will be the address of the read queue structure of the driver for the stream corresponding to fildes in the strfdinsert structure. The len field in the databuf strbuf structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

flags specifies the type of message to be created. An ordinary (non-priority) message is created if flags is set to 0, a high priority message is created if flags is set to RS_HIPRI. For normal messages, I_FDINSERT will block if the stream write queue is full due to internal flow control conditions. For high priority messages, I_FDINSERT does not block on this condition. For normal messages, I_FDINSERT does not block when the write queue is full and O_NDELAY or O_NONBLOCK is set. Instead, it fails and sets errno to EAGAIN.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks, regardless of priority or whether O_NDELAY or O_NONBLOCK has been specified. No partial message is sent. On failure, errno is set to one of the following values:

EAGAIN A non-priority message was specified, the O_NDELAY or O_NONBLOCK flag is set, and the stream write queue is full due to internal flow control conditions.

ENOSR Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.

EFAULT *arg* points, or the buffer area specified in ctlbuf or databuf is, outside the allocated address space.

EINVAL One of the following: fildes in the strfdinsert structure is not a valid, open stream file descriptor; the size of a pointer plus offset is greater than the len field for the buffer specified through ctlptr; offset does not specify a properly-aligned location in the data buffer; an undefined value is stored in flags.

ENXIO Hangup received on fildes of the ioctl call or fildes in the strfdinsert structure.

ERANGE The len field for the buffer specified through databuf does not fall within the range specified by the maximum and minimum packet sizes of the topmost stream module, or the len field for the buffer specified through databuf is larger than the maximum configured size of the data part of a message, or the len field for the buffer specified through ctlbuf is larger than the maximum configured size of the control part of a message.

I_FDINSERT can also fail if an error message was received by the stream head of the stream corresponding to fildes in the strfdinsert structure. In this case, errno will be set to the value in the message.

I_STR Constructs an internal STREAMS ioctl message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to send user ioctl requests to downstream modules and drivers. It allows information to be sent with the ioctl, and will return to the user any information sent upstream by the downstream recipient. I_STR blocks until the system responds with either a positive or negative acknowledgement message, or until the request "times out" after some period of time. If the request times out, it fails with errno set to ETIME.

At most, one I_STR can be active on a stream. Further I_STR calls will block until the active I_STR completes at the stream head. The default timeout interval for these requests is 15 seconds. The O_NDELAY and O_NONBLOCK [see open(2)] flags have no effect on this call.

To send requests downstream, *arg* must point to a strioctl structure which contains the following members:

```
int     ic_cmd;
int     ic_timout;
int     ic_len;
char    *ic_dp;
```

ic_cmd is the internal ioctl command intended for a downstream module or driver and ic_timout is the number of seconds (-1 = infinite, 0 = use default, >0 = as specified) an I_STR request will wait for acknowledgement before timing out. The default timeout is infinite. ic_len is the number of bytes in the data argument and ic_dp is a pointer to the data argument. The ic_len field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by ic_dp should be large enough to contain the maximum amount of data that any module or the driver in the stream can return).

The stream head will convert the information pointed to by the strioctl structure to an internal ioctl command message and send it downstream. On failure, errno is set to one of the following values:

ENOSR Unable to allocate buffers for the ioctl message due to insufficient STREAMS memory resources.

EFAULT *arg* points, or the buffer area specified by ic_dp and ic_len (separately for data sent and data returned) is, outside the allocated address space.

EINVAL ic_len is less than 0 or ic_len is larger than the maximum configured size of the data part of a message or ic_timout is less than -1.

ENXIO Hangup received on *fildes*.

ETIME A downstream ioctl timed out before acknowledgement was received.

An I_STR can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the stream head. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the ioctl command sent downstream fails. For these cases, I_STR will fail with errno set to the value in the message.

I_SWROPT Sets the write mode using the value of the argument *arg*. Legal bit settings for *arg* are:

SNDZERO Send a zero-length message downstream when a write of 0 bytes occurs.

To not send a zero-length message when a write of 0 bytes occurs, this bit must not be set in *arg*.

On failure, errno may be set to the following value:

EINVAL *arg* is the the above legal value.

I_GWROPT Returns the current write mode setting, as described above, in the int that is pointed to by the argument *arg*.

I_SENDFD Requests the stream associated with *fildes* to send a message, containing a file pointer, to the stream head at the other end of a stream pipe. The file pointer corresponds to *arg*, which must be an open file descriptor.

I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue [see intro(2)] of the stream head at the other end of the stream pipe to which it is connected. On failure, errno is set to one of the following values:

EAGAIN The sending stream is unable to allocate a message block to contain the file pointer.

EAGAIN The read queue of the receiving stream head is full and cannot accept the message sent by I_SENDFD.

EBADF *arg* is not a valid, open file descriptor.

EINVAL *fildes* is not connected to a stream pipe.

ENXIO Hangup received on *fildes*.

`I_RECVFD` Retrieves the file descriptor associated with the message sent by an `I_SENDFD` `ioctl` over a stream pipe. *arg* is a pointer to a data buffer large enough to hold an `strrecvfd` data structure containing the following members:

```
int fd;
uid_t uid;
gid_t gid;
char fill[8];
```

`fd` is an integer file descriptor. `uid` and `gid` are the user id and group id, respectively, of the sending stream.

If `O_NDELAY` and `O_NONBLOCK` are clear [see open(2)], `I_RECVFD` will block until a message is present at the stream head. If `O_NDELAY` or `O_NONBLOCK` is set, `I_RECVFD` will fail with `errno` set to `EAGAIN` if no message is present at the stream head.

If the message at the stream head is a message sent by an `I_SENDFD`, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the `fd` field of the `strrecvfd` structure. The structure is copied into the user data buffer pointed to by *arg*. On failure, `errno` is set to one of the following values:

`EAGAIN`	A message is not present at the stream head read queue, and the `O_NDELAY` or `O_NONBLOCK` flag is set.
`EBADMSG`	The message at the stream head read queue is not a message containing a passed file descriptor.
`EFAULT`	*arg* points outside the allocated address space.
`EMFILE`	`NOFILES` file descriptors are currently open.
`ENXIO`	Hangup received on *fildes*.
`EOVERFLOW`	*uid* or *gid* is too large to be stored in the structure pointed to by *arg*.

`I_LIST` Allows the user to list all the module names on the stream, up to and including the topmost driver name. If *arg* is `NULL`, the return value is the number of modules, including the driver, that are on the stream pointed to by *fildes*. This allows the user to allocate enough space for the module names. If *arg* is non-`NULL`, it should point to an `str_list` structure that has the following members:

```
int sl_nmods;
struct str_mlist   *sl_modlist;
```

The `str_mlist` structure has the following member:

```
char l_name[FMNAMESZ+1];
```

`sl_nmods` indicates the number of entries the user has allocated in the array and on return, `sl_modlist` contains the list of module names. The return value indicates the number of entries that have been filled in. On failure, `errno` may be set to one of the following values:

EINVAL The sl_nmods member is less than 1.

EAGAIN Unable to allocate buffers

I_ATMARK Allows the user to see if the current message on the stream head read queue is "marked" by some module downstream. *arg* determines how the checking is done when there may be multiple marked messages on the stream head read queue. It may take the following values:

ANYMARK Check if the message is marked.

LASTMARK Check if the message is the last one marked on the queue.

The return value is 1 if the mark condition is satisfied and 0 otherwise. On failure, errno may be set to the following value:

EINVAL Invalid *arg* value.

I_CKBAND Check if the message of a given priority band exists on the stream head read queue. This returns 1 if a message of a given priority exists, or -1 on error. *arg* should be an integer containing the value of the priority band in question. On failure, errno may be set to the following value:

EINVAL Invalid *arg* value.

I_GETBAND Returns the priority band of the first message on the stream head read queue in the integer referenced by *arg*. On failure, errno may be set to the following value:

ENODATA No message on the stream head read queue.

I_CANPUT Check if a certain band is writable. *arg* is set to the priority band in question. The return value is 0 if the priority band *arg* is flow controlled, 1 if the band is writable, or -1 on error. On failure, errno may be set to the following value:

EINVAL Invalid *arg* value.

I_SETCLTIME

Allows the user to set the time the stream head will delay when a stream is closing and there are data on the write queues. Before closing each module and driver, the stream head will delay for the specified amount of time to allow the data to drain. If, after the delay, data are still present, data will be flushed. *arg* is a pointer to the number of milliseconds to delay, rounded up to the nearest legal value on the system. The default is fifteen seconds. On failure, errno may be set to the following value:

EINVAL Invalid *arg* value.

I_GETCLTIME

Returns the close time delay in the long pointed by *arg*.

The following four commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK
: Connects two streams, where *fildes* is the file descriptor of the stream connected to the multiplexing driver, and *arg* is the file descriptor of the stream connected to another driver. The stream designated by *arg* gets connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgement message to the stream head regarding the linking operation. This call returns a multiplexor ID number (an identifier used to disconnect the multiplexor, see I_UNLINK) on success, and a -1 on failure. On failure, errno is set to one of the following values:

ENXIO
: Hangup received on *fildes*.

ETIME
: Time out before acknowledgement message was received at stream head.

EAGAIN
: Temporarily unable to allocate storage to perform the I_LINK.

ENOSR
: Unable to allocate storage to perform the I_LINK due to insufficient STREAMS memory resources.

EBADF
: *arg* is not a valid, open file descriptor.

EINVAL
: *fildes* stream does not support multiplexing.

EINVAL
: *arg* is not a stream, or is already linked under a multiplexor.

EINVAL
: The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given driver is linked into a multiplexing configuration in more than one place.

EINVAL
: *fildes* is the file descriptor of a pipe or FIFO.

An I_LINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_LINK will fail with errno set to the value in the message.

I_UNLINK
: Disconnects the two streams specified by *fildes* and *arg*. *fildes* is the file descriptor of the stream connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by the I_LINK. If *arg* is -1, then all Streams which were linked to *fildes* are disconnected. As in I_LINK, this command requires the multiplexing driver to acknowledge the unlink. On failure, errno is set to one of the following values:

ENXIO
: Hangup received on *fildes*.

ETIME
: Time out before acknowledgement message was received at stream head.

ENOSR
: Unable to allocate storage to perform the I_UNLINK due to insufficient STREAMS memory resources.

EINVAL *arg* is an invalid multiplexor ID number or *fildes* is not the stream on which the I_LINK that returned *arg* was performed.

EINVAL *fildes* is the file descriptor of a pipe or FIFO.

An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_UNLINK will fail with errno set to the value in the message.

I_PLINK Connects two streams, where *fildes* is the file descriptor of the stream connected to the multiplexing driver, and *arg* is the file descriptor of the stream connected to another driver. The stream designated by *arg* gets connected via a persistent link below the multiplexing driver. I_PLINK requires the multiplexing driver to send an acknowledgement message to the stream head regarding the linking operation. This call creates a persistent link which can exist even if the file descriptor *fildes* associated with the upper stream to the multiplexing driver is closed. This call returns a multiplexor ID number (an identifier that may be used to disconnect the multiplexor, see I_PUNLINK) on success, and a -1 on failure. On failure, errno may be set to one of the following values:

ENXIO Hangup received on *fildes*.

ETIME Time out before acknowledgement message was received at the stream head.

EAGAIN Unable to allocate STREAMS storage to perform the I_PLINK.

EBADF *arg* is not a valid, open file descriptor.

EINVAL *fildes* does not support multiplexing.

EINVAL *arg* is not a stream or is already linked under a multiplexor.

EINVAL The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given stream head is linked into a multiplexing configuration in more than one place.

EINVAL *fildes* is the file descriptor of a pipe or FIFO.

An I_PLINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error on a hangup is received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_PLINK will fail with errno set to the value in the message.

I_PUNLINK — Disconnects the two streams specified by *fildes* and *arg* that are connected with a persistent link. *fildes* is the file descriptor of the stream connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by I_PLINK when a stream was linked below the multiplexing driver. If *arg* is MUXID_ALL then all streams which are persistent links to *fildes* are disconnected. As in I_PLINK, this command requires the multiplexing driver to acknowledge the unlink. On failure, errno may be set to one of the following values:

- ENXIO — Hangup received on *fildes*.
- ETIME — Time out before acknowledgement message was received at the stream head.
- EAGAIN — Unable to allocate buffers for the acknowledgement message.
- EINVAL — Invalid multiplexor ID number.
- EINVAL — *fildes* is the file descriptor of a pipe or FIFO.

An I_PUNLINK can also fail while waiting for the multiplexing driver to acknowledge the link request if a message indicating an error or a hangup is received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_PUNLINK will fail with errno set to the value in the message.

SEE ALSO

close(2), fcntl(2), getmsg(2), intro(2), ioctl(2), open(2), poll(2), putmsg(2), read(2), signal(2), write(2), signal(5).

DIAGNOSTICS

Unless specified otherwise above, the return value from ioctl is 0 upon success and -1 upon failure with errno set as indicated.

NAME

strftime - language specific strings

DESCRIPTION

There can exist one printable file per locale to specify its date and time formatting information. These files must be kept in the directory /usr/lib/locale/<*locale*>/LC_TIME. The contents of these files are:

1. abbreviated month names (in order)
2. month names (in order)
3. abbreviated weekday names (in order)
4. weekday names (in order)
5. default strings that specify formats for locale time (%X) and locale date (%x).
6. default format for cftime, if the argument for cftime is zero or null.
7. AM (ante meridian) string
8. PM (post meridian) string

Each string is on a line by itself. All white space is significant. The order of the strings in the above list is the same order in which they must appear in the file.

EXAMPLE

/usr/lib/locale/C/LC_TIME

```
Jan
Feb
  ...
January
February
  ...
Sun
Mon
  ...
Sunday
Monday
  ...
%H:%M:%S
%m/%d/%y
%a %b %d %T %Z %Y
AM
PM
```

FILES

/usr/lib/locale/<*locale*>/LC_TIME

SEE ALSO

ctime(3C), setlocale(3C), strftime(3C)

NAME

sxt - pseudo-device driver

DESCRIPTION

The special file /dev/sxt is a pseudo-device driver that interposes a discipline between the standard tty line disciplines and a real device driver. The standard disciplines manipulate virtual tty structures (channels) declared by the /dev/sxt driver. /dev/sxt acts as a discipline manipulating a real tty structure declared by a real device driver. The /dev/sxt driver is currently only used by the shl(1) command.

Virtual ttys are named by inodes in the subdirectory /dev/sxt and are allocated in groups of up to eight. To allocate a group, a program should exclusively open a file with a name of the form /dev/sxt/??0 (channel 0) and then execute a SXTIOCLINK ioctl call to initiate the multiplexing.

Only one channel, the controlling channel, can receive input from the keyboard at a time; others attempting to read will be blocked.

There are two groups of ioctl(2) commands supported by sxt. The first group contains the standard ioctl commands described in termio(7), with the addition of the following:

TIOCEXCL — Set exclusive use mode: no further opens are permitted until the file has been closed.

TIOCNXCL — Reset exclusive use mode: further opens are once again permitted.

The second group are commands to sxt itself. Some of these may only be executed on channel 0.

SXTIOCLINK — Allocate a channel group and multiplex the virtual ttys onto the real tty. The argument is the number of channels to allocate. This command may only be executed on channel 0. Possible errors include:

- EINVAL — The argument is out of range.
- ENOTTY — The command was not issued from a real tty.
- ENXIO — linesw is not configured with sxt.
- EBUSY — An SXTIOCLINK command has already been issued for this real tty.
- ENOMEM — There is no system memory available for allocating the virtual tty structures.
- EBADF — Channel 0 was not opened before this call.

SXTIOCSWTCH — Set the controlling channel. Possible errors include:

- EINVAL — An invalid channel number was given.
- EPERM — The command was not executed from channel 0.

SXTIOCWF	Cause a channel to wait until it is the controlling channel. This command will return the error, EINVAL, if an invalid channel number is given.
SXTIOCUBLK	Turn off the loblk control flag in the virtual tty of the indicated channel. The error EINVAL will be returned if an invalid number or channel 0 is given.
SXTIOCSTAT	Get the status (blocked on input or output) of each channel and store in the sxtblock structure referenced by the argument. The error EFAULT will be returned if the structure cannot be written.
SXTIOCTRACE	Enable tracing. Tracing information is written to the console. This command has no effect if tracing is not configured.
SXTIOCNOTRACE	Disable tracing. This command has no effect if tracing is not configured.

FILES

/dev/sxt/??[0-7] Virtual tty devices

SEE ALSO

shl(1), stty(1) ioctl(2), open(2), termio(7).

NAME

`syslog.conf` - configuration file for syslogd system log daemon

SYNOPSIS

`/etc/syslog.conf`

DESCRIPTION

The file `/etc/syslog.conf` contains information used by the system log daemon, `syslogd`(1M), to forward a system message to appropriate log files and/or users. `syslog` preprocesses this file through `m4`(1) to obtain the correct information for certain log files.

A configuration entry is composed of two TAB-separated fields:

"selector action"

The *selector* field contains a semicolon-separated list of priority specifications of the form:

facility . *level* [; *facility* . *level*]

where *facility* is a system facility, or comma-separated list of facilities, and *level* is an indication of the severity of the condition being logged. Recognized values for *facility* include:

`user` Messages generated by user processes. This is the default priority for messages from programs or facilities not listed in this file.

`kern` Messages generated by the kernel.

`mail` The mail system.

`daemon` System daemons, such as `ftpd`(1M), `routed`(1M), and so on.

`auth` The authorization system: `login`(1), `su`(1M), `getty`(1M), and so on.

`lpr` The line printer spooling system: `lpr`(1), `lpc`(1M), `lpd`(1M), and so on.

`news` Reserved for the USENET network news system.

`uucp` Reserved for the UUCP system; it does not currently use the `syslog` mechanism.

`cron` The `cron` /at facility; `crontab`(1), `at`(1), `cron`(1M), and so on.

`local0-7` Reserved for local use.

`mark` For timestamp messages produced internally by `syslogd`.

`*` An asterisk indicates all facilities except for the `mark` facility.

Recognized values for *level* are (in descending order of severity):

`emerg` For panic conditions that would normally be broadcast to all users.

`alert` For conditions that should be corrected immediately, such as a corrupted system database.

`crit` For warnings about critical conditions, such as hard device errors.

`err` For other errors.

`warning` For warning messages.

`notice` For conditions that are not error conditions, but may require special handling.

`info` Informational messages.

`debug` For messages that are normally used only when debugging a program.

`none` Do not send messages from the indicated *facility* to the selected file. For example, a *selector* of

```
*.debug;mail.none
```

will send all messages *except* mail messages to the selected file.

The *action* field indicates where to forward the message. Values for this field can have one of four forms:

A filename, beginning with a leading slash, which indicates that messages specified by the *selector* are to be written to the specified file. The file will be opened in append mode.

The name of a remote host, prefixed with an `@`, as with: `@`*server*, which indicates that messages specified by the *selector* are to be forwarded to the `syslogd` on the named host.

A comma-separated list of usernames, which indicates that messages specified by the *selector* are to be written to the named users if they are logged in.

An asterisk, which indicates that messages specified by the *selector* are to be written to all logged-in users.

Blank lines are ignored. Lines for which the first nonwhite character is a '#' are treated as comments.

EXAMPLE

With the following configuration file:

```
*.notice;mail.info          /var/log/notice
*.crit                      /var/log/critical
kern,mark.debug             /dev/console
kern.err                    @server
*.emerg                     *
*.alert                     root,operator
*.alert;auth.warning        /var/log/auth
```

`syslogd` will log all mail system messages except `debug` messages and all `notice` (or higher) messages into a file named `/var/log/notice`. It logs all critical messages into `/var/log/critical`, and all kernel messages and 20-minute marks onto the system console.

Kernel messages of `err` (error) severity or higher are forwarded to the machine named *server*. Emergency messages are forwarded to all users. The users root and operator are informed of any `alert` messages. All messages from the authorization system of `warning` level or higher are logged in the file `/var/log/auth`.

FILES

/etc/syslog.conf
/var/log/notice
/var/log/critical
/var/log/auth

SEE ALSO

logger(1), lpr(1), syslogd(1M), syslog(3)

at(1), cron(1M), crontab(1), getty(1M), login(1), lp(1), m4(1), su(1M).

NAME

system - system configuration information file

DESCRIPTION

The system file is used during the configuration of a new operating system to obtain configuration information that cannot be obtained from the Equipped Device Table (EDT). The system file is /stand/system.

The system file generally contains a list of software drivers to include in the new bootable operating system, the assignment of system devices such as swapdev and rootdev, and instructions for excluding drivers from the configuration process.

The parser for the system file is case-sensitive. All upper case strings in the syntax below should be upper case in the system file as well. Nonterminal symbols are enclosed in angle brackets <>, whereas optional arguments are enclosed in square brackets []. Ellipses (. . .) indicate optional repetition of the argument for that line.

The symbols in the syntax description below are interpreted as follows:

<fname>	::=	pathname
<string>	::=	driver file name from /boot or EDT entry name
<device>	::=	special device name | DEV(*<major>*,*<minor>*)
<major>	::=	*<number>*
<minor>	::=	*<number>*
<number>	::=	decimal, octal or hex literal

The lines listed below may appear in any order. Blank lines may be inserted at any point. Comment lines must begin with an asterisk. Entries for EXCLUDE and INCLUDE are cumulative. For all other entries, the last line to appear in the file is used—any earlier entries are ignored.

BOOT: *<fname>*

Specifies the KERNEL object file to be used to build the bootable operating system; if *<fname>* is the keyword DEFAULT, the configuration program takes the KERNEL file from whatever boot directory it is using. For example, if the user types cunix -b */my_boot_directory* and the system file contains the DEFAULT keyword for the BOOT directive, then the KERNEL file used is */my_boot_directory*/KERNEL. If no -b option is used then cunix searches /boot by default; see cunix(1M).

EXCLUDE: *<string>* . . .

Specifies drivers to exclude from the configuration even if the device is found in the EDT.

INCLUDE: *<string>*[(*<number>*)] . . .

Specifies software drivers or loadable modules to be included in the configuration. The optional *<number>* (parentheses required) specifies the number of devices to be controlled by the driver (defaults to 1). This number corresponds to the builtin variable #C which may be referred to by expressions in part one of the master file.

ROOTDEV: *<device>*

Identifies the device containing the root file system.

SWAPDEV: *<device>* *<number>* *<number>*

Identifies the device to be used as swap space. The *<device>* in this case may be a special device file name or a regular file. The *<number>*s correspond to the block number the swap space starts at and the number of swap blocks available.

ICDDEV: *<fname>*

Specifies the regular special file containing an s5 file system image to be used for the In-Core Disk by the new operating system. cunix(1M) will call icdpatch(1M) to open and read the file if this field has a valid file name, *<fname>*.

FILES

/stand/system

SEE ALSO

crash(1M), cunix(1M), icdpatch(1M), and mkboot(1M), master(4)

NAME

tape - tape support

DESCRIPTION

Only the character (raw) interface is supported for tape drives.

The raw device nodes /dev/rmt/prefix_* allow the transfer of a specified number of bytes between the tape drive and a location in the user's address space.

Tape devices may be accessed using fixed or variable block sizes. When operating in fixed mode, tapes must be accessed using buffers in multiples of the configured block size, typically 512 bytes. Exabyte tapes use 1024 bytes. Variable block mode allows records to be any size from 1 byte to the device maximum length, typically 64 KB. However, not all tape devices support variable mode.

Attempts to access a tape in fixed mode with a block size not a multiple of the configured block size results in an error (EIO).

By default, the generic device nodes for cartridge tapes are configured for fixed block mode, and 9-track tape devices are configured for variable mode.

You can only write streaming tapes when the tape is positioned at beginning-of-tape (BOT) or end-of-data (EOD). You may not overwrite a streaming tape in the middle. To overwrite a streaming tape, you *must* rewind the tape before starting to write data. To append a streaming tape, you *must* either perform an MTEND tape ioctl operation before starting to write data or read until you reach EOD, and then close and re-open the tape for writing.

Drivers return EIO when you attempt to read past the end of data, attempt to forward space a record (MTFSR), or backward space a record (MTBSR) across an end-of-file mark.

When an end-of-file mark is encountered while reading a tape, a zero-length or partial read is returned. If a zero-length read is returned, the tape is positioned at the end-of-media side of the end-of-file mark. If a partial read is returned, the tape is positioned at the beginning-of-media side of the end-of-file mark, and the next read succeeds with zero bytes returned. After the zero-length read, additional attempts to read the tape return ENXIO.

Attempting to open a write-protected tape for writing fails and return EIO.

IMPORTANT INFORMATION

When dealing with tapes that contain multiple files or images, it is important to understand how the forward-space-file (fsf) and back-space-file (bsf) commands work. These commands move the tape by counting end-of-file marks actually past over and therefore position the tape to the beginning-of-tape and end-of-medium side of the last file mark skipped, respectively.

In order to get back to the beginning of the file just read you must rewind the tape if the file is the first file on the tape. If the file is second or later on the tape, issue the back-space-file (bsf) command twice followed by a single forward-space-file (fsf) command.

IOCTL COMMANDS

Tapes support several ioctl(2) functions on the character or raw devices. These functions permit control beyond the normal open(2), close(2), read(2), and write(2) system calls. Any attempt to utilize ioctl(2) functions not listed in the following table causes an EINVAL error to be returned. This table gives an overview of the available calls and their syntax, listed alphabetically and with descriptions.

CALL	SYNTAX
MTIOCTOP	ioctl *(fildes, MTIOCTOP, *arg)* struct mtop **arg;* The mtop structure and the value MTIOCTOP are defined in sys/mtio.h. Valid operation codes are: MTBSF, MTBSR, MTCEOM, MTEND, MTERA, MTFSF, MTFSR, MTNOP, MTOFFL, MTREW, MTTEN, and MTWEOF.
MTIOCGET mtget **arg;*	ioctl *(fildes,* MTIOCGET, **arg)* struct The mtget structure and the value MTIOCGET are defined in sys/mtio.h.

MTIOCTOP

The mtop structure is defined in sys/mtio.h. The operation this command performs depends on the value of the mt_op and mt_count fields. The following values for the mt_op field are supported:

MTBSF
: Moves the tape backward past mt_count filemarks. The tape is positioned at the beginning-of-medium side of the filemark. This function is not supported by all tape drives. If it is not supported, the operation fails, returning ENXIO. If it is supported, it will not fail if the operation is attempted before beginning-of-tape.

MTBSR
: Moves the tape backward past mt_count records. For streaming tapes, the record size is always the logical block size (512 bytes default, 1024 bytes for Exabyte). This function is not supported by all tape drives. Whenever it is not supported, the operation fails, returning ENXIO.

MTCEOM
: Clears the end-of-media indicator.

MTEND
: Spaces forward to the end-of-data. For 9-track tapes, it spaces forward two sequential filemarks and positions the tape between them.

MTERA
: Erases the tape. The tape is rewound, erased, and rewound again.

MTFSF
: Moves the tape forward past mt_count filemarks. The tape is positioned at the end-of-medium side of the filemark. If this operation is attempted while the tape is positioned at end-of-data, it fails with EIO.

MTFSR
: Moves the tape forward past mt_count records. For streaming tapes, the record size is always the logical block size (512 bytes default, 1024 bytes for Exabyte). This function is not supported by all tape drives. Whenever it is not supported, the operation fails, returning ENXIO.

MTNOP No operation.

MTOFFL Rewinds the tape and puts the drive offline. For some devices, this may just rewind the tape. Note: operations normally done during close (such as rewinding or writing filemarks) will not be attempted if the drive is put offline.

MTREW Rewinds the tape.

MTTEN Retensions the tape. This operation is not supported by all tape drives. Whenever it is not supported, the tape is rewound instead.

MTWEOF Writes an end-of-file record. An end-of-file can be used only after data has been written with the `write`(2) system call.

MTIOCGET
Returns status information about the tape drive. The `mt_type` field is set to the appropriate value defined in `sys/mtio.h`. Bits in the `mt_dsreg` field are set to indicate whether the tape is write protected or if the drive is offline. Note: if there is no tape in the drive, it is considered both offline and write-protected.

SEE ALSO

`intro`(7)

NAME

TCP - Internet Transmission Control Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, 0);

t = t_open("/dev/tcp", O_RDWR);
```

DESCRIPTION

TCP is the virtual circuit protocol of the Internet protocol family. It provides reliable, flow-controlled, in order, two-way transmission of data. It is a byte-stream protocol layered above the Internet Protocol (IP), the Internet protocol family's internetwork datagram delivery protocol.

Programs can access TCP using the socket interface as a SOCK_STREAM socket type, or using the Transport Level Interface (TLI) where it supports the connection-oriented (T_COTS_ORD) service type.

TCP uses IP's host-level addressing and adds its own per-host collection of port addresses. The endpoints of a TCP connection are identified by the combination of an IP address and a TCP port number. Although other protocols, such as the User Datagram Protocol (UDP), may use the same host and port address format, the port space of these protocols is distinct. See inet(7) for details on the common aspects of addressing in the Internet protocol family.

Sockets utilizing TCP are either active or passive. Active sockets initiate connections to passive sockets. Both types of sockets must have their local IP address and TCP port number bound with the bind(2) system call after the socket is created. By default, TCP sockets are active. A passive socket is created by calling the listen(2) system call after binding the socket with bind(). This establishes a queueing parameter for the passive socket. After this, connections to the passive socket can be received with the accept(2) system call. Active sockets use the connect(2) call after binding to initiate connections.

By using the special value INADDR_ANY, the local IP address can be left unspecified in the bind() call by either active or passive TCP sockets. This feature is usually used if the local address is either unknown or irrelevant. If left unspecified, the local IP address will be bound at connection time to the address of the network interface used to service the connection.

Once a connection has been established, data can be exchanged using the read(2) and write(2) system calls.

TCP supports one socket option which is set with setsockopt() and tested with getsockopt(2). Under most circumstances, TCP sends data when it is presented. When outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgement is received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays. Therefore, TCP provides a boolean option, TCP_NODELAY (defined in /usr/include/netinet/tcp.h), to defeat this algorithm. The option level for the `setsockopt()` call is the protocol number for TCP, available from `getprotobyname()` [see `getprotoent`(3N)].

Options at the IP level may be used with TCP; See `ip`(7).

TCP provides an urgent data mechanism, which may be invoked using the out-of-band provisions of `send`(2). The caller may mark one byte as urgent with the `MSG_OOB` flag to `send`(2). This sets an urgent pointer pointing to this byte in the TCP stream. The receiver on the other side of the stream is notified of the urgent data by a `SIGURG` signal. The `SIOCATMARK` `ioctl()` request returns a value indicating whether the stream is at the urgent mark. Because the system never returns data across the urgent mark in a single `read`(2) call, it is possible to advance to the urgent data in a simple loop which reads data, testing the socket with the `SIOCATMARK` `ioctl()` request, until it reaches the mark.

Incoming connection requests that include an IP source route option are noted, and the reverse source route is used in responding.

A checksum over all data helps TCP implement reliability. Using a window-based flow control mechanism that makes use of positive acknowledgements, sequence numbers, and a retransmission strategy, TCP can usually recover when datagrams are damaged, delayed, duplicated or delivered out of order by the underlying communication medium.

If the local TCP receives no acknowledgements from its peer for a period of time, as would be the case if the remote machine crashed, the connection is closed and an error is returned to the user. If the remote machine reboots or otherwise loses state information about a TCP connection, the connection is aborted and an error is returned to the user.

SEE ALSO

`read`(2), `write`(2), `accept`(3N), `bind`(3N), `connect`(3N), `getprotoent`(3N), `getsockopt`(3N), `listen`(3N), `send`(3N), `inet`(7), `ip`(7)

Postel, Jon, *Transmission Control Protocol - DARPA Internet Program Protocol Specification*, RFC 793, Network Information Center, SRI International, Menlo Park, Calif., September 1981

DIAGNOSTICS

A socket operation may fail if:

`EISCONN`	A `connect()` operation was attempted on a socket on which a `connect()` operation had already been performed.
`ETIMEDOUT`	A connection was dropped due to excessive retransmissions.
`ECONNRESET`	The remote peer forced the connection to be closed (usually because the remote machine has lost state information about the connection due to a crash).
`ECONNREFUSED`	The remote peer actively refused connection establishment (usually because no process is listening to the port).
`EADDRINUSE`	A `bind()` operation was attempted on a socket with a network address/port pair that has already been bound to another socket.

EADDRNOTAVAIL	A `bind()` operation was attempted on a socket with a network address for which no network interface exists.
EACCES	A `bind()` operation was attempted with a reserved port number and the effective user ID of the process was not the privileged user.
ENOBUFS	The system ran out of memory for internal data structures.

NAME

term - format of compiled term file

SYNOPSIS

/usr/share/lib/terminfo/?/*

DESCRIPTION

Compiled terminfo(4) descriptions are placed under the directory /usr/share/lib/terminfo. In order to avoid a linear search of a huge UNIX system directory, a two-level scheme is used: /usr/share/lib/terminfo/*c*/*name* where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, att4425 can be found in the file /usr/share/lib/terminfo/a/att4425. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it is the same on all hardware. An 8-bit byte is assumed, but no assumptions about byte ordering or sign extension are made. Thus, these binary terminfo files can be transported to other hardware with 8-bit bytes.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256**second+first*.) The value -1 is represented by 0377,0377, and the value -2 is represented by 0376,0377; other negative values are illegal. The -1 generally means that a capability is missing from this terminal. The -2 means that the capability has been cancelled in the terminfo source and also is to be considered missing.

The compiled file is created from the source file descriptions of the terminals (see the -I option of infocmp) by using the terminfo compiler, tic, and read by the routine setupterm [see curses(3X).] The file is divided into six parts in the following order: the header, terminal names, boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are (1) the magic number (octal 0432); (2) the size, in bytes, of the names section; (3) the number of bytes in the boolean section; (4) the number of short integers in the numbers section; (5) the number of offsets (short integers) in the strings section; (6) the size, in bytes, of the string table.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the bar (|) character (see term(5)). The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either 0 or 1 as the flag is present or absent. The value of 2 means that the flag has been cancelled. The capabilities are in the same order as the file <term.h>.

Between the boolean section and the number section, a null byte is inserted, if necessary, to ensure that the number section begins on an even byte offset. All short integers are aligned on a short word boundary.

The numbers section is similar to the boolean flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is -1 or -2, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of -1 or -2 means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^X or \c notation are stored in their interpreted form, not the printing representation. Padding information ($<nn>) and parameter information (%x) are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null terminated.

Note that it is possible for setupterm to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since setupterm has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine setupterm must be prepared for both possibilities — this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

As an example, here is terminal information on the AT&T Model 37 KSR terminal as output by the infocmp -I tty37 command:

```
37|tty37|AT&T model 37 teletype,
    hc, os, xon,
    bel=^G, cr=\r, cub1=\b, cud1=\n, cuu1=\E7, hd=\E9,
    hu=\E8, ind=\n,
```

And here is an octal dump of the term file, produced by the od -c /usr/share/lib/terminfo/t/tty37 command:

```
0000000 032 001      \0 032  \0 013  \0 021 001   3  \0   3   7   |   t
0000020   t   y   3   7   |   A   T   &   T       m   o   d   e   l
0000040   3   7       t   e   l   e   t   y   p   e  \0  \0  \0  \0  \0
0000060  \0  \0  \0 001  \0  \0  \0  \0  \0  \0  \0 001  \0  \0  \0  \0
0000100 001  \0  \0  \0  \0  \0 377 377 377 377 377 377 377 377 377 377
0000120 377 377 377 377 377 377 377 377 377 377 377 377 377 377   &  \0
0000140      \0 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000160 377 377   "  \0 377 377 377 377   (  \0 377 377 377 377 377 377
0000200 377 377   0  \0 377 377 377 377 377 377 377 377   -  \0 377 377
0000220 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000520 377 377 377 377 377 377 377 377 377 377 377 377 377 377   $  \0
0000540 377 377 377 377 377 377 377 377 377 377 377 377 377 377   *  \0
0000560 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0001160 377 377 377 377 377 377 377 377 377 377 377 377 377 377   3   7
0001200   |   t   t   y   3   7   |   A   T   &   T       m   o   d   e
0001220   l       3   7       t   e   l   e   t   y   p   e  \0  \r  \0
0001240  \n  \0  \n  \0 007  \0  \b  \0 033   8  \0 033   9  \0 033   7
0001260  \0  \0
0001261
```

Some limitations: total compiled entries cannot exceed 4096 bytes; all entries in the name field cannot exceed 128 bytes.

FILES

`/usr/share/lib/terminfo/?/*` compiled terminal description database
`/usr/include/term.h` `terminfo` header file

SEE ALSO

`curses`(3X).
`infocmp`(1M), `terminfo`(4), `term`(5)

NAME

terminfo - terminal capability data base

SYNOPSIS

/usr/share/lib/terminfo/?/*

DESCRIPTION

terminfo is a database produced by tic that describes the capabilities of devices such as terminals and printers. Devices are described in terminfo source files by specifying a set of capabilities, by quantifying certain aspects of the device, and by specifying character sequences that effect particular results. This database is often used by screen oriented applications such as vi and curses programs, as well as by some UNIX system commands such as ls and more. This usage allows them to work with a variety of devices without changes to the programs.

terminfo source files consist of one or more device descriptions. Each description consists of a header (beginning in column 1) and one or more lines that list the features for that particular device. Every line in a terminfo source file must end in a comma (,). Every line in a terminfo source file except the header must be indented with one or more white spaces (either spaces or tabs).

Entries in terminfo source files consist of a number of comma-separated fields. White space after each comma is ignored. Embedded commas must be escaped by using a backslash. The following example shows the format of a terminfo source file.

alias$_1$ | *alias*$_2$ | ... | *alias*$_n$ | *longname*,
<*white space*> am, lines #24,
<*white space*> home=\Eeh,

The first line, commonly referred to as the header line, must begin in column one and must contain at least two aliases separated by vertical bars. The last field in the header line must be the long name of the device and it may contain any string. Alias names must be unique in the terminfo database and they must conform to UNIX system file naming conventions [see tic(1M)]; they cannot, for example, contain white space or slashes.

Every device must be assigned a name, such as "vt100." Device names (except the long name) should be chosen using the following conventions. The name should not contain hyphens because hyphens are reserved for use when adding suffixes that indicate special modes.

These special modes may be modes that the hardware can be in, or user preferences. To assign a special mode to a particular device, append a suffix consisting of a hyphen and an indicator of the mode to the device name. For example, the -w suffix means "wide mode"; when specified, it allows for a width of 132 columns instead of the standard 80 columns. Therefore, if you want to use a vt100 device set to wide mode, name the device "vt100-w." Use the following suffixes where possible.

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	5410-w
-am	With auto. margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
-*n*	Number of lines on the screen	2300-40
-na	No arrow keys (leave them in local)	c100-na
-*n*p	Number of pages of memory	c100-4p
-rv	Reverse video	4415-rv

The terminfo reference manual page is organized in two sections: "Device Capabilities" and "Printer Capabilities."

PART 1: DEVICE CAPABILITIES

Capabilities in terminfo are of three types: Boolean capabilities (which show that a device has or does not have a particular feature), numeric capabilities (which quantify particular features of a device), and string capabilities (which provide sequences that can be used to perform particular operations on devices).

In the following table, Variable is the name by which a C programmer accesses a capability (at the terminfo level). Capname is the short name for a capability specified in the terminfo source file. It is used by a person updating the source file and by the tput command. Termcap Code is a two-letter sequence that corresponds to the termcap capability name. (Note that termcap is no longer supported.)

Capability names have no real length limit, but an informal limit of five characters has been adopted to keep them short. Whenever possible, capability names are chosen to be the same as or similar to those specified by the ANSI X3.64-1979 standard. Semantics are also intended to match those of the ANSI standard.

All string capabilities listed below may have padding specified, with the exception of those used for input. Input capabilities, listed under the Strings section in the following tables, have names beginning with key_. The #i symbol in the description field of the following tables refers to the *i*th parameter.

Booleans

Variable	Cap-name	Termcap Code	Description
auto_left_margin	bw	bw	cub1 wraps from column 0 to last column
auto_right_margin	am	am	Terminal has automatic margins
back_color_erase	bce	be	Screen erased with background color
can_change	ccc	cc	Terminal can re-define existing color
ceol_standout_glitch	xhp	xs	Standout not erased by overwriting (hp)
col_addr_glitch	xhpa	YA	Only positive motion for hpa/mhpa caps
cpi_changes_res	cpix	YF	Changing character pitch changes resolution
cr_cancels_micro_mode	crxm	YB	Using cr turns off micro mode

Variable	Cap-name	Termcap Code	Description
eat_newline_glitch	xenl	xn	Newline ignored after 80 columns (Concept)
erase_overstrike	eo	eo	Can erase overstrikes with a blank
generic_type	gn	gn	Generic line type (for example, dialup, switch)
hard_copy	hc	hc	Hardcopy terminal
hard_cursor	chts	HC	Cursor is hard to see
has_meta_key	km	km	Has a meta key (shift, sets parity bit)
has_print_wheel	daisy	YC	Printer needs operator to change character set
has_status_line	hs	hs	Has extra "status line"
hue_lightness_saturation	hls	hl	Terminal uses only HLS color notation (Tektronix)
insert_null_glitch	in	in	Insert mode distinguishes nulls
lpi_changes_res	lpix	YG	Changing line pitch changes resolution
memory_above	da	da	Display may be retained above the screen
memory_below	db	db	Display may be retained below the screen
move_insert_mode	mir	mi	Safe to move while in insert mode
move_standout_mode	msgr	ms	Safe to move in standout modes
needs_xon_xoff	nxon	nx	Padding won't work, xon/xoff required
no_esc_ctlc	xsb	xb	Beehive (f1=escape, f2=ctrl C)
non_rev_rmcup	nrrmc	NR	smcup does not reverse rmcup
no_pad_char	npc	NP	Pad character doesn't exist
over_strike	os	os	Terminal overstrikes on hard-copy terminal
prtr_silent	mc5i	5i	Printer won't echo on screen
row_addr_glitch	xvpa	YD	Only positive motion for vpa/mvpa caps
semi_auto_right_margin	sam	YE	Printing in last column causes cr
status_line_esc_ok	eslok	es	Escape can be used on the status line
dest_tabs_magic_smso	xt	xt	Destructive tabs, magic smso char (t1061)
tilde_glitch	hz	hz	Hazeltine; can't print tilde (~)
transparent_underline	ul	ul	Underline character overstrikes
xon_xoff	xon	xo	Terminal uses xon/xoff handshaking

Numbers

Variable	Cap-name	Termcap Code	Description
buffer_capacity	bufsz	Ya	Number of bytes buffered before printing
columns	cols	co	Number of columns in a line
dot_vert_spacing	spinv	Yb	Spacing of pins vertically in pins per inch
dot_horz_spacing	spinh	Yc	Spacing of dots horizontally in dots per inch
init_tabs	it	it	Tabs initially every # spaces
label_height	lh	lh	Number of rows in each label
label_width	lw	lw	Number of columns in each label

Variable	Cap-name	Termcap Code	Description
lines	lines	li	Number of lines on a screen or a page
lines_of_memory	lm	lm	Lines of memory if > lines; 0 means varies
magic_cookie_glitch	xmc	sg	Number of blank characters left by smso or rmso
max_colors	colors	Co	Maximum number of colors on the screen
max_micro_address	maddr	Yd	Maximum value in micro_..._address
max_micro_jump	mjump	Ye	Maximum value in parm_..._micro
max_pairs	pairs	pa	Maximum number of color-pairs on the screen
micro_col_size	mcs	Yf	Character step size when in micro mode
micro_line_size	mls	Yg	Line step size when in micro mode
no_color_video	ncv	NC	Video attributes that can't be used with colors
number_of_pins	npins	Yh	Number of pins in print-head
num_labels	nlab	Nl	Number of labels on screen (start at 1)
output_res_char	orc	Yi	Horizontal resolution in units per character
output_res_line	orl	Yj	Vertical resolution in units per line
output_res_horz_inch	orhi	Yk	Horizontal resolution in units per inch
output_res_vert_inch	orvi	Yl	Vertical resolution in units per inch
padding_baud_rate	pb	pb	Lowest baud rate where padding needed
virtual_terminal	vt	vt	Virtual terminal number (UNIX system)
wide_char_size	widcs	Yn	Character step size when in double wide mode
width_status_line	wsl	ws	Number of columns in status line

Strings

Variable	Cap-name	Termcap Code	Description
acs_chars	acsc	ac	Graphic charset pairs aAbBcC
alt_scancode_esc	scesca	S8	Alternate escape for scancode emulation (default is for vt100)
back_tab	cbt	bt	Back tab
bell	bel	bl	Audible signal (bell)
bit_image_repeat	birep	Zy	Repeat bit-image cell #1 #2 times (use tparm)
bit_image_newline	binel	Zz	Move to next row of the bit image (use tparm)
bit_image_carriage_return	bicr	Yv	Move to beginning of same row (use tparm)
carriage_return	cr	cr	Carriage return
change_char_pitch	cpi	ZA	Change number of characters per inch
change_line_pitch	lpi	ZB	Change number of lines per inch
change_res_horz	chr	ZC	Change horizontal resolution
change_res_vert	cvr	ZD	Change vertical resolution
change_scroll_region	csr	cs	Change to lines #1 through #2 (vt100)
char_padding	rmp	rP	Like ip but when in replace mode

Variable	Cap-name	Termcap Code	Description
char_set_names	csnm	Zy	List of character set names
clear_all_tabs	tbc	ct	Clear all tab stops
clear_margins	mgc	MC	Clear all margins (top, bottom, and sides)
clear_screen	clear	cl	Clear screen and home cursor
clr_bol	el1	cb	Clear to beginning of line, inclusive
clr_eol	el	ce	Clear to end of line
clr_eos	ed	cd	Clear to end of display
code_set_init	csin	ci	Init sequence for multiple codesets
color_names	colornm	Yw	Give name for color #1
column_address	hpa	ch	Horizontal position absolute
command_character	cmdch	CC	Terminal settable cmd character in prototype
cursor_address	cup	cm	Move to row #1 col #2
cursor_down	cud1	do	Down one line
cursor_home	home	ho	Home cursor (if no cup)
cursor_invisible	civis	vi	Make cursor invisible
cursor_left	cub1	le	Move left one space.
cursor_mem_address	mrcup	CM	Memory relative cursor addressing
cursor_normal	cnorm	ve	Make cursor appear normal (undo vs/vi)
cursor_right	cuf1	nd	Non-destructive space (cursor or carriage right)
cursor_to_ll	ll	ll	Last line, first column (if no cup)
cursor_up	cuu1	up	Upline (cursor up)
cursor_visible	cvvis	vs	Make cursor very visible
define_bit_image_region	defbi	Yx	Define rectangular bit-image region (use tparm)
define_char	defc	ZE	Define a character in a character set †
delete_character	dch1	dc	Delete character
delete_line	dl1	dl	Delete line
device_type	devt	dv	Indicate language/codeset support
dis_status_line	dsl	ds	Disable status line
display_pc_char	dispc	S1	Display PC character
down_half_line	hd	hd	Half-line down (forward 1/2 linefeed)
ena_acs	enacs	eA	Enable alternate character set
end_bit_image_region	endbi	Yy	End a bit-image region (use tparm)
enter_alt_charset_mode	smacs	as	Start alternate character set
enter_am_mode	smam	SA	Turn on automatic margins
enter_blink_mode	blink	mb	Turn on blinking
enter_bold_mode	bold	md	Turn on bold (extra bright) mode
enter_ca_mode	smcup	ti	String to begin programs that use cup
enter_delete_mode	smdc	dm	Delete mode (enter)
enter_dim_mode	dim	mh	Turn on half-bright mode

Variable	Cap-name	Termcap Code	Description
enter_doublewide_mode	swidm	ZF	Enable double wide printing
enter_draft_quality	sdrfq	ZG	Set draft quality print
enter_insert_mode	smir	im	Insert mode (enter)
enter_italics_mode	sitm	ZH	Enable italics
enter_leftward_mode	slm	ZI	Enable leftward carriage motion
enter_micro_mode	smicm	ZJ	Enable micro motion capabilities
enter_near_letter_quality	snlq	ZK	Set near-letter quality print
enter_normal_quality	snrmq	ZL	Set normal quality print
enter_pc_charset_mode	smpch	S2	Enter PC character display mode
enter_protected_mode	prot	mp	Turn on protected mode
enter_reverse_mode	rev	mr	Turn on reverse video mode
enter_scancode_mode	smsc	S4	Enter PC scancode mode
enter_secure_mode	invis	mk	Turn on blank mode (characters invisible)
enter_shadow_mode	sshm	ZM	Enable shadow printing
enter_standout_mode	smso	so	Begin standout mode
enter_subscript_mode	ssubm	ZN	Enable subscript printing
enter_superscript_mode	ssupm	ZO	Enable superscript printing
enter_underline_mode	smul	us	Start underscore mode
enter_upward_mode	sum	ZP	Enable upward carriage motion
enter_xon_mode	smxon	SX	Turn on xon/xoff handshaking
erase_chars	ech	ec	Erase #1 characters
exit_alt_charset_mode	rmacs	ae	End alternate character set
exit_am_mode	rmam	RA	Turn off automatic margins
exit_attribute_mode	sgr0	me	Turn off all attributes
exit_ca_mode	rmcup	te	String to end programs that use cup
exit_delete_mode	rmdc	ed	End delete mode
exit_doublewide_mode	rwidm	ZQ	Disable double wide printing
exit_insert_mode	rmir	ei	End insert mode
exit_italics_mode	ritm	ZR	Disable italics
exit_leftward_mode	rlm	ZS	Enable rightward (normal) carriage motion
exit_micro_mode	rmicm	ZT	Disable micro motion capabilities
exit_pc_charset_mode	rmpch	S3	Disable PC character display mode
exit_scancode_mode	rmsc	S5	Disable PC scancode mode
exit_shadow_mode	rshm	ZU	Disable shadow printing
exit_standout_mode	rmso	se	End standout mode
exit_subscript_mode	rsubm	ZV	Disable subscript printing
exit_superscript_mode	rsupm	ZW	Disable superscript printing
exit_underline_mode	rmul	ue	End underscore mode
exit_upward_mode	rum	ZX	Enable downward (normal) carriage motion
exit_xon_mode	rmxon	RX	Turn off xon/xoff handshaking
flash_screen	flash	vb	Visible bell (may not move cursor)

Variable	Cap-name	Termcap Code	Description
form_feed	ff	ff	Hardcopy terminal page eject
from_status_line	fsl	fs	Return from status line
init_1string	is1	i1	Terminal or printer initialization string
init_2string	is2	is	Terminal or printer initialization string
init_3string	is3	i3	Terminal or printer initialization string
init_file	if	if	Name of initialization file
init_prog	iprog	iP	Path name of program for initialization
initialize_color	initc	Ic	Initialize the definition of color
initialize_pair	initp	Ip	Initialize color-pair
insert_character	ich1	ic	Insert character
insert_line	il1	al	Add new blank line
insert_padding	ip	ip	Insert pad after character inserted

The "key_" strings are sent by specific keys. The "key_" descriptions include the macro, defined in curses.h, for the code returned by the curses routine getch when the key is pressed [see curs_getch(3X)].

key_a1	ka1	K1	KEY_A1, upper left of keypad
key_a3	ka3	K3	KEY_A3, upper right of keypad
key_b2	kb2	K2	KEY_B2, center of keypad
key_backspace	kbs	kb	KEY_BACKSPACE, sent by backspace key
key_beg	kbeg	@1	KEY_BEG, sent by beg(inning) key
key_btab	kcbt	kB	KEY_BTAB, sent by back-tab key
key_c1	kc1	K4	KEY_C1, lower left of keypad
key_c3	kc3	K5	KEY_C3, lower right of keypad
key_cancel	kcan	@2	KEY_CANCEL, sent by cancel key
key_catab	ktbc	ka	KEY_CATAB, sent by clear-all-tabs key
key_clear	kclr	kC	KEY_CLEAR, sent by clear-screen or erase key
key_close	kclo	@3	KEY_CLOSE, sent by close key
key_command	kcmd	@4	KEY_COMMAND, sent by cmd (command) key
key_copy	kcpy	@5	KEY_COPY, sent by copy key
key_create	kcrt	@6	KEY_CREATE, sent by create key
key_ctab	kctab	kt	KEY_CTAB, sent by clear-tab key
key_dc	kdch1	kD	KEY_DC, sent by delete-character key
key_dl	kdl1	kL	KEY_DL, sent by delete-line key
key_down	kcud1	kd	KEY_DOWN, sent by terminal down-arrow key
key_eic	krmir	kM	KEY_EIC, sent by rmir or smir in insert mode
key_end	kend	@7	KEY_END, sent by end key
key_enter	kent	@8	KEY_ENTER, sent by enter/send key
key_eol	kel	kE	KEY_EOL, sent by clear-to-end-of-line

Variable	Cap-name	Termcap Code	Description
			key
key_eos	ked	kS	KEY_EOS, sent by clear-to-end-of-screen key
key_exit	kext	@9	KEY_EXIT, sent by exit key
key_f0	kf0	k0	KEY_F(0), sent by function key f0
key_f1	kf1	k1	KEY_F(1), sent by function key f1
key_f2	kf2	k2	KEY_F(2), sent by function key f2
key_f3	kf3	k3	KEY_F(3), sent by function key f3
key_f4	kf4	k4	KEY_F(4), sent by function key f4
key_f5	kf5	k5	KEY_F(5), sent by function key f5
key_f6	kf6	k6	KEY_F(6), sent by function key f6
key_f7	kf7	k7	KEY_F(7), sent by function key f7
key_f8	kf8	k8	KEY_F(8), sent by function key f8
key_f9	kf9	k9	KEY_F(9), sent by function key f9
key_f10	kf10	k;	KEY_F(10), sent by function key f10
key_f11	kf11	F1	KEY_F(11), sent by function key f11
key_f12	kf12	F2	KEY_F(12), sent by function key f12
key_f13	kf13	F3	KEY_F(13), sent by function key f13
key_f14	kf14	F4	KEY_F(14), sent by function key f14
key_f15	kf15	F5	KEY_F(15), sent by function key f15
key_f16	kf16	F6	KEY_F(16), sent by function key f16
key_f17	kf17	F7	KEY_F(17), sent by function key f17
key_f18	kf18	F8	KEY_F(18), sent by function key f18
key_f19	kf19	F9	KEY_F(19), sent by function key f19
key_f20	kf20	FA	KEY_F(20), sent by function key f20
key_f21	kf21	FB	KEY_F(21), sent by function key f21
key_f22	kf22	FC	KEY_F(22), sent by function key f22
key_f23	kf23	FD	KEY_F(23), sent by function key f23
key_f24	kf24	FE	KEY_F(24), sent by function key f24
key_f25	kf25	FF	KEY_F(25), sent by function key f25
key_f26	kf26	FG	KEY_F(26), sent by function key f26
key_f27	kf27	FH	KEY_F(27), sent by function key f27
key_f28	kf28	FI	KEY_F(28), sent by function key f28
key_f29	kf29	FJ	KEY_F(29), sent by function key f29
key_f30	kf30	FK	KEY_F(30), sent by function key f30
key_f31	kf31	FL	KEY_F(31), sent by function key f31
key_f32	kf32	FM	KEY_F(32), sent by function key f32
key_f33	kf33	FN	KEY_F(13), sent by function key f13
key_f34	kf34	FO	KEY_F(34), sent by function key f34
key_f35	kf35	FP	KEY_F(35), sent by function key f35
key_f36	kf36	FQ	KEY_F(36), sent by function key f36
key_f37	kf37	FR	KEY_F(37), sent by function key f37
key_f38	kf38	FS	KEY_F(38), sent by function key f38
key_f39	kf39	FT	KEY_F(39), sent by function key f39

Variable	Cap-name	Termcap Code	Description
key_f40	kf40	FU	KEY_F(40), sent by function key f40
key_f41	kf41	FV	KEY_F(41), sent by function key f41
key_f42	kf42	FW	KEY_F(42), sent by function key f42
key_f43	kf43	FX	KEY_F(43), sent by function key f43
key_f44	kf44	FY	KEY_F(44), sent by function key f44
key_f45	kf45	FZ	KEY_F(45), sent by function key f45
key_f46	kf46	Fa	KEY_F(46), sent by function key f46
key_f47	kf47	Fb	KEY_F(47), sent by function key f47
key_f48	kf48	Fc	KEY_F(48), sent by function key f48
key_f49	kf49	Fd	KEY_F(49), sent by function key f49
key_f50	kf50	Fe	KEY_F(50), sent by function key f50
key_f51	kf51	Ff	KEY_F(51), sent by function key f51
key_f52	kf52	Fg	KEY_F(52), sent by function key f52
key_f53	kf53	Fh	KEY_F(53), sent by function key f53
key_f54	kf54	Fi	KEY_F(54), sent by function key f54
key_f55	kf55	Fj	KEY_F(55), sent by function key f55
key_f56	kf56	Fk	KEY_F(56), sent by function key f56
key_f57	kf57	Fl	KEY_F(57), sent by function key f57
key_f58	kf58	Fm	KEY_F(58), sent by function key f58
key_f59	kf59	Fn	KEY_F(59), sent by function key f59
key_f60	kf60	Fo	KEY_F(60), sent by function key f60
key_f61	kf61	Fp	KEY_F(61), sent by function key f61
key_f62	kf62	Fq	KEY_F(62), sent by function key f62
key_f63	kf63	Fr	KEY_F(63), sent by function key f63
key_find	kfnd	@0	KEY_FIND, sent by find key
key_help	khlp	%1	KEY_HELP, sent by help key
key_home	khome	kh	KEY_HOME, sent by home key
key_ic	kich1	kI	KEY_IC, sent by ins-char/enter ins-mode key
key_il	kil1	kA	KEY_IL, sent by insert-line key
key_left	kcub1	kl	KEY_LEFT, sent by terminal left-arrow key
key_ll	kll	kH	KEY_LL, sent by home-down key
key_mark	kmrk	%2	KEY_MARK, sent by mark key
key_message	kmsg	%3	KEY_MESSAGE, sent by message key
key_move	kmov	%4	KEY_MOVE, sent by move key
key_next	knxt	%5	KEY_NEXT, sent by next-object key
key_npage	knp	kN	KEY_NPAGE, sent by next-page key
key_open	kopn	%6	KEY_OPEN, sent by open key
key_options	kopt	%7	KEY_OPTIONS, sent by options key
key_ppage	kpp	kP	KEY_PPAGE, sent by previous-page key
key_previous	kprv	%8	KEY_PREVIOUS, sent by previous-object key
key_print	kprt	%9	KEY_PRINT, sent by print or copy key

Variable	Cap-name	Termcap Code	Description
key_redo	krdo	%0	KEY_REDO, sent by redo key
key_reference	kref	&1	KEY_REFERENCE, sent by ref(erence) key
key_refresh	krfr	&2	KEY_REFRESH, sent by refresh key
key_replace	krpl	&3	KEY_REPLACE, sent by replace key
key_restart	krst	&4	KEY_RESTART, sent by restart key
key_resume	kres	&5	KEY_RESUME, sent by resume key
key_right	kcuf1	kr	KEY_RIGHT, sent by terminal right-arrow key
key_save	ksav	&6	KEY_SAVE, sent by save key
key_sbeg	kBEG	&9	KEY_SBEG, sent by shifted beginning key
key_scancel	kCAN	&0	KEY_SCANCEL, sent by shifted cancel key
key_scommand	kCMD	*1	KEY_SCOMMAND, sent by shifted command key
key_scopy	kCPY	*2	KEY_SCOPY, sent by shifted copy key
key_screate	kCRT	*3	KEY_SCREATE, sent by shifted create key
key_sdc	kDC	*4	KEY_SDC, sent by shifted delete-char key
key_sdl	kDL	*5	KEY_SDL, sent by shifted delete-line key
key_select	kslt	*6	KEY_SELECT, sent by select key
key_send	kEND	*7	KEY_SEND, sent by shifted end key
key_seol	kEOL	*8	KEY_SEOL, sent by shifted clear-line key
key_sexit	kEXT	*9	KEY_SEXIT, sent by shifted exit key
key_sf	kind	kF	KEY_SF, sent by scroll-forward/down key
key_sfind	kFND	*0	KEY_SFIND, sent by shifted find key
key_shelp	kHLP	#1	KEY_SHELP, sent by shifted help key
key_shome	kHOM	#2	KEY_SHOME, sent by shifted home key
key_sic	kIC	#3	KEY_SIC, sent by shifted input key
key_sleft	kLFT	#4	KEY_SLEFT, sent by shifted left-arrow key
key_smessage	kMSG	%a	KEY_SMESSAGE, sent by shifted message key
key_smove	kMOV	%b	KEY_SMOVE, sent by shifted move key
key_snext	kNXT	%c	KEY_SNEXT, sent by shifted next key
key_soptions	kOPT	%d	KEY_SOPTIONS, sent by shifted options key
key_sprevious	kPRV	%e	KEY_SPREVIOUS, sent by shifted prev key
key_sprint	kPRT	%f	KEY_SPRINT, sent by shifted print key
key_sr	kri	kR	KEY_SR, sent by scroll-backward/up key
key_sredo	kRDO	%g	KEY_SREDO, sent by shifted redo key
key_sreplace	kRPL	%h	KEY_SREPLACE, sent by shifted replace key
key_sright	kRIT	%i	KEY_SRIGHT, sent by shifted

Variable	Cap-name	Termcap Code	Description
			right-arrow key
key_srsume	kRES	%j	KEY_SRSUME, sent by shifted resume key
key_ssave	kSAV	!1	KEY_SSAVE, sent by shifted save key
key_ssuspend	kSPD	!2	KEY_SSUSPEND, sent by shifted suspend key
key_stab	khts	kT	KEY_STAB, sent by set-tab key
key_sundo	kUND	!3	KEY_SUNDO, sent by shifted undo key
key_suspend	kspd	&7	KEY_SUSPEND, sent by suspend key
key_undo	kund	&8	KEY_UNDO, sent by undo key
key_up	kcuu1	ku	KEY_UP, sent by terminal up-arrow key
keypad_local	rmkx	ke	Out of "keypad-transmit" mode
keypad_xmit	smkx	ks	Put terminal in "keypad-transmit" mode
lab_f0	lf0	l0	Labels on function key f0 if not f0
lab_f1	lf1	l1	Labels on function key f1 if not f1
lab_f2	lf2	l2	Labels on function key f2 if not f2
lab_f3	lf3	l3	Labels on function key f3 if not f3
lab_f4	lf4	l4	Labels on function key f4 if not f4
lab_f5	lf5	l5	Labels on function key f5 if not f5
lab_f6	lf6	l6	Labels on function key f6 if not f6
lab_f7	lf7	l7	Labels on function key f7 if not f7
lab_f8	lf8	l8	Labels on function key f8 if not f8
lab_f9	lf9	l9	Labels on function key f9 if not f9
lab_f10	lf10	la	Labels on function key f10 if not f10
label_off	rmln	LF	Turn off soft labels
label_on	smln	LO	Turn on soft labels
meta_off	rmm	mo	Turn off "meta mode"
meta_on	smm	mm	Turn on "meta mode" (8th bit)
micro_column_address	mhpa	ZY	Like column_address for micro adjustment
micro_down	mcud1	ZZ	Like cursor_down for micro adjustment
micro_left	mcub1	Za	Like cursor_left for micro adjustment
micro_right	mcuf1	Zb	Like cursor_right for micro adjustment
micro_row_address	mvpa	Zc	Like row_address for micro adjustment
micro_up	mcuu1	Zd	Like cursor_up for micro adjustment
newline	nel	nw	Newline (behaves like cr followed by lf)
order_of_pins	porder	Ze	Matches software bits to print-head pins
orig_colors	oc	oc	Set all color(-pair)s to the original ones
orig_pair	op	op	Set default color-pair to the original one
pad_char	pad	pc	Pad character (rather than null)
parm_dch	dch	DC	Delete #1 chars

Variable	Cap-name	Termcap Code	Description
parm_delete_line	dl	DL	Delete #1 lines
parm_down_cursor	cud	DO	Move down #1 lines.
parm_down_micro	mcud	Zf	Like parm_down_cursor for micro adjust.
parm_ich	ich	IC	Insert #1 blank chars
parm_index	indn	SF	Scroll forward #1 lines.
parm_insert_line	il	AL	Add #1 new blank lines
parm_left_cursor	cub	LE	Move cursor left #1 spaces
parm_left_micro	mcub	Zg	Like parm_left_cursor for micro adjust.
parm_right_cursor	cuf	RI	Move right #1 spaces.
parm_right_micro	mcuf	Zh	Like parm_right_cursor for micro adjust.
parm_rindex	rin	SR	Scroll backward #1 lines.
parm_up_cursor	cuu	UP	Move cursor up #1 lines.
parm_up_micro	mcuu	Zi	Like parm_up_cursor for micro adjust.
pc_term_options	pctrm	S6	PC terminal options
pkey_key	pfkey	pk	Prog funct key #1 to type string #2
pkey_local	pfloc	pl	Prog funct key #1 to execute string #2
pkey_plab	pfxl	xl	Prog key #1 to xmit string #2 and show string #3
pkey_xmit	pfx	px	Prog funct key #1 to xmit string #2
plab_norm	pln	pn	Prog label #1 to show string #2
print_screen	mc0	ps	Print contents of the screen
prtr_non	mc5p	pO	Turn on the printer for #1 bytes
prtr_off	mc4	pf	Turn off the printer
prtr_on	mc5	po	Turn on the printer
repeat_char	rep	rp	Repeat char #1 #2 times
req_for_input	rfi	RF	Send next input char (for ptys)
reset_1string	rs1	r1	Reset terminal completely to sane modes
reset_2string	rs2	r2	Reset terminal completely to sane modes
reset_3string	rs3	r3	Reset terminal completely to sane modes
reset_file	rf	rf	Name of file containing reset string
restore_cursor	rc	rc	Restore cursor to position of last sc
row_address	vpa	cv	Vertical position absolute
save_cursor	sc	sc	Save cursor position
scancode_escape	scesc	S7	Escape for scancode emulation
scroll_forward	ind	sf	Scroll text up
scroll_reverse	ri	sr	Scroll text down
select_char_set	scs	Zj	Select character set
set0_des_seq	s0ds	s0	Shift into codeset 0 (EUC set 0, ASCII)
set1_des_seq	s1ds	s1	Shift into codeset 1
set2_des_seq	s2ds	s2	Shift into codeset 2
set3_des_seq	s3ds	s3	Shift into codeset 3
set_a_background	setab	AB	Set background color using ANSI escape

Variable	Cap-name	Termcap Code	Description
set_a_foreground	setaf	AF	Set foreground color using ANSI escape
set_attributes	sgr	sa	Define the video attributes #1-#9
set_background	setb	Sb	Set current background color
set_bottom_margin	smgb	Zk	Set bottom margin at current line
set_bottom_margin_parm	smgbp	Zl	Set bottom margin at line #1 or #2 lines from bottom
set_color_band	setcolor	Yz	Change to ribbon color #1
set_color_pair	scp	sp	Set current color-pair
set_foreground	setf	Sf	Set current foreground color1
set_left_margin	smgl	ML	Set left margin at current line
set_left_margin_parm	smglp	Zm	Set left (right) margin at column #1 (#2)
set_lr_margin	smglr	ML	Sets both left and right margins
set_page_length	slines	YZ	Set page length to #1 lines (use tparm)
set_right_margin	smgr	MR	Set right margin at current column
set_right_margin_parm	smgrp	Zn	Set right margin at column #1
set_tab	hts	st	Set a tab in all rows, current column
set_tb_margin	smgtb	MT	Sets both top and bottom margins
set_top_margin	smgt	Zo	Set top margin at current line
set_top_margin_parm	smgtp	Zp	Set top (bottom) margin at line #1 (#2)
set_window	wind	wi	Current window is lines #1-#2 cols #3-#4
start_bit_image	sbim	Zq	Start printing bit image graphics
start_char_set_def	scsd	Zr	Start definition of a character set
stop_bit_image	rbim	Zs	End printing bit image graphics
stop_char_set_def	rcsd	Zt	End definition of a character set
subscript_characters	subcs	Zu	List of "subscript-able" characters
superscript_characters	supcs	Zv	List of "superscript-able" characters
tab	ht	ta	Tab to next 8-space hardware tab stop
these_cause_cr	docr	Zw	Printing any of these chars causes cr
to_status_line	tsl	ts	Go to status line, col #1
underline_char	uc	uc	Underscore one char and move past it
up_half_line	hu	hu	Half-line up (reverse 1/2 linefeed)
xoff_character	xoffc	XF	X-off character
xon_character	xonc	XN	X-on character
zero_motion	zerom	Zx	No motion for the subsequent character

Sample Entry

The following entry, which describes the AT&T 610 terminal, is among the more complex entries in the terminfo file as of this writing.

```
610|610bct|ATT610|att610|AT&T610;80column;98key keyboard
    am, eslok, hs, mir, msgr, xenl, xon,
    cols#80, it#8, lh#2, lines#24, lw#8, nlab#8, wsl#80,
    acsc=''aaffggjjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{||}}~~,
    bel=^G, blink=\E[5m, bold=\E[1m, cbt=\E[Z,
    civis=\E[?25l, clear=\E[H\E[J, cnorm=\E[?25h\E[?12l,
    cr=\r, csr=\E[%i%p1%d;%p2%dr, cub=\E[%p1%dD, cub1=\b,
    cud=\E[%p1%dB, cud1=\E[B, cuf=\E[%p1%dC, cuf1=\E[C,
    cup=\E[%i%p1%d;%p2%dH, cuu=\E[%p1%dA, cuu1=\E[A,
```

```
    cvvis=\E[?12;25h, dch=\E[%p1%dP, dch1=\E[P, dim=\E[2m,
    dl=\E[%p1%dM, dl1=\E[M, ed=\E[J, el=\E[K, el1=\E[1K,
    flash=\E[?5h$<200>\E[?5l, fsl=\E8, home=\E[H, ht=\t,
    ich=\E[%p1%d@, il=\E[%p1%dL, il1=\E[L, ind=\ED, .ind=\ED$<9>,
    invis=\E[8m,
    is1=\E[8;0 | \E[?3;4;5;13;15l\E[13;20l\E[?7h\E[12h\E(B\E)0,
    is2=\E[0m^O, is3=\E(B\E)0, kLFT=\E[\s@, kRIT=\E[\sA,
    kbs=^H, kcbt=\E[Z, kclr=\E[2J, kcub1=\E[D, kcud1=\E[B,
    kcuf1=\E[C, kcuu1=\E[A, kf1=\EOc, kf10=\ENp,
    kf11=\ENq, kf12=\ENr, kf13=\ENs, kf14=\ENt, kf2=\EOd,
    kf3=\EOe, kf4=\EOf, kf5=\EOg, kf6=\EOh, kf7=\EOi,
    kf8=\EOj, kf9=\ENo, khome=\E[H, kind=\E[S, kri=\E[T,
    ll=\E[24H, mc4=\E[?4i, mc5=\E[?5i, nel=\EE,
    pfxl=\E[%p1%d;%p2%l%02dq%?%p1%{9}%<%t\s\s\sF%p1%1d\s\s\s\s\s
\s\s\s\s\s\s%;%p2%s,
    pln=\E[%p1%d;0;0;0q%p2%:-16.16s, rc=\E8, rev=\E[7m,
    ri=\EM, rmacs=^O, rmir=\E[4l, rmln=\E[2p, rmso=\E[m,
    rmul=\E[m, rs2=\Ec\E[?3l, sc=\E7,
    sgr=\E[0%?%p6%t;1%;%?%p5%t;2%;%?%p2%t;4%;%?%p4%t;5%;
%?%p3%p1% | %t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,
    sgr0=\E[m^O, smacs=^N, smir=\E[4h, smln=\E[p,
    smso=\E[7m, smul=\E[4m, tsl=\E7\E[25;%i%p1%dx,
```

Types of Capabilities in the Sample Entry

The sample entry shows the formats for the three types of `terminfo` capabilities listed: Boolean, numeric, and string. All capabilities specified in the `terminfo` source file must be followed by commas, including the last capability in the source file. In `terminfo` source files, capabilities are referenced by their capability names (as shown in the previous tables).

Boolean capabilities are specified simply by their comma separated cap names.

Numeric capabilities are followed by the character '#' and then a positive integer value. Thus, in the sample, `cols` (which shows the number of columns available on a device) is assigned the value `80` for the AT&T 610. (Values for numeric capabilities may be specified in decimal, octal, or hexadecimal, using normal C programming language conventions.)

Finally, string-valued capabilities such as `el` (clear to end of line sequence) are listed by a two- to five-character capname, an '=', and a string ended by the next occurrence of a comma. A delay in milliseconds may appear anywhere in such a capability, preceded by `$` and enclosed in angle brackets, as in `el=\EK$<3>`. Padding characters are supplied by `tput`. The delay can be any of the following: a number, a number followed by an asterisk, such as `5*`, a number followed by a slash, such as `5/`, or a number followed by both, such as `5*/`. A '`*`' shows that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert characters, the factor is still the number of lines affected. This is always 1 unless the device has `in` and the software uses it.) When a '`*`' is specified, it is sometimes useful to give a delay of the form `3.5` to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A '/' indicates that the padding is mandatory. If a device has xon defined, the padding information is advisory and will only be used for cost estimates or when the device is in raw mode. Mandatory padding will be transmitted regardless of the setting of xon. If padding (whether advisory or mandatory) is specified for bel or flash, however, it will always be used, regardless of whether xon is specified.

terminfo offers notation for encoding special characters. Both \E and \e map to an ESCAPE character, ^*x* maps to a control *x* for any appropriate *x*, and the sequences \n, \l, \r, \t, \b, \f, and \s give a newline, linefeed, return, tab, backspace, formfeed, and space, respectively. Other escapes include: \^ for caret (^); \\ for backslash (\); \, for comma (,); \: for colon (:); and \0 for null. (\0 will actually produce \200, which does not terminate a string but behaves as a null character on most devices, providing CS7 is specified. [See stty(1).] Finally, characters may be given as three octal digits after a backslash (for example, \123).

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second ind in the example above. Note that capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.

Preparing Descriptions

The most effective way to prepare a device description is by imitating the description of a similar device in terminfo and building up a description gradually, using partial descriptions with vi to check that they are correct. Be aware that a very unusual device may expose deficiencies in the ability of the terminfo file to describe it or the inability of vi to work with that device. To test a new device description, set the environment variable TERMINFO to the pathname of a directory containing the compiled description you are working on and programs will look there rather than in /usr/share/lib/terminfo. To get the padding for insert-line correct (if the device manufacturer did not document it) a severe test is to comment out xon, edit a large file at 9600 baud with vi, delete 16 or so lines from the middle of the screen, and then press the u key several times quickly. If the display is corrupted, more padding is usually needed. A similar test can be used for insert-character.

Section 1-1: Basic Capabilities

The number of columns on each line for the device is given by the cols numeric capability. If the device has a screen, then the number of lines on the screen is given by the lines capability. If the device wraps around to the beginning of the next line when it reaches the right margin, then it should have the am capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the clear string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the os capability. If the device is a printing terminal, with no soft copy unit, specify both hc and os. If there is a way to move the cursor to the left edge of the current row, specify this as cr. (Normally this will be carriage return, control M.) If there is a way to produce an audible signal (such as a bell or a beep), specify it as bel. If, like most devices, the device uses the xon-xoff flow-control protocol, specify xon.

If there is a way to move the cursor one position to the left (such as backspace), that capability should be given as cub1. Similarly, sequences to move to the right, up, and down should be given as cuf1, cuu1, and cud1, respectively. These local cursor motions must not alter the text they pass over; for example, you would not normally use cuf1=\s because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in terminfo are undefined at the left and top edges of a screen terminal. Programs should never attempt to backspace around the left edge, unless bw is specified, and should never attempt to go up locally off the top. To scroll text up, a program goes to the bottom left corner of the screen and sends the ind (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the ri (reverse index) string. The strings ind and ri are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are indn and rin. These versions have the same semantics as ind and ri, except that they take one parameter and scroll the number of lines specified by that parameter. They are also undefined except at the appropriate edge of the screen.

The am capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a cuf1 from the last column. Backward motion from the left edge of the screen is possible only when bw is specified. In this case, cub1 will move to the right edge of the previous row. If bw is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the device has switch selectable automatic margins, am should be specified in the terminfo source file. In this case, initialization strings should turn on this option, if possible. If the device has a command that moves to the first column of the next line, that command can be given as nel (newline). It does not matter if the command clears the remainder of the current line, so if the device has no cr and lf it may still be possible to craft a working nel out of one or both of them.

These capabilities suffice to describe hardcopy and screen terminals. Thus the AT&T 5320 hardcopy terminal is described as follows:

```
5320|att5320|AT&T 5320 hardcopy terminal,
    am, hc, os,
    cols#132,
    bel=^G, cr=\r, cub1=\b, cnd1=\n,
    dch1=\E[P, dl1=\E[M,
    ind=\n,
```

while the Lear Siegler ADM-3 is described as

```
adm3 | lsi adm3,
  am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H,
  cud1=^J, ind=^J, lines#24,
```

Section 1-2: Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with printf-like escapes (%*x*) in it. For example, to address the cursor, the cup capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by mrcup.

The parameter mechanism uses a stack and special `%` codes to manipulate the stack in the manner of Reverse Polish Notation (postfix). Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary. Operations are in postfix form with the operands in the usual order. That is, to subtract 5 from the first parameter, one would use `%p1%{5}%-`.

The `%` encodings have the following meanings:

`%%` outputs '%'

`%[[:]`*flags*][*width*[.*precision*]][`doxXs`]
: as in `printf`, flags are `[-+#]` and space

`%c` print pop gives %c

`%p[1-9]`
: push *i*th parm

`%P[a-z]`
: set dynamic variable [a-z] to pop

`%g[a-z]`
: get dynamic variable [a-z] and push it

`%P[A-Z]`
: set static variable [a-z] to pop

`%g[A-Z]`
: get static variable [a-z] and push it

`%'`*c*`'` push char constant *c*

`%{`*nn*`}` push decimal constant *nn*

`%l` push strlen(pop)

`%+ %- %* %/ %m`
: arithmetic (`%m` is mod): push(pop $integer_2$ op pop $integer_1$)

`%& %| %^`
: bit operations: push(pop $integer_2$ op pop $integer_1$)

`%= %> %<`
: logical operations: push(pop $integer_2$ op pop $integer_1$)

`%A %O` logical operations: and, or

`%! %~` unary operations: push(op pop)

`%i` (for ANSI terminals) add 1 to first parm, if one parm present, or first two parms, if more than one parm present

`%?` *expr* `%t` *thenpart* `%e` *elsepart* `%;`
: if-then-else, `%e` *elsepart* is optional; else-if's are possible ala Algol 68: `%?` c_1 `%t` b_1 `%e` c_2 `%t` b_2 `%e` c_3 `%t` b_3 `%e` c_4 `%t` b_4 `%e` b_5`%;`
c_i are conditions, b_i are bodies.

If the "-" flag is used with "%[doxXs]", then a colon (`:`) must be placed between the "%" and the "-" to differentiate the flag from the binary "%-" operator, for example, "`%:-16.16s`".

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent `\E&a12c03Y` padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are zero-padded as two digits. Thus its `cup` capability is:

```
cup=\E&a%p2%2.2dc%p1%2.2dY$<6>
```

The Micro-Term ACT-IV needs the current row and column sent preceded by a `^T`, with the row and column simply encoded in binary, "`cup=^T%p1%c%p2%c`". Devices that use "`%c`" need to be able to backspace the cursor (`cub1`), and to move the cursor up one line on the screen (`cuu1`). This is necessary because it is not always safe to transmit `\n`, `^D`, and `\r`, as the system may change or discard them. (The library routines dealing with `terminfo` set tty modes so that tabs are never expanded, so `\t` is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus "`cup=\E=%p1%'\s'%+%c%p2%'\s'%+%c`". After sending "`\E=`", this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

Section 1-3: Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as `home`; similarly a fast way of getting to the lower left-hand corner can be given as `ll`; this may involve going up with `cuu1` from the home position, but a program should never do this itself (unless `ll` does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the `\EH` sequence on Hewlett-Packard terminals cannot be used for `home` without losing some of the other features on the terminal.)

If the device has row or column absolute-cursor addressing, these can be given as single parameter capabilities `hpa` (horizontal position absolute) and `vpa` (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to `cup`. If there are parameterized local motions (for example, move *n* spaces to the right) these can be given as `cud`, `cub`, `cuf`, and `cuu` with a single parameter indicating how many spaces to move. These are primarily useful if the device does not have `cup`, such as the Tektronix 4025.

If the device needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as `smcup` and `rmcup`. This arises, for example, from terminals, such as the Concept, with more than one page of memory. If the device has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the device for cursor addressing to work properly. This is also used for the Tektronix 4025, where `smcup` sets the command character to be the one used by `terminfo`. If the `smcup` sequence will not restore the screen after an `rmcup` sequence is output (to the state prior to outputting `rmcup`), specify `nrrmc`.

Section 1-4: Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as `el`. If the terminal can clear from the beginning of the line to the current position inclusive, leaving the cursor where it is, this should be given as `el1`. If the terminal can clear from the current position to the end of the display, then this should be given as `ed`. `ed` is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true `ed` is not available.)

Section 1-5: Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as `il1`; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as `dl1`; this is done only from the first position on the line to be deleted. Versions of `il1` and `dl1` which take a single parameter and insert or delete that many lines can be given as `il` and `dl`.

If the terminal has a settable destructive scrolling region (like the VT100) the command to set this can be described with the `csr` capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command — the `sc` and `rc` (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using `ri` or `ind` on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

To determine whether a terminal has destructive scrolling regions or non-destructive scrolling regions, create a scrolling region in the middle of the screen, place data on the bottom line of the scrolling region, move the cursor to the top line of the scrolling region, and do a reverse index (`ri`) followed by a delete line (`dl1`) or index (`ind`). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by the `dl1` or `ind`, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify `csr` if the terminal has non-destructive scrolling regions, unless `ind`, `ri`, `indn`, `rin`, `dl`, and `dl1` all simulate destructive scrolling.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string `wind`. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the `da` capability should be given; if display memory can be retained below, then `db` should be given. These indicate that deleting a line or scrolling a full screen may bring non-blank lines up from below or that scrolling back with `ri` may bring down non-blank lines.

Section 1-6: Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character operations which can be described using `terminfo`. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type "abc def" using local cursor motions (not spaces) between the abc and the def. Then position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the abc shifts over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability in, which stands for "insert null." While these are two logically separate attributes (one line versus multiline insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

terminfo can describe both terminals that have an insert mode and terminals which send a simple sequence to open a blank position on the current line. Give as smir the sequence to get into insert mode. Give as rmir the sequence to leave insert mode. Now give as ich1 any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give ich1; terminals that send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to ich1. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds padding in ip (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in ip. If your terminal needs both to be placed into an 'insert mode' and a special code to precede each inserted character, then both smir/rmir and ich1 can be given, and both will be used. The ich capability, with one parameter, *n*, will insert *n* blanks.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in rmp.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (for example, if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability mir to speed up inserting in this case. Omitting mir will affect only speed. Some terminals (notably Datamedia's) must not have mir because of the way their insert mode works.

Finally, you can specify dch1 to delete a single character, dch with one parameter, *n*, to delete *n* characters, and delete mode by giving smdc and rmdc to enter and exit delete mode (any mode the terminal needs to be placed in for dch1 to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as ech with one parameter.

Section 1-7: Highlighting, Underlining, and Visible Bells

Your device may have one or more kinds of display attributes that allow you to highlight selected characters when they appear on the screen. The following display modes (shown with the names by which they are set) may be available: a blinking screen (blink), bold or extra-bright characters (bold), dim or half-bright characters (dim), blanking or invisible text (invis), protected text (prot), a reverse-video screen (rev), and an alternate character set (smacs to enter this mode and rmacs to exit it). (If a command is necessary before you can enter alternate character set mode, give the sequence in enacs or "enable alternate-character-set" mode.) Turning on any of these modes singly may or may not turn off other modes.

sgr0 should be used to turn off all video enhancement capabilities. It should always be specified because it represents the only way to turn off some capabilities, such as dim or blink.

You should choose one display method as *standout mode* [see curses(3X)] and use it to highlight error messages and other kinds of text to which you want to draw attention. Choose a form of display that provides strong contrast but that is easy on the eyes. (We recommend reverse-video plus half-bright or reverse-video alone.) The sequences to enter and exit standout mode are given as smso and rmso, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then xmc should be given to tell how many spaces are left.

Sequences to begin underlining and end underlining can be specified as smul and rmul , respectively. If the device has a sequence to underline the current character and to move the cursor one space to the right (such as the Micro-Term MIME), this sequence can be specified as uc.

Terminals with the "magic cookie" glitch (xmc) deposit special "cookies" when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the msgr capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), then this can be given as flash; it must not move the cursor. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as cvvis. The boolean chts should also be given. If there is a way to make the cursor completely invisible, give that as civis. The capability cnorm should be given which undoes the effects of either of these modes.

If your terminal generates underlined characters by using the underline character (with no special sequences needed) even though it does not otherwise overstrike characters, then you should specify the capability ul. For devices on which a character overstriking another leaves both characters on the screen, specify the capability os. If overstrikes are erasable with a blank, then this should be indicated by specifying eo.

If there is a sequence to set arbitrary combinations of modes, this should be given as sgr (set attributes), taking nine parameters. Each parameter is either 0 or nonzero, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need to be supported by sgr; only those for which corresponding separate attribute commands exist should be supported. For example, let's assume that the terminal in question needs the following escape sequences to turn on various modes.

tparm Parameter	Attribute	Escape Sequence
	none	`\E[0m`
p1	standout	`\E[0;4;7m`
p2	underline	`\E[0;3m`
p3	reverse	`\E[0;4m`
p4	blink	`\E[0;5m`
p5	dim	`\E[0;7m`
p6	bold	`\E[0;3;4m`
p7	invis	`\E[0;8m`
p8	protect	not available
p9	altcharset	`^O` (off) `^N` (on)

Note that each escape sequence requires a `0` to turn off other modes before turning on its own mode. Also note that, as suggested above, *standout* is set up to be the combination of *reverse* and *dim*. Also, because this terminal has no *bold* mode, *bold* is set up as the combination of *reverse* and *underline*. In addition, to allow combinations, such as *underline+blink*, the sequence to use would be `\E[0;3;5m`. The terminal doesn't have *protect* mode, either, but that cannot be simulated in any way, so `p8` is ignored. The *altcharset* mode is different in that it is either `^O` or `^N`, depending on whether it is off or on. If all modes were to be turned on, the sequence would be `\E[0;3;4;5;7;8m^N`.

Now look at when different sequences are output. For example, `;3` is output when either `p2` or `p6` is true, that is, if either *underline* or *bold* modes are turned on. Writing out the above sequences, along with their dependencies, gives the following:

Sequence	When to Output	terminfo Translation
`\E[0`	always	`\E[0`
`;3`	if `p2` or `p6`	`%?%p2%p6%|%t;3%;`
`;4`	if `p1` or `p3` or `p6`	`%?%p1%p3%|%p6%|%t;4%;`
`;5`	if `p4`	`%?%p4%t;5%;`
`;7`	if `p1` or `p5`	`%?%p1%p5%|%t;7%;`
`;8`	if `p7`	`%?%p7%t;8%;`
`m`	always	`m`
`^N` or `^O`	if `p9` `^N`, else `^O`	`%?%p9%t^N%e^O%;`

Putting this all together into the `sgr` sequence gives:

```
sgr=\E[0%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%
        |%t;4%;%?%p5%t;5%;%?%p1%p5%
        |%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,
```

Remember that `sgr` and `sgr0` must always be specified.

Section 1-8: Keypad

If the device has a keypad that transmits sequences when the keys are pressed, this information can also be specified. Note that it is not possible to handle devices where the keypad only works in local (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, specify these sequences as `smkx` and `rmkx`. Otherwise the keypad is assumed to always transmit.

The sequences sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as kcub1, kcuf1, kcuu1, kcud1, and khome, respectively. If there are function keys such as f0, f1, ..., f63, the sequences they send can be specified as kf0, kf1, ..., kf63. If the first 11 keys have labels other than the default f0 through f10, the labels can be given as lf0, lf1, ..., lf10. The codes transmitted by certain other special keys can be given: kll (home down), kbs (backspace), ktbc (clear all tabs), kctab (clear the tab stop in this column), kclr (clear screen or erase key), kdch1 (delete character), kdl1 (delete line), krmir (exit insert mode), kel (clear to end of line), ked (clear to end of screen), kich1 (insert character or enter insert mode), kil1 (insert line), knp (next page), kpp (previous page), kind (scroll forward/down), kri (scroll backward/up), khts (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as ka1, ka3, kb2, kc1, and kc3. These keys are useful when the effects of a 3 by 3 directional pad are needed. Further keys are defined above in the capabilities list.

Strings to program function keys can be specified as pfkey, pfloc, and pfx. A string to program screen labels should be specified as pln. Each of these strings takes two parameters: a function key identifier and a string to program it with. pfkey causes pressing the given key to be the same as the user typing the given string; pfloc causes the string to be executed by the terminal in local mode; and pfx causes the string to be transmitted to the computer. The capabilities nlab, lw and lh define the number of programmable screen labels and their width and height. If there are commands to turn the labels on and off, give them in smln and rmln. smln is normally output after one or more pln sequences to make sure that the change becomes visible.

Section 1-9: Tabs and Initialization

If the device has hardware tabs, the command to advance to the next tab stop can be given as ht (usually control I). A "backtab" command that moves leftward to the next tab stop can be given as cbt. By convention, if tty modes show that tabs are being expanded by the computer rather than being sent to the device, programs should not use ht or cbt (even if they are present) because the user may not have the tab stops properly set. If the device has hardware tabs that are initially set every *n* spaces when the device is powered up, the numeric parameter it is given, showing the number of spaces the tabs are set to. This is normally used by tput init [see tput(1)] to determine whether to set the mode for hardware tab expansion and whether to set the tab stops. If the device has tab stops that can be saved in nonvolatile memory, the terminfo description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as tbc (clear all tab stops) and hts (set a tab stop in the current column of every row).

Other capabilities include: is1, is2, and is3, initialization strings for the device; iprog, the path name of a program to be run to initialize the device; and if, the name of a file containing long initialization strings. These strings are expected to set the device into modes consistent with the rest of the terminfo description. They must be sent to the device each time the user logs in and be output in the following order: run the program iprog; output is1; output is2; set the margins using mgc, smgl and smgr; set the tabs using tbc and hts; print the file if; and finally output is3. This is usually done using the init option of tput.

Most initialization is done with is2. Special device modes can be set up without duplicating strings by putting the common sequences in is2 and special cases in is1 and is3. Sequences that do a reset from a totally unknown state can be given as rs1, rs2, rf, and rs3, analogous to is1, is2, is3, and if. (The method using files, if and rf, is used for a few terminals, from /usr/share/lib/tabset/*; however, the recommended method is to use the initialization and reset strings.) These strings are output by tput reset, which is used when the terminal gets into a wedged state. Commands are normally placed in rs1, rs2, rs3, and rf only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set a terminal into 80-column mode would normally be part of is2, but on some terminals it causes an annoying glitch on the screen and is not normally needed because the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tabs than can be described by using tbc and hts, the sequence can be placed in is2 or if.

Any margin can be cleared with mgc. (For instructions on how to specify commands to set and clear margins, see "Margins" below under "Printer Capabilities.")

Section 1-10: Delays

Certain capabilities control padding in the tty driver. These are primarily needed by hard-copy terminals, and are used by tput init to set tty modes appropriately. Delays embedded in the capabilities cr, ind, cub1, ff, and tab can be used to set the appropriate delay bits to be set in the tty driver. If pb (padding baud rate) is given, these values can be ignored at baud rates below the value of pb.

Section 1-11: Status Lines

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit h19's 25th line, or the 24th line of a VT100 which is set to a 23-line scrolling region), the capability hs should be given. Special strings that go to a given column of the status line and return from the status line can be given as tsl and fsl. (fsl must leave the cursor position in the same place it was before tsl. If necessary, the sc and rc strings can be included in tsl and fsl to get this effect.) The capability tsl takes one parameter, which is the column number of the status line the cursor is to be moved to.

If escape sequences and other special commands, such as tab, work while in the status line, the flag eslok can be given. A string which turns off the status line (or otherwise erases its contents) should be given as dsl. If the terminal has commands to save and restore the position of the cursor, give them as sc and rc. The status line is normally assumed to be the same width as the rest of the screen, for example, cols. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter wsl.

Section 1-12: Line Graphics

If the device has a line drawing alternate character set, the mapping of glyph to character would be given in acsc. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

Glyph Name	vt100+ Character
arrow pointing right	+
arrow pointing left	,
arrow pointing down	.
solid square block	0
lantern symbol	I
arrow pointing up	-
diamond	`
checker board (stipple)	a
degree symbol	f
plus/minus	g
board of squares	h
lower right corner	j
upper right corner	k
upper left corner	l
lower left corner	m
plus	n
scan line 1	o
horizontal line	q
scan line 9	s
left tee (├)	t
right tee (┤)	u
bottom tee (┴)	v
top tee (┬)	w
vertical line	x
bullet	~

The best way to describe a new device's line graphics set is to add a third column to the above table with the characters for the new device that produce the appropriate glyph when the device is in the alternate character set mode. For example,

Glyph Name	vt100+ char	New tty char
upper left corner	l	R
lower left corner	m	F
upper right corner	k	T
lower right corner	j	G
horizontal line	q	,
vertical line	x	.

Now write down the characters left to right, as in "acsc=lRmFkTjGq\,x.".

In addition, terminfo allows you to define multiple character sets. See Section 2-5 for details.

Section 1-13: Color Manipulation

Let us define two methods of color manipulation: the Tektronix method and the HP method. The Tektronix method uses a set of N predefined colors (usually 8) from which a user can select "current" foreground and background colors. Thus a terminal can support up to N colors mixed into N*N color-pairs to be displayed on the screen at the same time. When using an HP method the user cannot define the foreground independently of the background, or vice-versa. Instead, the user must define an entire color-pair at once. Up to M color-pairs, made from 2*M different colors, can be defined this way. Most existing color terminals belong to one of these two classes of terminals.

The numeric variables `colors` and `pairs` define the number of colors and color-pairs that can be displayed on the screen at the same time. If a terminal can change the definition of a color (for example, the Tektronix 4100 and 4200 series terminals), this should be specified with `ccc` (can change color). To change the definition of a color (Tektronix 4200 method), use `initc` (initialize color). It requires four arguments: color number (ranging from 0 to `colors`-1) and three RGB (red, green, and blue) values or three HLS colors (Hue, Lightness, Saturation). Ranges of RGB and HLS values are terminal dependent.

Tektronix 4100 series terminals only use HLS color notation. For such terminals (or dual-mode terminals to be operated in HLS mode) one must define a boolean variable `hls`; that would instruct the `curses init_color` routine to convert its RGB arguments to HLS before sending them to the terminal. The last three arguments to the `initc` string would then be HLS values.

If a terminal can change the definitions of colors, but uses a color notation different from RGB and HLS, a mapping to either RGB or HLS must be developed.

To set current foreground or background to a given color, use `setaf` (set ANSI foreground) and `setab` (set ANSI background). They require one parameter: the number of the color. To initialize a color-pair (HP method), use `initp` (initialize pair). It requires seven parameters: the number of a color-pair (range=0 to `pairs`-1), and six RGB values: three for the foreground followed by three for the background. (Each of these groups of three should be in the order RGB.) When `initc` or `initp` are used, RGB or HLS arguments should be in the order "red, green, blue" or "hue, lightness, saturation"), respectively. To make a color-pair current, use `scp` (set color-pair). It takes one parameter, the number of a color-pair.

Some terminals (for example, most color terminal emulators for PCs) erase areas of the screen with current background color. In such cases, `bce` (background color erase) should be defined. The variable `op` (original pair) contains a sequence for setting the foreground and the background colors to what they were at the terminal start-up time. Similarly, `oc` (original colors) contains a control sequence for setting all colors (for the Tektronix method) or color-pairs (for the HP method) to the values they had at the terminal start-up time.

Some color terminals substitute color for video attributes. Such video attributes should not be combined with colors. Information about these video attributes should be packed into the `ncv` (no color video) variable. There is a one-to-one correspondence between the nine least significant bits of that variable and the video attributes. The following table depicts this correspondence.

Attribute	Bit Position	Decimal Value
A_STANDOUT	0	1
A_UNDERLINE	1	2
A_REVERSE	2	4
A_BLINK	3	8
A_DIM	4	16
A_BOLD	5	32
A_INVIS	6	64
A_PROTECT	7	128
A_ALTCHARSET	8	256

When a particular video attribute should not be used with colors, the corresponding ncv bit should be set to 1; otherwise it should be set to zero. To determine the information to pack into the ncv variable, you must add together the decimal values corresponding to those attributes that cannot coexist with colors. For example, if the terminal uses colors to simulate reverse video (bit number 2 and decimal value 4) and bold (bit number 5 and decimal value 32), the resulting value for ncv will be 36 (4 + 32).

Section 1-14: Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as pad. Only the first character of the pad string is used. If the terminal does not have a pad character, specify npc.

If the terminal can move up or down half a line, this can be indicated with hu (half-line up) and hd (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as ff (usually control L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string rep. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, tparm(repeat_char, 'x', 10) is the same as xxxxxxxxxx.

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with cmdch. A prototype command character is chosen which is used in all capabilities. This character is given in the cmdch capability to identify it. The following convention is supported on some UNIX systems: If the environment variable CC exists, all occurrences of the prototype character are replaced with the character in CC.

Terminal descriptions that do not represent a specific kind of known terminal, such as *switch*, *dialup*, *patch*, and *network*, should include the gn (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to *virtual* terminal descriptions for which the escape sequences are known.) If the terminal is one of those supported by the UNIX system virtual terminal protocol, the terminal number can be given as vt. A line-turnaround sequence to be transmitted before doing reads should be specified in rfi.

If the device uses xon/xoff handshaking for flow control, give xon. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted. Sequences to turn on and off xon/xoff handshaking may be given in smxon and rmxon. If the characters used for handshaking are not ^S and ^Q, they may be specified with xonc and xoffc.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with km. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as smm and rmm.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with lm. A value of lm#0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

Media copy strings which control an auxiliary printer connected to the terminal can be given as mc0: print the contents of the screen, mc4: turn off the printer, and mc5: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. A variation, mc5p, takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify mc5i (silent printer). All text, including mc4, is transparently passed to the printer while an mc5p is in effect.

Section 1-15: Special Cases

The working model used by terminfo fits most terminals reasonably well. However, some terminals do not completely match that model, requiring special support by terminfo. These are not meant to be construed as deficiencies in the terminals; they are just differences between the working model and the actual hardware. They may be unusual devices or, for some reason, do not have all the features of the terminfo model implemented.

Terminals that cannot display tilde (~) characters, such as certain Hazeltine terminals, should indicate hz.

Terminals that ignore a linefeed immediately after an am wrap, such as the Concept 100, should indicate xenl. Those terminals whose cursor remains on the right-most column until another character has been received, rather than wrapping immediately upon receiving the right-most character, such as the VT100, should also indicate xenl.

If el is required to get rid of standout (instead of writing normal text on top of it), xhp should be given.

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate xt (destructive tabs). This capability is also taken to mean that it is not possible to position the cursor on top of a "magic cookie." Therefore, to erase standout mode, it is necessary, instead, to use delete and insert line.

Those Beehive Superbee terminals which do not transmit the escape or control-C characters, should specify xsb, indicating that the f1 key is to be used for escape and the f2 key for control C.

Section 1-16: Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability `use` can be given with the name of the similar terminal. The capabilities given before `use` override those in the terminal type invoked by `use`. A capability can be canceled by placing *xx*@ to the left of the capability definition, where *xx* is the capability. For example, the entry

```
att4424-2|Teletype 4424 in display function group ii,
    rev@, sgr@, smul@, use=att4424,
```

defines an AT&T 4424 terminal that does not have the `rev`, `sgr`, and `smul` capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one `use` capability may be given.

PART 2: PRINTER CAPABILITIES

The `terminfo` database allows you to define capabilities of printers as well as terminals. To find out what capabilities are available for printers as well as for terminals, see the two lists under "Device Capabilities" that list capabilities by variable and by capability name.

Section 2-1: Rounding Values

Because parameterized string capabilities work only with integer values, we recommend that `terminfo` designers create strings that expect numeric values that have been rounded. Application designers should note this and should always round values to the nearest integer before using them with a parameterized string capability.

Section 2-2: Printer Resolution

A printer's resolution is defined to be the smallest spacing of characters it can achieve. In general printers have independent resolution horizontally and vertically. Thus the vertical resolution of a printer can be determined by measuring the smallest achievable distance between consecutive printing baselines, while the horizontal resolution can be determined by measuring the smallest achievable distance between the left-most edges of consecutive printed, identical, characters.

All printers are assumed to be capable of printing with a uniform horizontal and vertical resolution. The view of printing that `terminfo` currently presents is one of printing inside a uniform matrix: All characters are printed at fixed positions relative to each "cell" in the matrix; furthermore, each cell has the same size given by the smallest horizontal and vertical step sizes dictated by the resolution. (The cell size can be changed as will be seen later.)

Many printers are capable of "proportional printing," where the horizontal spacing depends on the size of the character last printed. `terminfo` does not make use of this capability, although it does provide enough capability definitions to allow an application to simulate proportional printing.

A printer must not only be able to print characters as close together as the horizontal and vertical resolutions suggest, but also of "moving" to a position an integral multiple of the smallest distance away from a previous position. Thus printed characters can be spaced apart a distance that is an integral multiple of the smallest distance, up to the length or width of a single page.

Some printers can have different resolutions depending on different "modes." In "normal mode," the existing `terminfo` capabilities are assumed to work on columns and lines, just like a video terminal. Thus the old `lines` capability would give the length of a page in lines, and the `cols` capability would give the width of a page in columns. In "micro mode," many `terminfo` capabilities work on increments of lines and columns. With some printers the micro mode may be concomitant with normal mode, so that all the capabilities work at the same time.

Section 2-3: Specifying Printer Resolution

The printing resolution of a printer is given in several ways. Each specifies the resolution as the number of smallest steps per distance:

Specification of Printer Resolution

Characteristic	Number of Smallest Steps
`orhi`	Steps per inch horizontally
`orvi`	Steps per inch vertically
`orc`	Steps per column
`orl`	Steps per line

When printing in normal mode, each character printed causes movement to the next column, except in special cases described later; the distance moved is the same as the per-column resolution. Some printers cause an automatic movement to the next line when a character is printed in the rightmost position; the distance moved vertically is the same as the per-line resolution. When printing in micro mode, these distances can be different, and may be zero for some printers.

Specification of Printer Resolution

Automatic Motion after Printing	
Normal Mode:	
`orc`	Steps moved horizontally
`orl`	Steps moved vertically
Micro Mode:	
`mcs`	Steps moved horizontally
`mls`	Steps moved vertically

Some printers are capable of printing wide characters. The distance moved when a wide character is printed in normal mode may be different from when a regular width character is printed. The distance moved when a wide character is printed in micro mode may also be different from when a regular character is printed in micro mode, but the differences are assumed to be related: If the distance moved for a regular character is the same whether in normal mode or micro mode (`mcs=orc`), then the distance moved for a wide character is also the same whether in normal mode or micro mode. This doesn't mean the normal character distance is necessarily the same as the wide character distance, just that the distances don't change with a change in normal to micro mode. However, if the distance moved for a regular character is different in micro mode from the distance moved in normal mode (`mcs<orc`), the micro mode distance is assumed to be the same for a wide character printed in micro mode, as the table below shows.

Specification of Printer Resolution Automatic Motion after Printing Wide Character	
Normal Mode or Micro Mode (mcs = orc):	
widcs	Steps moved horizontally
Micro Mode (mcs < orc):	
mcs	Steps moved horizontally

There may be control sequences to change the number of columns per inch (the character pitch) and to change the number of lines per inch (the line pitch). If these are used, the resolution of the printer changes, but the type of change depends on the printer:

Specification of Printer Resolution Changing the Character/Line Pitches	
cpi	Change character pitch
cpix	If set, cpi changes orhi, otherwise changes orc
lpi	Change line pitch
lpix	If set, lpi changes orvi, otherwise changes orl
chr	Change steps per column
cvr	Change steps per line

The cpi and lpi string capabilities are each used with a single argument, the pitch in columns (or characters) and lines per inch, respectively. The chr and cvr string capabilities are each used with a single argument, the number of steps per column and line, respectively.

Using any of the control sequences in these strings will imply a change in some of the values of orc, orhi, orl, and orvi. Also, the distance moved when a wide character is printed, widcs, changes in relation to orc. The distance moved when a character is printed in micro mode, mcs, changes similarly, with one exception: if the distance is 0 or 1, then no change is assumed (see items marked with † in the following table).

Programs that use cpi, lpi, chr, or cvr should recalculate the printer resolution (and should recalculate other values see "Effect of Changing Printing Resolution" under "Dot-Mapped Graphics").

Specification of Printer Resolution
Effects of Changing the Character/Line Pitches

Before	*After*
Using cpi with cpix clear:	
orhi´	**orhi**
orc´	$\mathbf{orc}=\dfrac{\mathbf{orhi}}{V_{cpi}}$
Using cpi with cpix set:	

orhi′	$\mathbf{orhi}=\mathbf{orc}\cdot V_{cpi}$
orc′	**orc**
Using `lpi` with `lpix` clear:	
orvi′	**orvi**
orl′	$\mathbf{orl}=\dfrac{\mathbf{orvi}}{V_{lpi}}$
Using `lpi` with `lpix` set:	
orvi′	$\mathbf{orvi}=\mathbf{orl}\cdot V_{lpi}$
orl′	**orl**
Using `chr`:	
orhi′	**orhi**
orc′	V_{chr}
Using `cvr`:	
orvi′	**orvi**
orl′	V_{cvr}
Using `cpi` or `chr`:	
widcs′	$\mathbf{widcs}=\mathbf{widcs}'\dfrac{\mathbf{orc}}{\mathbf{orc}'}$
mcs′	$\mathbf{mcs}=\mathbf{mcs}'\dfrac{\mathbf{orc}}{\mathbf{orc}'}$

V_{cpi}, V_{lpi}, V_{chr}, and V_{cvr} are the arguments used with `cpi`, `lpi`, `chr`, and `cvr`, respectively. The prime marks (′) indicate the old values.

Section 2-4: Capabilities that Cause Movement

In the following descriptions, "movement" refers to the motion of the "current position." With video terminals this would be the cursor; with some printers this is the carriage position. Other printers have different equivalents. In general, the current position is where a character would be displayed if printed.

`terminfo` has string capabilities for control sequences that cause movement a number of full columns or lines. It also has equivalent string capabilities for control sequences that cause movement a number of smallest steps.

String Capabilities for Motion

`mcub1`	Move 1 step left
`mcuf1`	Move 1 step right
`mcuu1`	Move 1 step up
`mcud1`	Move 1 step down
`mcub`	Move *N* steps left
`mcuf`	Move *N* steps right
`mcuu`	Move *N* steps up
`mcud`	Move *N* steps down

mhpa	Move *N* steps from the left
mvpa	Move *N* steps from the top

The latter six strings are each used with a single argument, *N*.

Sometimes the motion is limited to less than the width or length of a page. Also, some printers don't accept absolute motion to the left of the current position. terminfo has capabilities for specifying these limits.

Limits to Motion

mjump	Limit on use of mcub1, mcuf1, mcuu1, mcud1
maddr	Limit on use of mhpa, mvpa
xhpa	If set, hpa and mhpa can't move left
xvpa	If set, vpa and mvpa can't move up

If a printer needs to be in a "micro mode" for the motion capabilities described above to work, there are string capabilities defined to contain the control sequence to enter and exit this mode. A boolean is available for those printers where using a carriage return causes an automatic return to normal mode.

Entering/Exiting Micro Mode

smicm	Enter micro mode
rmicm	Exit micro mode
crxm	Using cr exits micro mode

The movement made when a character is printed in the rightmost position varies among printers. Some make no movement, some move to the beginning of the next line, others move to the beginning of the same line. terminfo has boolean capabilities for describing all three cases.

What Happens After Character Printed in Rightmost Position

sam	Automatic move to beginning of same line

Some printers can be put in a mode where the normal direction of motion is reversed. This mode can be especially useful when there are no capabilities for leftward or upward motion, because those capabilities can be built from the motion reversal capability and the rightward or downward motion capabilities. It is best to leave it up to an application to build the leftward or upward capabilities, though, and not enter them in the terminfo database. This allows several reverse motions to be strung together without intervening wasted steps that leave and reenter reverse mode.

Entering/Exiting Reverse Modes

slm	Reverse sense of horizontal motions
rlm	Restore sense of horizontal motions
sum	Reverse sense of vertical motions
rum	Restore sense of vertical motions

While sense of horizontal motions reversed:

mcub1	Move 1 step right
mcuf1	Move 1 step left
mcub	Move *N* steps right
mcuf	Move *N* steps left
cub1	Move 1 column right
cuf1	Move 1 column left
cub	Move *N* columns right
cuf	Move *N* columns left

While sense of vertical motions reversed:

mcuu1	Move 1 step down
mcud1	Move 1 step up
mcuu	Move *N* steps down
mcud	Move *N* steps up
cuu1	Move 1 line down
cud1	Move 1 line up
cuu	Move *N* lines down
cud	Move *N* lines up

The reverse motion modes should not affect the mvpa and mhpa absolute motion capabilities. The reverse vertical motion mode should, however, also reverse the action of the line "wrapping" that occurs when a character is printed in the right-most position. Thus printers that have the standard terminfo capability am defined should experience motion to the beginning of the previous line when a character is printed in the right-most position under reverse vertical motion mode.

The action when any other motion capabilities are used in reverse motion modes is not defined; thus, programs must exit reverse motion modes before using other motion capabilities.

Two miscellaneous capabilities complete the list of new motion capabilities. One of these is needed for printers that move the current position to the beginning of a line when certain control characters, such as "line-feed" or "form-feed," are used. The other is used for the capability of suspending the motion that normally occurs after printing a character.

Miscellaneous Motion Strings	
docr	List of control characters causing cr
zerom	Prevent auto motion after printing next single character

Margins

terminfo provides two strings for setting margins on terminals: one for the left and one for the right margin. Printers, however, have two additional margins, for the top and bottom margins of each page. Furthermore, some printers require not using motion strings to move the current position to a margin and then fixing the margin there, but require the specification of where a margin should be regardless of the current position. Therefore terminfo offers six additional strings for defining margins with printers.

Setting Margins	
smgl	Set left margin at current column
smgr	Set right margin at current column
smgb	Set bottom margin at current line
smgt	Set top margin at current line
smgbp	Set bottom margin at line *N*
smglp	Set left margin at column *N*
smgrp	Set right margin at column *N*
smgtp	Set top margin at line *N*

The last four strings are used with one or more arguments that give the position of the margin or margins to set. If both of smglp and smgrp are set, each is used with a single argument, *N*, that gives the column number of the left and right margin, respectively. If both of smgtp and smgbp are set, each is used to set the top and bottom margin, respectively: smgtp is used with a single argument, *N*, the line number of the top margin; however, smgbp is used with two arguments, *N* and *M*, that give the line number of the bottom margin, the first counting from the top of the page and the second counting from the bottom. This accommodates the two styles of specifying the bottom margin in different manufacturers' printers. When coding a terminfo entry for a printer that has a settable bottom margin, only the first or second parameter should be used, depending on the printer. When writing an application that uses smgbp to set the bottom margin, both arguments must be given.

If only one of smglp and smgrp is set, then it is used with two arguments, the column number of the left and right margins, in that order. Likewise, if only one of smgtp and smgbp is set, then it is used with two arguments that give the top and bottom margins, in that order, counting from the top of the page. Thus when coding a terminfo entry for a printer that requires setting both left and right or top and bottom margins simultaneously, only one of smglp and smgrp or smgtp and smgbp should be defined; the other should be left blank. When writing an application that uses these string capabilities, the pairs should be first checked to see if each in the pair is set or only one is set, and should then be used accordingly.

In counting lines or columns, line zero is the top line and column zero is the leftmost column. A zero value for the second argument with smgbp means the bottom line of the page.

All margins can be cleared with mgc.

Shadows, Italics, Wide Characters, Superscripts, Subscripts

Five new sets of strings are used to describe the capabilities printers have of enhancing printed text.

Enhanced Printing	
sshm	Enter shadow-printing mode
rshm	Exit shadow-printing mode
sitm	Enter italicizing mode

`ritm`	Exit italicizing mode
`swidm`	Enter wide character mode
`rwidm`	Exit wide character mode
`ssupm`	Enter superscript mode
`rsupm`	Exit superscript mode
`supcs`	List of characters available as superscripts
`ssubm`	Enter subscript mode
`rsubm`	Exit subscript mode
`subcs`	List of characters available as subscripts

If a printer requires the `sshm` control sequence before every character to be shadow-printed, the `rshm` string is left blank. Thus programs that find a control sequence in `sshm` but none in `rshm` should use the `sshm` control sequence before every character to be shadow-printed; otherwise, the `sshm` control sequence should be used once before the set of characters to be shadow-printed, followed by `rshm`. The same is also true of each of the `sitm`/`ritm`, `swidm`/`rwidm`, `ssupm`/`rsupm`, and `ssubm`/ `rsubm` pairs.

Note that `terminfo` also has a capability for printing emboldened text (`bold`). While shadow printing and emboldened printing are similar in that they "darken" the text, many printers produce these two types of print in slightly different ways. Generally, emboldened printing is done by overstriking the same character one or more times. Shadow printing likewise usually involves overstriking, but with a slight movement up and/or to the side so that the character is "fatter."

It is assumed that enhanced printing modes are independent modes, so that it would be possible, for instance, to shadow print italicized subscripts.

As mentioned earlier, the amount of motion automatically made after printing a wide character should be given in `widcs`.

If only a subset of the printable ASCII characters can be printed as superscripts or subscripts, they should be listed in `supcs` or `subcs` strings, respectively. If the `ssupm` or `ssubm` strings contain control sequences, but the corresponding `supcs` or `subcs` strings are empty, it is assumed that all printable ASCII characters are available as superscripts or subscripts.

Automatic motion made after printing a superscript or subscript is assumed to be the same as for regular characters. Thus, for example, printing any of the following three examples will result in equivalent motion:

Bi B_i B^i

Note that the existing `msgr` boolean capability describes whether motion control sequences can be used while in "standout mode." This capability is extended to cover the enhanced printing modes added here. `msgr` should be set for those printers that accept any motion control sequences without affecting shadow, italicized, widened, superscript, or subscript printing. Conversely, if `msgr` is not set, a program should end these modes before attempting any motion.

Section 2-5: Alternate Character Sets

In addition to allowing you to define line graphics (described in Section 1-12), `terminfo` lets you define alternate character sets. The following capabilities cover printers and terminals with multiple selectable or definable character sets.

Alternate Character Sets	
`scs`	Select character set *N*
`scsd`	Start definition of character set *N*, *M* characters
`defc`	Define character *A*, *B* dots wide, descender *D*
`rcsd`	End definition of character set *N*
`csnm`	List of character set names
`daisy`	Printer has manually changed print-wheels

The `scs`, `rcsd`, and `csnm` strings are used with a single argument, *N*, a number from 0 to 63 that identifies the character set. The `scsd` string is also used with the argument *N* and another, *M*, that gives the number of characters in the set. The `defc` string is used with three arguments: *A* gives the ASCII code representation for the character, *B* gives the width of the character in dots, and *D* is zero or one depending on whether the character is a "descender" or not. The `defc` string is also followed by a string of "image-data" bytes that describe how the character looks (see below).

Character set 0 is the default character set present after the printer has been initialized. Not every printer has 64 character sets, of course; using `scs` with an argument that doesn't select an available character set should cause a null result from `tparm`.

If a character set has to be defined before it can be used, the `scsd` control sequence is to be used before defining the character set, and the `rcsd` is to be used after. They should also cause a null result from `tparm` when used with an argument *N* that doesn't apply. If a character set still has to be selected after being defined, the `scs` control sequence should follow the `rcsd` control sequence. By examining the results of using each of the `scs`, `scsd`, and `rcsd` strings with a character set number in a call to `tparm`, a program can determine which of the three are needed.

Between use of the `scsd` and `rcsd` strings, the `defc` string should be used to define each character. To print any character on printers covered by `terminfo`, the ASCII code is sent to the printer. This is true for characters in an alternate set as well as "normal" characters. Thus the definition of a character includes the ASCII code that represents it. In addition, the width of the character in dots is given, along with an indication of whether the character should descend below the print line (such as the lower case letter "g" in most character sets). The width of the character in dots also indicates the number of image-data bytes that will follow the `defc` string. These image-data bytes indicate where in a dot-matrix pattern ink should be applied to "draw" the character; the number of these bytes and their form are defined below under "Dot-Mapped Graphics."

It's easiest for the creator of `terminfo` entries to refer to each character set by number; however, these numbers will be meaningless to the application developer. The `csnm` string alleviates this problem by providing names for each number.

When used with a character set number in a call to `tparm`, the `csnm` string will produce the equivalent name. These names should be used as a reference only. No naming convention is implied, although anyone who creates a `terminfo` entry for a printer should use names consistent with the names found in user documents for the printer. Application developers should allow a user to specify a character set by number (leaving it up to the user to examine the `csnm` string to determine the correct number), or by name, where the application examines the `csnm` string to determine the corresponding character set number.

These capabilities are likely to be used only with dot-matrix printers. If they are not available, the strings should not be defined. For printers that have manually changed print-wheels or font cartridges, the boolean `daisy` is set.

Section 2-6: Dot-Matrix Graphics

Dot-matrix printers typically have the capability of reproducing "raster-graphics" images. Three new numeric capabilities and three new string capabilities can help a program draw raster-graphics images independent of the type of dot-matrix printer or the number of pins or dots the printer can handle at one time.

Dot-Matrix Graphics	
`npins`	Number of pins, *N*, in print-head
`spinv`	Spacing of pins vertically in pins per inch
`spinh`	Spacing of dots horizontally in dots per inch
`porder`	Matches software bits to print-head pins
`sbim`	Start printing bit image graphics, *B* bits wide
`rbim`	End printing bit image graphics

The `sbim` sring is used with a single argument, *B*, the width of the image in dots.

The model of dot-matrix or raster-graphics that `terminfo` presents is similar to the technique used for most dot-matrix printers: each pass of the printer's print-head is assumed to produce a dot-matrix that is *N* dots high and *B* dots wide. This is typically a wide, squat, rectangle of dots. The height of this rectangle in dots will vary from one printer to the next; this is given in the `npins` numeric capability. The size of the rectangle in fractions of an inch will also vary; it can be deduced from the `spinv` and `spinh` numeric capabilities. With these three values an application can divide a complete raster-graphics image into several horizontal strips, perhaps interpolating to account for different dot spacing vertically and horizontally.

The `sbim` and `rbim` strings are used to start and end a dot-matrix image, respectively. The `sbim` string is used with a single argument that gives the width of the dot-matrix in dots. A sequence of "image-data bytes" are sent to the printer after the `sbim` string and before the `rbim` string. The number of bytes is a integral multiple of the width of the dot-matrix; the multiple and the form of each byte is determined by the `porder` string as described below.

The `porder` string is a comma separated list of pin numbers optionally followed by an numerical offset. The offset, if given, is separated from the list with a semicolon. The position of each pin number in the list corresponds to a bit in an 8-bit data byte. The pins are numbered consecutively from 1 to `npins`, with 1 being the top pin. Note that the term "pin" is used loosely here; "ink-jet" dot-matrix printers don't have pins, but can be considered to have an equivalent method of applying a single dot of ink to paper. The bit positions in `porder` are in groups of 8, with the first position in each group the most significant bit and the last position the least significant bit. An application produces 8-bit bytes in the order of the groups in `porder`.

An application computes the "image-data bytes" from the internal image, mapping vertical dot positions in each print-head pass into 8-bit bytes, using a 1 bit where ink should be applied and 0 where no ink should be applied. This can be reversed (0 bit for ink, 1 bit for no ink) by giving a negative pin number. If a position is skipped in `porder`, a 0 bit is used. If a position has a lower case 'x' instead of a pin number, a 1 bit is used in the skipped position. For consistency, a lower case 'o' can be used to represent a 0 filled, skipped bit. There must be a multiple of 8 bit positions used or skipped in `porder`; if not, 0 bits are used to fill the last byte in the least significant bits. The offset, if given, is added to each data byte; the offset can be negative.

Some examples may help clarify the use of the `porder` string. The AT&T 470, AT&T 475 and C.Itoh 8510 printers provide eight pins for graphics. The pins are identified top to bottom by the 8 bits in a byte, from least significant to most. The `porder` strings for these printers would be `8,7,6,5,4,3,2,1`. The AT&T 478 and AT&T 479 printers also provide eight pins for graphics. However, the pins are identified in the reverse order. The `porder` strings for these printers would be `1,2,3,4,5,6,7,8`. The AT&T 5310, AT&T 5320, DEC LA100, and DEC LN03 printers provide six pins for graphics. The pins are identified top to bottom by the decimal values 1, 2, 4, 8, 16 and 32. These correspond to the low six bits in an 8-bit byte, although the decimal values are further offset by the value 63. The `porder` string for these printers would be `,,6,5,4,3,2,1;63`, or alternately `o,o,6,5,4,3,2,1;63`.

Section 2-7: Effect of Changing Printing Resolution

If the control sequences to change the character pitch or the line pitch are used, the pin or dot spacing may change:

Dot-Matrix Graphics

Changing the Character/Line Pitches	
`cpi`	Change character pitch
`cpix`	If set, `cpi` changes `spinh`
`lpi`	Change line pitch
`lpix`	If set, `lpi` changes `spinv`

Programs that use `cpi` or `lpi` should recalculate the dot spacing:

Dot-Matrix Graphics

Effects of Changing the Character/Line Pitches

Before	*After*
Using `cpi` with `cpix` clear:	
spinh′	**spinh**
Using `cpi` with `cpix` set:	
spinh′	$\mathbf{spinh}=\mathbf{spinh}'\cdot\frac{\mathbf{orhi}}{\mathbf{orhi}'}$

Dot-Matrix Graphics Effects of Changing the Character/Line Pitches	
Before	*After*
Using lpi with lpix clear:	
spinv′	**spinv**
Using lpi with lpix set:	
spinv′	$\mathbf{spinv}=\mathbf{spinv}'\cdot\frac{\mathbf{orhi}}{\mathbf{orhi}'}$
Using chr:	
spinh′	**spinh**
Using cvr:	
spinv′	**spinv**

orhi′ and **orhi** are the values of the horizontal resolution in steps per inch, before using cpi and after using cpi, respectively. Likewise, **orvi′** and **orvi** are the values of the vertical resolution in steps per inch, before using lpi and after using lpi, respectively. Thus, the changes in the dots per inch for dot-matrix graphics follow the changes in steps per inch for printer resolution.

Section 2-8: Print Quality

Many dot-matrix printers can alter the dot spacing of printed text to produce near "letter quality" printing or "draft quality" printing. Usually it is important to be able to choose one or the other because the rate of printing generally falls off as the quality improves. There are three new strings used to describe these capabilities.

Print Quality	
snlq	Set near-letter quality print
snrmq	Set normal quality print
sdrfq	Set draft quality print

The capabilities are listed in decreasing levels of quality. If a printer doesn't have all three levels, one or two of the strings should be left blank as appropriate.

Section 2-9: Printing Rate and Buffer Size

Because there is no standard protocol that can be used to keep a program synchronized with a printer, and because modern printers can buffer data before printing it, a program generally cannot determine at any time what has been printed. Two new numeric capabilities can help a program estimate what has been printed.

Print Rate/Buffer Size	
cps	Nominal print rate in characters per second
bufsz	Buffer capacity in characters

cps is the nominal or average rate at which the printer prints characters; if this value is not given, the rate should be estimated at one-tenth the prevailing baud rate. bufsz is the maximum number of subsequent characters buffered before the guaranteed printing of an earlier character, assuming proper flow control has been used. If this value is not given it is assumed that the printer does not buffer characters, but prints them as they are received.

As an example, if a printer has a 1000-character buffer, then sending the letter "a" followed by 1000 additional characters is guaranteed to cause the letter "a" to print. If the same printer prints at the rate of 100 characters per second, then it should take 10 seconds to print all the characters in the buffer, less if the buffer is not full. By keeping track of the characters sent to a printer, and knowing the print rate and buffer size, a program can synchronize itself with the printer.

Note that most printer manufacturers advertise the maximum print rate, not the nominal print rate. A good way to get a value to put in for `cps` is to generate a few pages of text, count the number of printable characters, and then see how long it takes to print the text.

Applications that use these values should recognize the variability in the print rate. Straight text, in short lines, with no embedded control sequences will probably print at close to the advertised print rate and probably faster than the rate in `cps`. Graphics data with a lot of control sequences, or very long lines of text, will print at well below the advertised rate and below the rate in `cps`. If the application is using `cps` to decide how long it should take a printer to print a block of text, the application should pad the estimate. If the application is using `cps` to decide how much text has already been printed, it should shrink the estimate. The application will thus err in favor of the user, who wants, above all, to see all the output in its correct place.

FILES

`/usr/share/lib/terminfo/?/*`	compiled terminal description database
`/usr/share/lib/.COREterm/?/*`	subset of compiled terminal description database
`/usr/share/lib/tabset/*`	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs)

SEE ALSO

`curses`(3X), `ls`(1), `pg`(1), `printf`(3S), `stty`(1), `tic`(1M), `tput`(1), `tty`(1), `vi`(1)

NOTES

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in `terminfo` and to build up a description gradually, using partial descriptions with a screen oriented editor, such as `vi`, to check that they are correct. To easily test a new terminal description the environment variable `TERMINFO` can be set to the pathname of a directory containing the compiled description, and programs will look there rather than in `/usr/share/lib/terminfo`.

NAME

termio - general terminal interface

SYNOPSIS

```
#include <termio.h>

ioctl(int fildes, int request, struct termio *arg);
ioctl(int fildes, int request, int arg);

#include <termios.h>

ioctl(int fildes, int request, struct termios *arg);
```

DESCRIPTION

System V supports a general interface for asynchronous communications ports that is hardware-independent. The user interface to this functionality is via function calls (the preferred interface) described in termios(2) or ioctl commands described in this section. This section also discusses the common features of the terminal subsystem which are relevant with both user interfaces.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open terminal files; they are opened by the system and become a user's standard input, output, and error files. The very first terminal file opened by the session leader, which is not already associated with a session, becomes the controlling terminal for that session. The controlling terminal plays a special role in handling quit and interrupt signals, as discussed below. The controlling terminal is inherited by a child process during a fork(2). A process can break this association by changing its session using setsid(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 character input buffers of the system become completely full, which is rare (for example, if the number of characters in the line discipline buffer exceeds {MAX_CANON} and IMAXBEL [see below] is not set), or when the user has accumulated {MAX_INPUT} number of input characters that have not yet been read by some program. When the input limit is reached, all the characters saved in the buffer up to that point are thrown away without notice.

Session Management (Job Control)

A control terminal will distinguish one of the process groups in the session associated with it to be the foreground process group. All other process groups in the session are designated as background process groups. This foreground process group plays a special role in handling signal-generating input characters, as discussed below. By default, when a controlling terminal is allocated, the controlling process's process group is assigned as foreground process group.

Background process groups in the controlling process's session are subject to a job control line discipline when they attempt to access their controlling terminal. Process groups can be sent signals that will cause them to stop, unless they have made other arrangements. An exception is made for members of orphaned process groups. These are process groups which do not have a member with a parent in another process group that is in the same session and therefore shares the same controlling terminal. When a member's orphaned process group attempts to access its controlling terminal, errors will be returned. since there is no process to continue it if it should stop.

If a member of a background process group attempts to read its controlling terminal, its process group will be sent a SIGTTIN signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding SIGTTIN, or is a member of an orphaned process group, the read will fail with errno set to EIO, and no signal will be sent.

If a member of a background process group attempts to write its controlling terminal and the TOSTOP bit is set in the c_lflag field, its process group will be sent a SIGTTOU signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding SIGTTOU, the write will succeed. If the process is not ignoring or holding SIGTTOU and is a member of an orphaned process group, the write will fail with errno set to EIO, and no signal will be sent.

If TOSTOP is set and a member of a background process group attempts to ioctl its controlling terminal, and that ioctl will modify terminal parameters (for example, TCSETA, TCSETAW, TCSETAF, or TIOCSPGRP), its process group will be sent a SIGTTOU signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding SIGTTOU, the ioctl will succeed. If the process is not ignoring or holding SIGTTOU and is a member of an orphaned process group, the write will fail with errno set to EIO, and no signal will be sent.

Canonical Mode Input Processing

Normally, terminal input is processed in units of lines. A line is delimited by a newline (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 necessary, however, 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. The ERASE character (by default, the character #) erases the last character typed. The WERASE character (the character control-W) erases the last "word" typed in the current input line (but not any preceding spaces or tabs). A "word" is defined as a sequence of non-blank characters, with tabs counted as blanks. Neither ERASE nor WERASE will erase beyond the beginning of the line. The KILL character (by default, the character @) kills (deletes) the entire input line, and optionally outputs a newline character. All these characters operate on a key stroke basis, independent of any backspacing or tabbing that may have been done. The REPRINT character (the character control-R) prints a newline followed by all characters that have not been read. Reprinting also occurs automatically if characters that would normally be erased from the screen are fouled by program output. The characters are reprinted as if they were being echoed; consequencely, if ECHO is not set, they are not printed.

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.

Non-canonical Mode Input Processing

In non-canonical mode input processing, input characters are not assembled into lines, and erase and kill processing does not occur. The MIN and TIME values are used to determine how to process the characters received.

MIN represents the minimum number of characters that should be received when the read is satisfied (that is, when the characters are returned to the user). TIME is a timer of 0.10-second granularity that is used to timeout bursty and short-term data transmissions. The values for MIN and TIME should be set by the programmer in the termios or termio structure. The four possible values for MIN and TIME and their interactions are described below.

Case A: MIN > 0, TIME > 0
In this case, TIME serves as an intercharacter timer and is activated after the first character is received. Since it is an intercharacter timer, it is reset after a character is received. The interaction between MIN and TIME is as follows: as soon as one character is received, the intercharacter timer is started. If MIN characters are received before the intercharacter timer expires (note that the timer is reset upon receipt of each character), the read is satisfied. If the timer expires before MIN characters are received, the characters received to that point are returned to the user. Note that if TIME expires, at least one character will be returned because the timer would not have been enabled unless a character was received. In this case (MIN > 0, TIME > 0), the read sleeps until the MIN and TIME mechanisms are activated by the receipt of the first character. If the number of characters read is less than the number of characters available, the timer is not reactivated and the subsequent read is satisfied immediately.

Case B: MIN > 0, TIME = 0
In this case, since the value of TIME is zero, the timer plays no role and only MIN is significant. A pending read is not satisfied until MIN characters are received (the pending read sleeps until MIN characters are received). A program that uses this case to read record based terminal I/O may block indefinitely in the read operation.

Case C: MIN = 0, TIME > 0
In this case, since MIN = 0, TIME no longer represents an intercharacter timer: it now serves as a read timer that is activated as soon as a read is done. A read is satisfied as soon as a single character is received or the read timer expires. Note that, in this case, if the timer expires, no character is returned. If the timer does not expire, the only way the read can be satisfied is if a character is received. In this case, the read will not block indefinitely waiting for a character; if no character is received within TIME*.10 seconds after the read is initiated, the read returns with zero characters.

Case D: MIN = 0, TIME = 0
In this case, return is immediate. The minimum of either the number of characters requested or the number of characters currently available is returned without waiting for more characters to be input.

Comparison of the Different Cases of MIN, TIME Interaction

Some points to note about MIN and TIME:

1. In the following explanations, note that the interactions of MIN and TIME are not symmetric. For example, when MIN > 0 and TIME = 0, TIME has no effect. However, in the opposite case, where MIN = 0 and TIME > 0, both MIN and TIME play a role in that MIN is satisfied with the receipt of a single character.
2. Also note that in case A (MIN > 0, TIME > 0), TIME represents an intercharacter timer, whereas in case C (TIME = 0, TIME > 0), TIME represents a read timer.

These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, where MIN > 0, exist to handle burst mode activity (for example, file transfer programs), where a program would like to process at least MIN characters at a time. In case A, the intercharacter timer is activated by a user as a safety measure; in case B, the timer is turned off.

Cases C and D exist to handle single character, timed transfers. These cases are readily adaptable to screen-based applications that need to know if a character is present in the input queue before refreshing the screen. In case C, the read is timed, whereas in case D, it is not.

Another important note is that MIN is always just a minimum. It does not denote a record length. For example, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, then 20 characters will be returned to the user.

Writing Characters

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 as they are typed if echoing has been enabled. 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 is drained down to some threshold, the program is resumed.

Special Characters

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

INTR (Rubout or ASCII DEL) generates a SIGINT signal. SIGINT is sent to all frequent processes associated with the controlling 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(5)].

QUIT (CTRL-| or ASCII FS) generates a SIGQUIT 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 does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

WERASE (CTRL-W or ASCII ETX) erases the preceding ''word''. It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

KILL (@) deletes the entire line, as delimited by a NL, EOF, EOL, or EOL2 character.

REPRINT	(CTRL-R or ASCII DC2) reprints all characters, preceded by a newline, that have not been read.
EOF	(CTRL-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 newline, and the EOF is discarded. Thus, if no characters are waiting (that is, the EOF occurred at the beginning of a line) zero characters are passed back, which is the standard end-of-file indication. The EOF character is not echoed unless it is escaped or ECHOCTL is set. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.
NL	(ASCII LF) is the normal line delimiter. It cannot be changed or escaped.
EOL	(ASCII NULL) is an additional line delimiter, like NL. It is not normally used.
EOL2	is another additional line delimiter.
SWTCH	(CTRL-Z or ASCII EM) is used only when `shl` layers is invoked.
SUSP	(CTRL-Z or ASCII SUB) generates a `SIGTSTP` signal. `SIGTSTP` stops all processes in the foreground process group for that terminal.
DSUSP	(CTRL-Y or ASCII EM) It generates a `SIGTSTP` signal as SUSP does, but the signal is sent when a process in the foreground process group attempts to read the DSUSP character, rather than when it is typed.
STOP	(CTRL-S or ASCII DC3) can be used to suspend output temporarily. 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	(CTRL-Q or ASCII DC1) is used to resume output. Output has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read.
DISCARD	(CTRL-O or ASCII SI) causes subsequent output to be discarded. Output is discarded until another DISCARD character is typed, more input arrives, or the condition is cleared by a program.
LNEXT	(CTRL-V or ASCII SYN) causes the special meaning of the next character to be ignored. This works for all the special characters mentioned above. It allows characters to be input that would otherwise be interpreted by the system (for example, KILL, QUIT).

The character values for INTR, QUIT, ERASE, WERASE, KILL, REPRINT, EOF, EOL, EOL2, SWTCH, SUSP, DSUSP, STOP, START, DISCARD, and LNEXT may be changed to suit individual tastes. If the value of a special control character is _POSIX_VDISABLE (0), the function of that special control character is disabled. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done. Any of the special characters may be preceded by the LNEXT character, in which case no special function is done.

Modem Disconnect

When a modem disconnect is detected, a `SIGHUP` signal is sent to the terminal's controlling process. Unless other arrangements have been made, these signals cause the process to terminate. If `SIGHUP` is ignored or caught, any subsequent read returns with an end-of-file indication until the terminal is closed.

If the controlling process is not in the foreground process group of the terminal, a SIGTSTP is sent to the terminal's foreground process group. Unless other arrangements have been made, these signals cause the processes to stop.

Processes in background process groups that attempt to access the controlling terminal after modem disconnect while the terminal is still allocated to the session will receive appropriate SIGTTOU and SIGTTIN signals. Unless other arrangements have been made, this signal causes the processes to stop.

The controlling terminal will remain in this state until it is reinitialized with a successful open by the controlling process, or deallocated by the controlling process.

Terminal Parameters

The parameters that control the behavior of devices and modules providing the termios interface are specified by the termios structure defined by termios.h. Several ioctl(2) system calls that fetch or change these parameters use this structure that contains the following members:

```
tcflag_t   c_iflag;            /* input modes */
tcflag_t   c_oflag;            /* output modes */
tcflag_t   c_cflag;            /* control modes */
tcflag_t   c_lflag;            /* local modes */
cc_t       c_cc[NCCS];         /* control chars */
```

The special control characters are defined by the array c_cc. The symbolic name NCCS is the size of the control-character array and is also defined by termios.h. The relative positions, subscript names, and typical default values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	VEOL2	NUL
7	VSWTCH	NUL
8	VSTRT	DC1
9	VSTOP	DC3
10	VSUSP	SUB
11	VDSUSP	EM
12	VREPRINT	DC2
13	VDISCRD	SI
14	VWERASE	ETB
15	VLNEXT	SYN
16-19	reserved	

For the non-canonical mode the positions of VEOF and VEOL are shared by VMIN and VTIME:

4	VMIN	used to set the value of MIN
5	VTIME	used to set the value of TIME

Input Modes

The c_iflag field describes the basic terminal input control:

IGNBRK	Ignore break condition.
BRKINT	Signal interrupt on break.
IGNPAR	Ignore characters with parity errors.
PARMRK	Mark parity errors.
INPCK	Enable input parity check.
ISTRIP	Strip character.
INLCR	Map NL to CR on input.
IGNCR	Ignore CR.
ICRNL	Map CR to NL on input.
IUCLC	Map upper-case to lower-case on input.
IXON	Enable start/stop output control.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IMAXBEL	Echo BEL on input line too long.

If IGNBRK is set, a break condition (a character framing error with data all zeros) detected on input is ignored, that is, not put on the input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break condition shall flush the input and output queues and if the terminal is the controlling terminal of a foreground process group, the break condition generates a single SIGINT signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition is read as a single ASCII NULL character (´\0´), or if PARMRK is set, as ´\377´, ´\0´, ´\0´.

If IGNPAR is set, a byte with framing or parity errors (other than break) is ignored.

If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than break) is given to the application as the three-character sequence: ´\377´, ´\0´, *X*, where X is the data of the byte received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of ´\377´ is given to the application as ´\377´, ´\377´. If neither IGNPAR nor PARMRK is set, a framing or parity error (other than break) is given to the application as a single ASCII NULL character (´\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. Note that whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled. If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected will recognize the parity bit, but the terminal special file will not check whether this is set correctly or not.

If ISTRIP is set, valid input characters are first stripped to seven bits, otherwise all eight 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 suspends output and a received START character restarts output. The STOP and START characters will not be read, but will merely perform flow control functions. If IXANY is set, any input character restarts output that has been suspended.

If IXOFF is set, the system transmits a STOP character when the input queue is nearly full, and a START character when enough input has been read so that the input queue is nearly empty again.

If IMAXBEL is set, the ASCII BEL character is echoed if the input stream overflows. Further input is not stored, but any input already present in the input stream is not disturbed. If IMAXBEL is not set, no BEL character is echoed, and all input present in the input queue is discarded if the input stream overflows.

The initial input control value is BRKINT, ICRNL, IXON, ISTRIP.

Output Modes

The c_oflag field specifies the system treatment of output:

OPOST	Post-process output.
OLCUC	Map lower case to upper on output.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NULL.
NLDLY	Select newline delays:
NL0	
NL1	
CRDLY	Select carriage-return delays:
CR0	
CR1	
CR2	
CR3	
TABDLY	Select horizontal tab delays:
TAB0	or tab expansion:
TAB1	
TAB2	
TAB3	Expand tabs to spaces.
XTABS	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	
BS1	
VTDLY	Select vertical tab delays:
VT0	
VT1	
FFDLY	Select form feed delays:
FF0	
FF1	

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 ONRET is set, the NL character is assumed to do the carriage-return function; the column pointer is set to 0 and the delays specified for CR are used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains 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 are transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay. If OFDEL is set, the fill character is DEL; otherwise it is NULL.

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

Newline delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the newline delays. If OFILL is set, two fill characters are 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 transmits 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 are transmitted for any delay.

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

The actual delays depend on line speed and system load.

The initial output control value is OPOST, ONLCR, TAB3.

Control Modes

The c_cflag field describes the hardware control of the terminal:

CBAUD	Baud rate:
B0	Hang up
B50	50 baud
B75	75 baud
B110	110 baud
B134	134 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud
B2400	2400 baud

B4800	4800 baud
B9600	9600 baud
B19200	19200 baud
EXTA	External A
B38400	38400 baud
EXTB	External B

CSIZE	Character size:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits

CSTOPB	Send two stop bits, else one
CREAD	Enable receiver
PARENB	Parity enable
PARODD	Odd parity, else even
HUPCL	Hang up on last close
CLOCAL	Local line, else dial-up
CIBAUD	Input baud rate, if different from output rate
PAREXT	Extended parity for mark and space parity

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 is not asserted. Normally, this disconnects the line. If the CIBAUD bits are not zero, they specify the input baud rate, with the CBAUD bits specifying the output baud rate; otherwise, the output and input baud rates are both specified by the CBAUD bits. The values for the CIBAUD bits are the same as the values for the CBAUD bits, shifted left IBSHIFT bits. 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 is used. For example, at 110 baud, two stops bits are required.

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

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

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

Local Modes

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

`ISIG`	Enable signals.
`ICANON`	Canonical input (erase and kill processing).
`XCASE`	Canonical upper/lower presentation.
`ECHO`	Enable echo.
`ECHOE`	Echo erase character as BS-SP-BS.
`ECHOK`	Echo NL after kill character.
`ECHONL`	Echo NL.
`NOFLSH`	Disable flush after interrupt or quit.
`TOSTOP`	Send `SIGTTOU` for background output.
`ECHOCTL`	Echo control characters as ^*char*, delete as ^?.
`ECHOPRT`	Echo erase character as character erased.
`ECHOKE`	BS-SP-BS erase entire line on line kill.
`FLUSHO`	Output is being flushed.
`PENDIN`	Retype pending input at next read or input character.
`IEXTEN`	Enable extended (implementation-defined) functions.

If `ISIG` is set, each input character is checked against the special control characters INTR, QUIT, SWTCH, SUSP, STATUS, and DSUSP. 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.

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, EOL, and EOL2. If `ICANON` is not set, read requests are satisfied directly from the input queue. A read is not satisfied until at least `MIN` characters have been received or the timeout value `TIME` has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. 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.

1. If `ECHO` and `ECHOE` are set, and `ECHOPRT` is not set, the ERASE and WERASE characters are echoed as one or more ASCII BS SP BS, which clears the last character(s) from a CRT screen.

2. If ECHO and ECHOPRT are set, the first ERASE and WERASE character in a sequence echoes as a backslash (\), followed by the characters being erased. Subsequent ERASE and WERASE characters echo the characters being erased, in reverse order. The next non-erase character causes a slash (/) to be typed before it is echoed. ECHOPRT should be used for hard copy terminals.
3. If ECHOKE is set, the kill character is echoed by erasing each character on the line from the screen (using the mechanism selected by ECHOE and ECHOPRT).
4. If ECHOK is set, and ECHOKE is not set, the NL character is echoed after the kill character to emphasize that the line is deleted. Note that an escape character (\) or an LNEXT character preceding the erase or kill character removes any special function.
5. If ECHONL is set, the NL character is echoed even if ECHO is not set. This is useful for terminals set to local echo (so called half-duplex).

If ECHOCTL is set, all control characters (characters with codes between 0 and 37 octal) other than ASCII TAB, ASCII NL, the START character, and the STOP character, ASCII CR, and ASCII BS are echoed as ^X, where X is the character given by adding 100 octal to the code of the control character (so that the character with octal code 1 is echoed as ^A), and the ASCII DEL character, with code 177 octal, is echoed as ^?.

If NOFLSH is set, the normal flush of the input and output queues associated with the INTR, QUIT, and SUSP characters is not done. This bit should be set when restarting system calls that read from or write to a terminal [see sigaction(2)].

If TOSTOP is set, the signal SIGTTOU is sent to a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal normally stops the process. Otherwise, the output generated by that process is output to the current output stream. Processes that are blocking or ignoring SIGTTOU signals are excepted and allowed to produce output, if any.

If FLUSHO is set, data written to the terminal is discarded. This bit is set when the FLUSH character is typed. A program can cancel the effect of typing the FLUSH character by clearing FLUSHO.

If PENDIN is set, any input that has not yet been read is reprinted when the next character arrives as input.

If IEXTEN is set, the following implementation-defined functions are enabled: special characters (WERASE, REPRINT, DISCARD, and LNEXT) and local flags (TOSTOP, ECHOCTL, ECHOPRT, ECHOKE, FLUSHO, and PENDIN).

The initial line-discipline control value is ISIG, ICANON, ECHO, ECHOK.

Terminal Size

The number of lines and columns on the terminal's display is specified in the winsize structure defined by sys/termios.h and includes the following members:

```
unsigned  short   ws_row;    /* rows, in characters */
unsigned  short   ws_col;    /* columns, in characters */
unsigned  short   ws_xpixel;/* horizontal size, in pixels */
unsigned  short   ws_ypixel;/* vertical size, in pixels */
```

termio Structure

The System V `termio` structure is used by some `ioctls`; it is defined by `sys/termio.h` and includes the following members:

```
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 symbolic name `NCC` is the size of the control-character array and is also defined by `termio.h`. The relative positions, subscript names, and typical default values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	VEOL2	NUL
7	reserved	

For the non-canonical mode the positions VEOF and VEOL are shared by VMIN and VTIME:

4	VMIN	used to set the value of MIN
5	VTIME	used to set the value of TIME

The calls that use the `termio` structure only affect the flags and control characters that can be stored in the `termio` structure; all other flags and control characters are unaffected.

Modem Lines

On special files representing serial ports, the modem control lines supported by the hardware can be read, and the modem status lines supported by the hardware can be changed. The following modem control and status lines may be supported by a device; they are defined by `sys/termios.h`:

TIOCM_LE	line enable
TIOCM_DTR	data terminal ready
TIOCM_RTS	request to send
TIOCM_ST	secondary transmit
TIOCM_SR	secondary receive
TIOCM_CTS	clear to send
TIOCM_CAR	carrier detect
TIOCM_RNG	ring
TIOCM_DSR	data set ready

TIOCM_CD is a synonym for TIOCM_CAR, and TIOCM_RI is a synonym for TIOCM_RNG. Not all of these are necessarily supported by any particular device; check the manual page for the device in question.

ioctls

The ioctls supported by devices and STREAMS modules providing the termios interface are listed below. Some calls may not be supported by all devices or modules. The functionality provided by these calls is also available through the preferred function call interface specified on termios(2).

TCGETS The argument is a pointer to a termios structure. The current terminal parameters are fetched and stored into that structure.

TCSETS The argument is a pointer to a termios structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.

TCSETSW The argument is a pointer to a termios structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.

TCSETSF The argument is a pointer to a termios structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

TCGETA The argument is a pointer to a termio structure. The current terminal parameters are fetched, and those parameters that can be stored in a termio structure are stored into that structure.

TCSETA The argument is a pointer to a termio structure. Those terminal parameters that can be stored in a termio structure are set from the values stored in that structure. The change is immediate.

TCSETAW The argument is a pointer to a termio structure. Those terminal parameters that can be stored in a termio structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.

TCSETAF The argument is a pointer to a termio structure. Those terminal parameters that can be stored in a termio structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

TCSBRK The argument is an int value. Wait for the output to drain. If the argument is 0, then send a break (zero valued bits for 0.25 seconds).

TCXONC — Start/stop control. The argument is an int value. If the argument is 0, suspend output; if 1, restart suspended output; if 2, suspend input; if 3, restart suspended input.

TCFLSH — The argument is an int value. If the argument is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues. On some controllers, if the argument is 0, input flow control characters will be flushed, causing the unflushed output queue to overflow a busy output device.

TIOCGPGRP — The argument is a pointer to a pid_t. Set the value of that pid_t to the process group ID of the foreground process group associated with the terminal. See termios(2) for a description or TCGETPGRP.

TIOCSPGRP — The argument is a pointer to a pid_t. Associate the process group whose process group ID is specified by the value of that pid_t with the terminal. The new process group value must be in the range of valid process group ID values. Otherwise, the error EPERM is returned. See termios(2) for a description of TCSETPGRP.

TIOCGSID — The argument is a pointer to a pid_t. The session ID of the terminal is fetched and stored in the pid_t.

TIOCGWINSZ — The argument is a pointer to a winsize structure. The terminal driver's notion of the terminal size is stored into that structure.

TIOCSWINSZ — The argument is a pointer to a winsize structure. The terminal driver's notion of the terminal size is set from the values specified in that structure. If the new sizes are different from the old sizes, a SIGWINCH signal is set to the process group of the terminal.

TIOCMBIS — The argument is a pointer to an int whose value is a mask containing modem control lines to be turned on. The control lines whose bits are set in the argument are turned on; no other control lines are affected.

TIOCMBIC — The argument is a pointer to an int whose value is a mask containing modem control lines to be turned off. The control lines whose bits are set in the argument are turned off; no other control lines are affected.

TIOCMGET — The argument is a pointer to an int. The current state of the modem status lines is fetched and stored in the int pointed to by the argument.

TIOCMSET — The argument is a pointer to an int containing a new set of modem control lines. The modem control lines are turned on or off, depending on whether the bit for that mode is set or clear.

FILES

files in or under /dev

SEE ALSO

fork(2), ioctl(2), setsid(2), signal(2), termios(2), streamio(7)

NAME

termiox - extended general terminal interface

DESCRIPTION

The extended general terminal interface supplements the termio(7) general terminal interface by adding support for asynchronous hardware flow control, isochronous flow control and clock modes, and local implementations of additional asynchronous features. Please refer to the device specific man pages of the device being utilized to determine whether hardware flow control is supported. Some systems may not support all of these capabilities because of either hardware or software limitations. Other systems may not permit certain functions to be disabled. In these cases the appropriate bits will be ignored. See termiox.h for your system to find out which capabilities are supported.

Hardware Flow Control Modes

Hardware flow control supplements the termio(7) IXON, IXOFF, and IXANY character flow control. Character flow control occurs when one device controls the data transfer of another device by the insertion of control characters in the data stream between devices. Hardware flow control occurs when one device controls the data transfer of another device using electrical control signals on wires (circuits) of the asynchronous interface. Isochronous hardware flow control occurs when one device controls the data transfer of another device by asserting or removing the transmit clock signals of that device. Character flow control and hardware flow control may be simultaneously set.

In asynchronous, full duplex applications, the use of the Electronic Industries Association's EIA-232-D Request To Send (RTS) and Clear To Send (CTS) circuits is the preferred method of hardware flow control. An interface to other hardware flow control methods is included to provide a standard interface to these existing methods.

The EIA-232-D standard specified only uni-directional hardware flow control - the Data Circuit-terminating Equipment or Data Communications Equipment (DCE) indicates to the Data Terminal Equipment (DTE) to stop transmitting data. The termiox(7) interface allows both uni-directional and bi-directional hardware flow control; when bi-directional flow control is enabled, either the DCE or DTE can indicate to each other to stop transmitting data across the interface. Note: It is assumed that the asynchronous port is configured as a DTE. If the connected device is also a DTE and not a DCE, then DTE to DTE (for example, terminal or printer connected to computer) hardware flow control is possible by using a null modem to interconnect the appropriate data and control circuits.

Clock Modes

Isochronous communication is a variation of asynchronous communication whereby two communicating devices may provide transmit and/or receive clock to each other. Incoming clock signals can be taken from the baud rate generator on the local isochronous port controller, from CCITT V.24 circuit 114, Transmitter Signal Element Timing - DCE source (EIA-232-D pin 15), or from CCITT V.24 circuit 115, Receiver Signal Element Timing - DCE source (EIA-232-D pin 17). Outgoing clock signals can be sent on CCITT V.24 circuit 113, Transmitter Signal Element Timing - DTE source (EIA-232-D pin 24), on CCITT V.24 circuit 128, Receiver Signal Element Timing - DTE source (no EIA-232-D pin), or not sent at all.

In terms of clock modes, traditional asynchronous communication is implemented simply by using the local baud rate generator as the incoming transmit and receive clock source and not outputting any clock signals.

Terminal Parameters

The parameters that control the behavior of devices providing the termiox interface are specified by the termiox structure, defined in the sys/termiox.h header file. Several ioctl(2) system calls that fetch or change these parameters use this structure:

```
#define NFF       5
struct   termiox {
   unsigned short   x_hflag;      /* hardware flow control
                                     modes */
   unsigned short   x_cflag;      /* clock modes */
   unsigned short   x_rflag[NFF];/* reserved modes */
   unsigned short   x_sflag;      /* spare local modes */
};
```

The x_hflag field describes hardware flow control modes:

RTSXOFF	0000001	Enable RTS hardware flow control on input.
CTSXON	0000002	Enable CTS hardware flow control on output.
DTRXOFF	0000004	Enable DTR hardware flow control on input.
CDXON	0000010	Enable CD hardware flow control on output.
ISXOFF	0000020	Enable isochronous hardware flow control on input.

The EIA-232-D DTR and CD circuits are used to establish a connection between two systems. The RTS circuit is also used to establish a connection with a modem. Thus, both DTR and RTS are activated when an asynchronous port is opened. If DTR is used for hardware flow control, then RTS must be used for connectivity. If CD is used for hardware flow control, then CTS must be used for connectivity. Thus, RTS and DTR (or CTS and CD) cannot both be used for hardware flow control at the same time. Other mutual exclusions may apply, such as the simultaneous setting of the termio(7) HUPCL and the termiox(7) DTRXOFF bits, which use the DTE ready line for different functions.

Variations of different hardware flow control methods may be selected by setting the the appropriate bits. For example, bi-directional RTS/CTS flow control is selected by setting both the RTSXOFF and CTSXON bits and bi-directional DTR/CTS flow control is selected by setting both the DTRXOFF and CTSXON. Modem control or uni-directional CTS hardware flow control is selected by setting only the CTSXON bit.

As previously mentioned, it is assumed that the local asynchronous port (for example, computer) is configured as a DTE. If the connected device (for example, printer) is also a DTE, it is assumed that the device is connected to the computer's asynchronous port via a null modem that swaps control circuits (typically RTS and CTS). The connected DTE drives RTS and the null modem swaps RTS and CTS so that the remote RTS is received as CTS by the local DTE. In the case that CTSXON is set for hardware flow control, printer's lowering of its RTS would cause CTS seen by the computer to be lowered. Output to the printer is suspended until the printer's raising of its RTS, which would cause CTS seen by the computer to be raised.

If RTSXOFF is set, the Request To Send (RTS) circuit (line) will be raised, and if the asynchronous port needs to have its input stopped, it will lower the Request To Send (RTS) line. If the RTS line is lowered, it is assumed that the connected device will stop its output until RTS is raised.

If CTSXON is set, output will occur only if the Clear To Send (CTS) circuit (line) is raised by the connected device. If the CTS line is lowered by the connected device, output is suspended until CTS is raised.

If DTRXOFF is set, the DTE Ready (DTR) circuit (line) will be raised, and if the asynchronous port needs to have its input stopped, it will lower the DTE Ready (DTR) line. If the DTR line is lowered, it is assumed that the connected device will stop its output until DTR is raised.

If CDXON is set, output will occur only if the Received Line Signal Detector (CD) circuit (line) is raised by the connected device. If the CD line is lowered by the connected device, output is suspended until CD is raised.

If ISXOFF is set, and if the isochronous port needs to have its input stopped, it will stop the outgoing clock signal. It is assumed that the connected device is using this clock signal to create its output. Transit and receive clock sources are programmed using the x_cflag fields. If the port is not programmed for external clock generation, ISXOFF is ignored. Output isochronous flow control is supported by appropriate clock source programming using the x_cflag field and enabled at the remote connected device.

The x_cflag field specifies the system treatment of clock modes.

XMTCLK	0000007	Transmit clock source:
XCIBRG	0000000	Get transmit clock from internal baud rate generator.
XCTSET	0000001	Get transmit clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
XCRSET	0000002	Get transmit clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.
RCVCLK	0000070	Receive clock source:
RCIBRG	0000000	Get receive clock from internal baud rate generator.
RCTSET	0000010	Get receive clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
RCRSET	0000020	Get receive clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.
TSETCLK	0000700	Transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24, clock source:
TSETCOFF	0000000	TSET clock not provided.

TSETCRBRG	0000100	Output receive baud rate generator on circuit 113.
TSETCTBRG	0000200	Output transmit baud rate generator on circuit 113.
TSETCTSET	0000300	Output transmitter signal element timing (DCE source) on circuit 113.
TSETCRSET	0000400	Output receiver signal element timing (DCE source) on circuit 113.
RSETCLK	0007000	Receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin, clock source:
RSETCOFF	0000000	RSET clock not provided.
RSETCRBRG	0001000	Output receive baud rate generator on circuit 128.
RSETCTBRG	0002000	Output transmit baud rate generator on circuit 128.
RSETCTSET	0003000	Output transmitter signal element timing (DCE source) on circuit 128.
RSETCRSET	0004000	Output receiver signal element timing (DCE) on circuit 128.

If the XMTCLK field has a value of XCIBRG the transmit clock is taken from the hardware internal baud rate generator, as in normal asynchronous transmission. If XMTCLK = XCTSET the transmit clock is taken from the Transmitter Signal Element Timing (DCE source) circuit. If XMTCLK = XCRSET the transmit clock is taken from the Receiver Signal Element Timing (DCE source) circuit.

If the RCVCLK field has a value of RCIBRG the receive clock is taken from the hardware Internal Baud Rate Generator, as in normal asynchronous transmission. If RCVCLK = RCTSET the receive clock is taken from the Transmitter Signal Element Timing (DCE source) circuit. If RCVCLK = RCRSET the receive clock is taken from the Receiver Signal Element Timing (DCE source) circuit.

If the TSETCLK field has a value of TSETCOFF the Transmitter Signal Element Timing (DTE source) circuit is not driven. If TSETCLK = TSETCRBRG the Transmitter Signal Element Timing (DTE source) circuit is driven by the Receive Baud Rate Generator. If TSETCLK = TSETCTBRG the Transmitter Signal Element Timing (DTE source) circuit is driven by the Transmit Baud Rate Generator. If TSETCLK = TSETCTSET the Transmitter Signal Element Timing (DTE source) circuit is driven by the Transmitter Signal Element Timing (DCE source). If TSETCLK = TSETCRBRG the Transmitter Signal Element Timing (DTE source) circuit is driven by the Receiver Signal Element Timing (DCE source).

If the RSETCLK field has a value of RSETCOFF the Receiver Signal Element Timing (DTE source) circuit is not driven. If RSETCLK = RSETCRBRG the Receiver Signal Element Timing (DTE source) circuit is driven by the Receive Baud Rate Generator. If RSETCLK = RSETCTBRG the Receiver Signal Element Timing (DTE source) circuit is driven by the Transmit Baud Rate Generator. If RSETCLK = RSETCTSET the Receiver Signal Element Timing (DTE source) circuit is driven by the Transmitter Signal Element Timing (DCE source). If RSETCLK = RSETCRBRG the Receiver Signal Element Timing (DTE source) circuit is driven by the Receiver Signal Element Timing (DCE source).

The x_rflag is reserved for future interface definitions and should not be used by any implementations. The x_sflag may be used by local implementations wishing to customize their terminal interface using the termiox(7) ioctl system calls.

IOCTLS

The ioctl(2) system calls have the form:

ioctl (*fildes, command, arg*)
struct termiox **arg*;

The commands using this form are:

TCGETX The argument is a pointer to a termiox structure. The current terminal parameters are fetched and stored into that structure.

TCSETX The argument is a pointer to a termiox structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.

TCSETXW The argument is a pointer to a termiox structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that will affect output.

TCSETXF The argument is a pointer to a termiox structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

FILES

/dev/*

SEE ALSO

stty(1), ioctl(2), termio(7)

NAME

ticlts, ticots, ticotsord - loopback transport providers

SYNOPSIS

```
#include <sys/ticlts.h>
#include <sys/ticots.h>
#include <sys/ticotsord.h>
```

DESCRIPTION

The devices known as ticlts, ticots, and ticotsord are "loopback transport providers," that is, stand-alone networks at the transport level. Loopback transport providers are transport providers in every sense except one: only one host (the local machine) is "connected to" a loopback network. Loopback transports present a TPI (STREAMS-level) interface to application processes and are intended to be accessed via the TLI (application-level) interface. They are implemented as clone devices and support address spaces consisting of "flex-addresses," that is, arbitrary sequences of octets, of length > 0, represented by a netbuf structure.

ticlts is a datagram-mode transport provider. It offers (connectionless) service of type T_CLTS. Its default address size is TCL_DEFAULTADDRSZ. ticlts prints the following error messages (see t_rcvuderr(3N)):

TCL_BADADDR	bad address specification
TCL_BADOPT	bad option specification
TCL_NOPEER	bound
TCL_PEERBADSTATE	peer in wrong state

ticots is a virtual circuit-mode transport provider. It offers (connection-oriented) service of type T_COTS. Its default address size is TCO_DEFAULTADDRSZ. ticots prints the following disconnect messages (see t_rcvdis(3N)):

TCO_NOPEER	no listener on destination address
TCO_PEERNOROOMONQ	peer has no room on connect queue
TCO_PEERBADSTATE	peer in wrong state
TCO_PEERINITIATED	peer-initiated disconnect
TCO_PROVIDERINITIATED	provider-initiated disconnect

ticotsord is a virtual circuit-mode transport provider, offering service of type T_COTS_ORD (connection-oriented service with orderly release). Its default address size is TCOO_DEFAULTADDRSZ. ticotsord prints the following disconnect messages (see t_rcvdis(3N)):

TCOO_NOPEER	no listener on destination address
TCOO_PEERNOROOMONQ	peer has no room on connect queue
TCOO_PEERBADSTATE	peer in wrong state
TCOO_PEERINITIATED	peer-initiated disconnect
TCOO_PROVIDERINITIATED	provider-initiated disconnect

USAGE

Loopback transports support a local IPC mechanism through the TLI interface. Applications implemented in a transport provider-independent manner on a client-server model using this IPC are transparently transportable to networked environments.

Transport provider-independent applications must not include the header files listed in the synopsis section above. In particular, the options are (like all transport provider options) provider dependent.

ticlts and ticots support the same service types (T_CLTS and T_COTS) supported by the OSI transport-level model. The use of ticlts and ticots is encouraged.

ticotsord supports the same service type (T_COTSORD) supported by the TCP/IP model. The use of ticotsord is discouraged except for reasons of compatibility.

FILES

/dev/ticlts
/dev/ticots
/dev/ticotsord

NAME

timednet.conf - time daemon network configuration file.

SYNOPSIS

/etc/timednet.conf

DESCRIPTION

/etc/timednet.conf describes the configuration of a site's time daemon network. It is examined by the startup script /etc/init.d/timed to determine if and in what manner in.timed should be started.

EXAMPLE

The following example describes the format of /etc/timednet.conf:

```
#
#F1       F2    F3       F4          F5
#
jibboo    yes   master
raygun    yes   master              testnet
charm     YES   slave               testnet
neptune   no    slave    netA:netB
```

Field 1:

The hostname of the host that this entry pertains to.

Field 2:

Determines whether are not a time daemon should be started on the host whose name appears in field 1. Legal values are yes, YES, and no. If the field contains YES then in.timed will be started in trace mode.

Field 3:

Determines whether in.timed will be started in master or slave mode. Legal values are master or slave.

Field 4:

A list of networks (see /etc/networks) that in.timed will exclusively monitor (see the -n option of in.timed). If more than one network appears in the list, each network must be separated by a colon (:) with no intervening white space between the colon and the network names. If there are no networks to monitor, then the field must contain a dash (-).

Field 5:

A list of networks (see /etc/networks) that in.timed will ignore (see the -i option of in.timed). If more than one network appears in the list, each network must be separated by a colon (:) with no intervening white space between the colon and the network names. If there are no networks to ignore, then the field must contain a dash (-).

Lines beginning with a pound sign (#) will be treated as comments and ignored.

NOTES

Network interfaces specified in `/etc/if.ignore` will also be ignored by `in.timed`. Whether both files, either file, or neither file exist on a system, it is the system administrator's responsibility to ensure a appropriate configuration.

SEE ALSO

`date`(1), `in.timed`(1M), `timedc`(1M), `if.ignore`(4).

NAME

timezone - set default system time zone

SYNOPSIS

/etc/TIMEZONE

DESCRIPTION

This file sets and exports the time zone environmental variable TZ.

This file is "dotted" into other files that must know the time zone.

EXAMPLES

/etc/TIMEZONE for the east coast:

```
#     Time Zone
TZ=EST5EDT
export TZ
```

SEE ALSO

rc2(1M), ctime(3C), profile(4), environ(5).

NAME

timod - Transport Interface cooperating STREAMS module

DESCRIPTION

timod is a STREAMS module for use with the Transport Interface (TI) functions of the Network Services library. The timod module converts a set of ioctl(2) calls into STREAMS messages that may be consumed by a transport protocol provider which supports the Transport Interface. This allows a user to initiate certain TI functions as atomic operations.

The timod module must be pushed onto only a stream terminated by a transport protocol provider which supports the TI.

All STREAMS messages, with the exception of the message types generated from the ioctl commands described below, will be transparently passed to the neighboring STREAMS module or driver. The messages generated from the following ioctl commands are recognized and processed by the timod module. The format of the ioctl call is:

```
#include <sys/stropts.h>
          -
          -
struct strioctl strioctl;
          -
          -
strioctl.ic_cmd = cmd;
strioctl.ic_timeout = INFTIM;
strioctl.ic_len = size;
strioctl.ic_dp = (char *)buf
ioctl(fildes, I_STR, &strioctl);
```

Where, on issuance, *size* is the size of the appropriate TI message to be sent to the transport provider and on return *size* is the size of the appropriate TI message from the transport provider in response to the issued TI message. *buf* is a pointer to a buffer large enough to hold the contents of the appropriate TI messages. The TI message types are defined in sys/tihdr.h. The possible values for the *cmd* field are:

TI_BIND — Bind an address to the underlying transport protocol provider. The message issued to the TI_BIND ioctl is equivalent to the TI message type T_BIND_REQ and the message returned by the successful completion of the ioctl is equivalent to the TI message type T_BIND_ACK.

TI_UNBIND — Unbind an address from the underlying transport protocol provider. The message issued to the TI_UNBIND ioctl is equivalent to the TI message type T_UNBIND_REQ and the message returned by the successful completion of the ioctl is equivalent to the TI message type T_OK_ACK.

TI_GETINFO — Get the TI protocol specific information from the transport protocol provider. The message issued to the TI_GETINFO ioctl is equivalent to the TI message type T_INFO_REQ and the message returned by the successful completion of the ioctl is equivalent to the TI message type T_INFO_ACK.

TI_OPTMGMT Get, set or negotiate protocol specific options with the transport protocol provider. The message issued to the TI_OPTMGMT ioctl is equivalent to the TI message type T_OPTMGMT_REQ and the message returned by the successful completion of the ioctl is equivalent to the TI message type T_OPTMGMT_ACK.

FILES

sys/timod.h
sys/tiuser.h
sys/tihdr.h
sys/errno.h

SEE ALSO

tirdwr(7).

DIAGNOSTICS

If the ioctl system call returns with a value greater than 0, the lower 8 bits of the return value will be one of the TI error codes as defined in sys/tiuser.h. If the TI error is of type TSYSERR, then the next 8 bits of the return value will contain an error as defined in sys/errno.h [see intro(2)].

NAME

tirdwr - Transport Interface read/write interface STREAMS module

DESCRIPTION

tirdwr is a STREAMS module that provides an alternate interface to a transport provider which supports the Transport Interface (TI) functions of the Network Services library (see Section 3N). This alternate interface allows a user to communicate with the transport protocol provider using the read(2) and write(2) system calls. The putmsg(2) and getmsg(2) system calls may also be used. However, putmsg and getmsg can only transfer data messages between user and stream.

The tirdwr module must only be pushed [see I_PUSH in streamio(7)] onto a stream terminated by a transport protocol provider which supports the TI. After the tirdwr module has been pushed onto a stream, none of the Transport Interface functions can be used. Subsequent calls to TI functions will cause an error on the stream. Once the error is detected, subsequent system calls on the stream will return an error with errno set to EPROTO.

The following are the actions taken by the tirdwr module when pushed on the stream, popped [see I_POP in streamio(7)] off the stream, or when data passes through it.

push When the module is pushed onto a stream, it will check any existing data destined for the user to ensure that only regular data messages are present. It will ignore any messages on the stream that relate to process management, such as messages that generate signals to the user processes associated with the stream. If any other messages are present, the I_PUSH will return an error with errno set to EPROTO.

write The module will take the following actions on data that originated from a write system call:

All messages with the exception of messages that contain control portions (see the putmsg and getmsg system calls) will be transparently passed onto the module's downstream neighbor.

Any zero length data messages will be freed by the module and they will not be passed onto the module's downstream neighbor.

Any messages with control portions will generate an error, and any further system calls associated with the stream will fail with errno set to EPROTO.

read The module will take the following actions on data that originated from the transport protocol provider:

All messages with the exception of those that contain control portions (see the putmsg and getmsg system calls) will be transparently passed onto the module's upstream neighbor.

The action taken on messages with control portions will be as follows:

Messages that represent expedited data will generate an error. All further system calls associated with the stream will fail with `errno` set to `EPROTO`.

Any data messages with control portions will have the control portions removed from the message prior to passing the message on to the upstream neighbor.

Messages that represent an orderly release indication from the transport provider will generate a zero length data message, indicating the end of file, which will be sent to the reader of the stream. The orderly release message itself will be freed by the module.

Messages that represent an abortive disconnect indication from the transport provider will cause all further `write` and `putmsg` system calls to fail with `errno` set to `ENXIO`. All further `read` and `getmsg` system calls will return zero length data (indicating end of file) once all previous data has been read.

With the exception of the above rules, all other messages with control portions will generate an error and all further system calls associated with the stream will fail with `errno` set to `EPROTO`.

Any zero length data messages will be freed by the module and they will not be passed onto the module's upstream neighbor.

pop When the module is popped off the stream or the stream is closed, the module will take the following action:

If an orderly release indication has been previously received, then an orderly release request will be sent to the remote side of the transport connection.

SEE ALSO

`getmsg`(2), `intro`(2), `putmsg`(2), `read`(2), `write`(2), `intro`(3), `streamio`(7), `timod`(7).

NAME

ts_dptbl - time-sharing dispatcher parameter table

DESCRIPTION

The process scheduler (or dispatcher) is the portion of the kernel that controls allocation of the CPU to processes. The scheduler supports the notion of scheduling classes where each class defines a scheduling policy, used to schedule processes within that class. Associated with each scheduling class is a set of priority queues on which ready to run processes are linked. These priority queues are mapped by the system configuration into a set of global scheduling priorities which are available to processes within the class. (The dispatcher always selects for execution the process with the highest global scheduling priority in the system.) The priority queues associated with a given class are viewed by that class as a contiguous set of priority levels numbered from 0 (lowest priority) to *n* (highest priority—a configuration-dependent value). The set of global scheduling priorities that the queues for a given class are mapped into might not start at zero and might not be contiguous (depending on the configuration).

Processes in the time-sharing class which are running in user mode (or in kernel mode before going to sleep) are scheduled according to the parameters in a time-sharing dispatcher parameter table (ts_dptbl). (Time-sharing processes running in kernel mode after sleeping are run within a special range of priorities reserved for such processes and are not affected by the parameters in the ts_dptbl until they return to user mode.) The ts_dptbl consists of an array of parameter structures (struct ts_dpent), one for each of the *n* priority levels used by time-sharing processes in user mode. The properties of a given priority level *i* are specified by the *i*th parameter structure in this array (ts_dptbl*i*).

A parameter structure consists of the following members. These are also described in the /usr/include/sys/ts.h header file.

ts_globpri
: The global scheduling priority associated with this priority level. The mapping between time-sharing priority levels and global scheduling priorities is determined at boot time by the system configuration. ts_globpri is the only member of the ts_dptbl which cannot be changed with dispadmin(1M).

ts_quantum
: The length of the time quantum allocated to processes at this level in ticks (HZ).

ts_tqexp
: Priority level of the new queue on which to place a process running at the current level if it exceeds its time quantum. Normally this field links to a lower priority time-sharing level that has a larger quantum.

ts_slpret
: Priority level of the new queue on which to place a process, that was previously in user mode at this level, when it returns to user mode after sleeping. Normally this field links to a higher priority level that has a smaller quantum.

ts_maxwait
: A per process counter, ts_dispwait is initialized to zero each time a time-sharing process is placed back on the dispatcher queue after its time quantum has expired or when it is awakened (ts_dispwait is not reset to zero when a process is preempted by a higher priority process). This counter is incremented once

per second for each process on the dispatcher queue. If a process's ts_dispwait value exceeds the ts_maxwait value for its level, the process's priority is changed to that indicated by ts_lwait. The purpose of this field is to prevent starvation.

ts_lwait Move a process to this new priority level if ts_dispwait is greater than ts_maxwait.

An administrator can affect the behavior of the time-sharing portion of the scheduler by reconfiguring the ts_dptbl. There are two methods available for doing this.

MASTER FILE

The ts_dptbl can be reconfigured at boot time by specifying the desired values in the ts master file and reconfiguring the system using the auto-configuration boot procedure; see mkboot(1M) and master(4). This is the only method that can be used to change the number of time-sharing priority levels or the set of global scheduling priorities used by the time-sharing class.

DISPADMIN CONFIGURATION FILE

With the exception of ts_globpri all of the members of the ts_dptbl can be examined and modified on a running system using the dispadmin(1M) command. Invoking dispadmin for the time-sharing class allows the administrator to retrieve the current ts_dptbl configuration from the kernel's in-core table, or overwrite the in-core table with values from a configuration file. The configuration file used for input to dispadmin must conform to the specific format described below.

Blank lines are ignored and any part of a line to the right of a # symbol is treated as a comment. The first non-blank, non-comment line must indicate the resolution to be used for interpreting the ts_quantum time quantum values. The resolution is specified as

RES=*res*

where *res* is a positive integer between 1 and 1,000,000,000 inclusive and the resolution used is the reciprocal of *res* in seconds (for example, RES=1000 specifies millisecond resolution). Although very fine (nanosecond) resolution may be specified, the time quantum lengths are rounded up to the next integral multiple of the system clock's resolution.

The remaining lines in the file are used to specify the parameter values for each of the time-sharing priority levels. The first line specifies the parameters for time-sharing level 0, the second line specifies the parameters for time-sharing level 1, etc. There must be exactly one line for each configured time-sharing priority level.

EXAMPLE

The following excerpt from a dispadmin configuration file illustrates the format. Note that for each line specifying a set of parameters there is a comment indicating the corresponding priority level. These level numbers indicate priority within the time-sharing class, and the mapping between these time-sharing priorities and the corresponding global scheduling priorities is determined by the configuration specified in the ts master file. The level numbers are strictly for the convenience of the administrator reading the file and, as with any comment, they are ignored by dispadmin. dispadmin assumes that the lines in the file are ordered by consecutive, increasing priority level (from 0 to the maximum configured time-sharing priority). The level numbers in the comments should normally agree with this ordering; if for some reason they don't, however, dispadmin is unaffected.

```
# Time-Sharing Dispatcher Configuration File
RES=1000

# ts_quantum ts_tqexp ts_slpret ts_maxwait ts_lwait  PRIORITY LEVEL
      500        0        10          5         10          #  0
      500        0        11          5         11          #  1
      500        1        12          5         12          #  2
      500        1        13          5         13          #  3
      500        2        14          5         14          #  4
      500        2        15          5         15          #  5
      450        3        16          5         16          #  6
      450        3        17          5         17          #  7
       .         .         .          .          .          .  .
       .         .         .          .          .          .  .
       .         .         .          .          .          .  .
       50       48        59          5         59          #  58
       50       49        59          5         59          #  59
```

FILES

/usr/include/sys/ts.h

NOTES

dispadmin does some limited sanity checking on the values supplied in the configuration file. The sanity checking is intended to ensure that the new ts_dptbl values do not cause the system to panic. The sanity checking does not attempt to analyze the effect that the new values will have on the performance of the system. Unusual ts_dptbl configurations may have a dramatic negative impact on the performance of the system.

No sanity checking is done on the ts_dptbl values specified in the ts master file. Specifying an inconsistent or nonsensical ts_dptbl configuration through the ts master file could cause serious performance problems and/or cause the system to panic.

SEE ALSO

dispadmin(1M), mkboot(1M), priocntl(1), priocntl(2), master(4).

NAME

ttcompat - V7, 4BSD and XENIX STREAMS compatibility module

SYNOPSIS

```
#include <sys/stream.h>
#include <sys/stropts.h>
#include <sys/ttcompat.h>
#include <sys/ttold.h>

ioctl(fd, I_PUSH, "ttcompat");
```

DESCRIPTION

ttcompat is a STREAMS module that translates the ioctl calls supported by the older Version 7, 4BSD and XENIX terminal drivers into the ioctl calls supported by the termio interface [see termio(7)]. All other messages pass through this module unchanged; the behavior of read and write calls is unchanged, as is the behavior of ioctl calls other than the ones supported by ttcompat.

This module can be automatically pushed onto a stream using the autopush mechanism when a terminal device is opened; it does not have to be explicitly pushed onto a stream. This module requires that the termios interface be supported by the modules and the application can push the driver downstream. The TCGETS, TCSETS, and TCSETSF ioctl calls must be supported; if any information set or fetched by those ioctl calls is not supported by the modules and driver downstream, some of the V7/4BSD/XENIX functions may not be supported. For example, if the CBAUD bits in the c_cflag field are not supported, the functions provided by the sg_ispeed and sg_ospeed fields of the sgttyb structure (see below) will not be supported. If the TCFLSH ioctl is not supported, the function provided by the TIOCFLUSH ioctl will not be supported. If the TCXONC ioctl is not supported, the functions provided by the TIOCSTOP and TIOCSTART ioctl calls will not be supported. If the TIOCMBIS and TIOCMBIC ioctl calls are not supported, the functions provided by the TIOCSDTR and TIOCCDTR ioctl calls will not be supported.

The basic ioctl calls use the sgttyb structure defined by sys/ioctl.h:

```
struct sgttyb {
        char    sg_ispeed;
        char    sg_ospeed;
        char    sg_erase;
        char    sg_kill;
        int     sg_flags;
};
```

The sg_ispeed and sg_ospeed fields describe the input and output speeds of the device, and reflect the values in the c_cflag field of the termios structure. The sg_erase and sg_kill fields of the argument structure specify the erase and kill characters respectively, and reflect the values in the VERASE and VKILL members of the c_cc field of the termios structure.

The sg_flags field of the argument structure contains several flags that determine the system's treatment of the terminal. They are mapped into flags in fields of the terminal state, represented by the termios structure.

Delay type 0 is always mapped into the equivalent delay type 0 in the c_oflag field of the termios structure. Other delay mappings are performed as follows:

sg_flags	c_oflag
BS1	BS1
FF1	VT1
CR1	CR2
CR2	CR3
CR3	not supported
TAB1	TAB1
TAB2	TAB2
XTABS	TAB3
NL1	ONLRET|CR1
NL2	NL1

If previous TIOCLSET or TIOCLBIS ioctl calls have not selected LITOUT or PASS8 mode, and if RAW mode is not selected, the ISTRIP flag is set in the c_iflag field of the termios structure, and the EVENP and ODDP flags control the parity of characters sent to the terminal and accepted from the terminal:

Parity is not to be generated on output or checked on input:

The character size is set to CS8 and the flag is cleared in the c_cflag field of the termios structure.

Even parity characters are to be generated on output and accepted on input:

The flag is set in the c_iflag field of the termios structure, the character size is set to CS7 and the flag is set in the c_cflag field of the termios structure.

Odd parity characters are to be generated on output and accepted on input:

The flag is set in the c_iflag field, the character size is set to CS7 and the and flags are set in the c_cflag field of the termios structure.

Even parity characters are to be generated on output and characters of either parity are to be accepted on input:

The flag is cleared in the c_iflag field, the character size is set to CS7 and the flag is set in the c_cflag field of the termios structure.

The RAW flag disables all output processing (the OPOST flag in the c_oflag field, and the XCASE flag in the c_lflag field, are cleared in the termios structure) and input processing (all flags in the c_iflag field other than the IXOFF and IXANY flags are cleared in the termios structure). 8 bits of data, with no parity bit, are accepted on input and generated on output; the character size is set to CS8 and the PARENB and PARODD flags are cleared in the c_cflag field of the termios structure. The signal-generating and line-editing control characters are disabled by clearing the ISIG and ICANON flags in the c_lflag field of the termios structure.

The CRMOD flag turns input RETURN characters into NEWLINE characters, and output and echoed NEWLINE characters to be output as a RETURN followed by a LINEFEED. The ICRNL flag in the c_iflag field, and the OPOST and ONLCR flags in the c_oflag field, are set in the termios structure.

The LCASE flag maps upper-case letters in the ASCII character set to their lower-case equivalents on input (the IUCLC flag is set in the c_iflag field), and maps lower-case letters in the ASCII character set to their upper-case equivalents on output (the OLCUC flag is set in the c_oflag field). Escape sequences are accepted on input, and generated on output, to handle certain ASCII characters not supported by older terminals (the XCASE flag is set in the c_lflag field).

Other flags are directly mapped to flags in the termios structure:

sg_flags	flags in termios structure
CBREAK	complement of ICANON in c_lflag field
ECHO	ECHO in c_lflag field
TANDEM	IXOFF in c_iflag field

Another structure associated with each terminal specifies characters that are special in both the old Version 7 and the newer 4BSD terminal interfaces. The following structure is defined by sys/ioctl.h:

```
struct tchars {
        char   t_intrc;        /* interrupt */
        char   t_quitc;        /* quit */
        char   t_startc;       /* start output */
        char   t_stopc;        /* stop output */
        char   t_eofc;         /* end-of-file */
        char   t_brkc;         /* input delimiter (like nl) */
};
```

XENIX defines the tchar structure as tc. The characters are mapped to members of the c_cc field of the termios structure as follows:

tchars	c_cc index
t_intrc	VINTR
t_quitc	VQUIT
t_startc	VSTART
t_stopc	VSTOP
t_eofc	VEOF
t_brkc	VEOL

Also associated with each terminal is a local flag word, specifying flags supported by the new 4BSD terminal interface. Most of these flags are directly mapped to flags in the termios structure:

local flags	flags in termios structure
LCRTBS	not supported
LPRTERA	ECHOPRT in the c_lflag field
LCRTERA	ECHOE in the c_lflag field
LTILDE	not supported
LTOSTOP	TOSTOP in the c_lflag field
LFLUSHO	FLUSHO in the c_lflag field
LNOHANG	CLOCAL in the c_cflag field

LCRTKIL	ECHOKE in the c_lflag field
LCTLECH	CTLECH in the c_lflag field
LPENDIN	PENDIN in the c_lflag field
LDECCTQ	complement of IXANY in the c_iflag field
LNOFLSH	NOFLSH in the c_lflag field

Another structure associated with each terminal is the ltchars structure which defines control characters for the new 4BSD terminal interface. Its structure is:

```
struct ltchars {
        char   t_suspc;       /* stop process signal */
        char   t_dsuspc;      /* delayed stop process signal */
        char   t_rprntc;      /* reprint line */
        char   t_flushc;      /* flush output (toggles) */
        char   t_werasc;      /* word erase */
        char   t_lnextc;      /* literal next character */
};
```

The characters are mapped to members of the c_cc field of the termios structure as follows:

ltchars	c_cc index
t_suspc	VSUSP
t_dsuspc	VDSUSP
t_rprntc	VREPRINT
t_flushc	VDISCARD
t_werasc	VWERASE
t_lnextc	VLNEXT

ioctls

ttcompat responds to the following ioctl calls. All others are passed to the module below.

TIOCGETP The argument is a pointer to an sgttyb structure. The current terminal state is fetched; the appropriate characters in the terminal state are stored in that structure, as are the input and output speeds. The values of the flags in the sg_flags field are derived from the flags in the terminal state and stored in the structure.

TIOCEXCL Set "exclusive-use" mode; no further opens are permitted until the file has been closed.

TIOCNXCL Turn off "exclusive-use" mode.

TIOCSETP The argument is a pointer to an sgttyb structure. The appropriate characters and input and output speeds in the terminal state are set from the values in that structure, and the flags in the terminal state are set to match the values of the flags in the sg_flags field of that structure. The state is changed with a TCSETSF ioctl so that the interface delays until output is quiescent, then throws away any unread characters, before changing the modes.

TIOCSETN The argument is a pointer to an sgttyb structure. The terminal state is changed as TIOCSETP would change it, but a TCSETS ioctl is used, so that the interface neither delays nor discards input.

TIOCHPCL — The argument is ignored. The HUPCL flag is set in the c_cflag word of the terminal state.

TIOCFLUSH — The argument is a pointer to an int variable. If its value is zero, all characters waiting in input or output queues are flushed. Otherwise, the value of the int is treated as the logical OR of the FREAD and FWRITE flags defined by sys/file.h; if the FREAD bit is set, all characters waiting in input queues are flushed, and if the FWRITE bit is set, all characters waiting in output queues are flushed.

TIOCBRK — The argument is ignored. The break bit is set for the device.

TIOCCBRK — The argument is ignored. The break bit is cleared for the device.

TIOCSDTR — The argument is ignored. The Data Terminal Ready bit is set for the device.

TIOCCDTR — The argument is ignored. The Data Terminal Ready bit is cleared for the device.

TIOCSTOP — The argument is ignored. Output is stopped as if the STOP character had been typed.

TIOCSTART — The argument is ignored. Output is restarted as if the START character had been typed.

TIOCGETC — The argument is a pointer to a tchars structure. The current terminal state is fetched, and the appropriate characters in the terminal state are stored in that structure.

TIOCSETC — The argument is a pointer to a tchars structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.

TIOCLGET — The argument is a pointer to an int. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state and stored in the int pointed to by the argument.

TIOCLBIS — The argument is a pointer to an int whose value is a mask containing flags to be set in the local flags word. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state; the specified flags are set, and the flags in the terminal state are set to match the new value of the local flags word.

TIOCLBIC — The argument is a pointer to an int whose value is a mask containing flags to be cleared in the local flags word. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state; the specified flags are cleared, and the flags in the terminal state are set to match the new value of the local flags word.

TIOCLSET — The argument is a pointer to an int containing a new set of local flags. The flags in the terminal state are set to match the new value of the local flags word.

TIOCGLTC	The argument is a pointer to an ltchars structure. The values of the appropriate characters in the terminal state are stored in that structure.
TIOCSLTC	The argument is a pointer to an ltchars structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.
FIORDCHK	FIORDCHK returns the number of immediately readable characters. The argument is ignored.
FIONREAD	FIONREAD returns the number of immediately readable characters in the int pointed to by the argument.
LDSMAP	Calls the function emsetmap(*tp*, *mp*) if the function is configured in the kernel.
LDGMAP	Calls the function emgetmap(*tp*, *mp*) if the function is configured in the kernel.
LDNMAP	Calls the function emunmap(*tp*, *mp*) if the function is configured in the kernel.

The following ioctls are returned as successful for the sake of compatibility. However, nothing significant is done (that is, the state of the terminal is not changed in any way).

TIOCSETD	LDOPEN
TIOCGETD	LDCLOSE
DIOCSETP	LDCHG
DIOCSETP	LDSETT
DIIOGETP	LDGETT

SEE ALSO

ioctl(2), termios(2), termio(7), ldterm(7)

NOTES

TIOCBRK and TIOCCBRK should be handled by the driver. FIONREAD and FIORDCHK are handled in the stream head.

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

console(7), ports(7)

NAME

`ttydefs` - file contains terminal line settings information for `ttymon`

DESCRIPTION

`/etc/ttydefs` is an administrative file that contains information used by `ttymon` to set up the speed and terminal settings for a TTY port.

The `ttydefs` file contains the following fields:

ttylabel — The string `ttymon` tries to match against the TTY port's *ttylabel* field in the port monitor administrative file. It often describes the speed at which the terminal is supposed to run, for example, `1200`.

initial-flags — Contains the initial `termio`(7) settings to which the terminal is to be set. For example, the system administrator will be able to specify what the default erase and kill characters will be. *initial-flags* must be specified in the syntax recognized by the `stty` command.

final-flags — *final-flags* must be specified in the same format as *initial-flags*. `ttymon` sets these final settings after a connection request has been made and immediately prior to invoking a port's service.

autobaud — If the autobaud field contains the character 'A', autobaud will be enabled. Otherwise, autobaud will be disabled. `ttymon` determines what line speed to set the TTY port to by analyzing the carriage returns entered. If autobaud has been disabled, the hunt sequence is used for baud rate determination.

nextlabel — If the user indicates that the current terminal setting is not appropriate by sending a BREAK, `ttymon` searchs for a `ttydefs` entry whose *ttylabel* field matches the *nextlabel* field. If a match is found, `ttymon` uses that field as its *ttylabel* field. A series of speeds is often linked together in this way into a closed set called a hunt sequence. For example, `4800` may be linked to `1200`, which in turn is linked to `2400`, which is finally linked to `4800`.

SEE ALSO

`sttydefs`(1M), `ttymon`(1M).

NAME

`ttysrch` - directory search list for ttyname

DESCRIPTION

`ttysrch` is an optional file that is used by the `ttyname` library routine. This file contains the names of directories in `/dev` that contain terminal and terminal-related device files. The purpose of this file is to improve the performance of `ttyname` by indicating which subdirectories in `/dev` contain terminal-related device files and should be searched first. These subdirectory names must appear on separate lines and must begin with `/dev`. Those path names that do not begin with `/dev` will be ignored and a warning will be sent to the console. Blank lines (lines containing only white space) and lines beginning with the comment character "#" will be ignored. For each file listed (except for the special entry `/dev`), `ttyname` will recursively search through subdirectories looking for a match. If `/dev` appears in the `ttysrch` file, the `/dev` directory itself will be searched but there will not be a recursive search through its subdirectories.

When `ttyname` searches through the device files, it tries to find a file whose major/minor device number, file system identifier, and inode number match that of the file descriptor it was given as an argument. If a match is not found, it will settle for a match of just major/minor device and file system identifier, if one can be found. However, if the file descriptor is associated with a cloned device (see clone(7)), this algorithm does not work efficiently because the inode number of the device file associated with a clonable device will never match the inode number of the file descriptor that was returned by the open of that clonable device. To help with these situations, entries can be put into the `/etc/ttysrch` file to improve performance when cloned devices are used as terminals on a system (for example, for remote login). However, this is only useful if the minor devices related to a cloned device are put into a subdirectory. (It is important to note that device files need not exist for cloned devices and if that is the case, `ttyname` will eventually fail.) For example if `/dev/tcp` is a cloned device, there could be a subdirectory `/dev/inet` that contains files `tcp000`, `tcp001`, `tcp002`, etc. that correspond to the minor devices of the starlan driver. An optional second field is used in the `/etc/ttysrch` file to indicate the matching criteria. This field is separated by white space (any combination of blanks or tabs). The letter `M` means major/minor device number, `F` means file system identifier, and `I` means inode number. If this field is not specified for an entry, the default is `MFI` which means try to match on all three. For cloned devices the field should be `MF`, which indicates that it is not necessary to match on the inode number.

There is another option called `A` which means alias. This option is immediately followed by the full path name (must also begin with `/dev`) of the alias for the device. After finding a device name (matching `MFI`), if the option `A` is present, `ttyname` appends the minor device number of the found device to the provided alias to form a new name. Then it checks the aliased device to make sure it is the same as the found device and returns the new name. For example, if `/dev/pts0` is hard linked to `/dev/pts/0` and the alias option is present, `ttyname()` returns `/dev/pts0`.

Without the `/etc/ttysrch` file, `ttyname` will search the `/dev` directory by first looking in the directories `/dev/term`, `/dev/pts`, and `/dev/xt`. If a system has terminal devices installed in directories other than these, it may help performance if the `ttysrch` file is created and contains that list of directories.

The command ps(1) maintains a database of terminal device names. If /etc/ttysrch is modified, the database file /etc/ps_data should be removed. Removing the database causes it to be automatically rebuilt.

EXAMPLE

A sample /etc/ttysrch file follows:

```
/dev/term  MFI
/dev/pts        MFI
/dev/xt         MFI
/dev/inet  MF
```

This file tells ttyname that it should first search through those directories listed and that when searching through the /dev/inet directory, if a file is encountered whose major/minor devices and file system identifier match that of the file descriptor argument to ttyname, this device name should be considered a match.

A sample /etc/ttysrch file for the alias option follows:

```
/dev/term  MFI
/dev/pts   A/dev/pts
/dev/xt         MFI
/dev/inet  MF
```

The second line in this file tells ttyname to return /dev/pts0 for /dev/pts/0, /dev/pts1 for /dev/pts/1 etc.

FILES

/etc/ps_data, /etc/ttysrch

SEE ALSO

ps(1), ttyname(3C), clone(7)

NAME

UDP - Internet User Datagram Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

t = t_open("/dev/udp", O_RDWR);
```

DESCRIPTION

UDP is a simple datagram protocol which is layered directly above the Internet Protocol (IP). Programs may access UDP using the socket interface, where it supports the SOCK_DGRAM socket type, or using the Transport Level Interface (TLI), where it supports the connectionless (T_CLTS) service type.

Within the socket interface, UDP is normally used with the sendto(), sendmsg(), recvfrom(), and recvmsg() calls [see send(2) and recv(2)]. If the connect(2) call is used to fix the destination for future packets, then the recv(2) or read(2) and send(2) or write(2) calls may be used.

UDP address formats are identical to those used by the Transmission Control Protocol (TCP). Like TCP, UDP uses a port number along with an IP address to identify the endpoint of communication. The UDP port number space is separate from the TCP port number space (that is, a UDP port may not be connected to a TCP port). The bind(2) call can be used to set the local address and port number of a UDP socket. The local IP address may be left unspecified in the bind() call by using the special value INADDR_ANY. If the bind() call is not done, a local IP address and port number will be assigned to the endpoint when the first packet is sent. Broadcast packets may be sent (assuming the underlying network supports this) by using a reserved broadcast address. This address is network interface dependent. Broadcasts may only be sent by the privileged user.

Options at the IP level may be used with UDP; see ip(7).

There are a variety of ways that a UDP packet can be lost or corrupted, including a failure of the underlying communication mechanism. UDP implements a checksum over the data portion of the packet. If the checksum of a received packet is in error, the packet will be dropped with no indication given to the user. A queue of received packets is provided for each UDP socket. This queue has a limited capacity. Arriving datagrams which will not fit within its *high-water* capacity are silently discarded.

UDP processes Internet Control Message Protocol (ICMP) error messages received in response to UDP packets it has sent. See icmp(7). ICMP source quench messages are ignored. ICMP destination unreachable, time exceeded and parameter problem messages disconnect the socket from its peer so that subsequent attempts to send packets using that socket will return an error. UDP will not guarantee that packets are delivered in the order they were sent. As well, duplicate packets may be generated in the communication process.

SEE ALSO

read(2), write(2), bind(3N), connect(3N), recv(3N), send(3N), icmp(7), inet(7), ip(7), tcp(7)

Postel, Jon, *User Datagram Protocol*, RFC 768, Network Information Center, SRI International, Menlo Park, Calif., August 1980

DIAGNOSTICS

A socket operation may fail if:

EISCONN	A connect() operation was attempted on a socket on which a connect() operation had already been performed, and the socket could not be successfully disconnected before making the new connection.
EISCONN	A sendto() or sendmsg() operation specifying an address to which the message should be sent was attempted on a socket on which a connect() operation had already been performed.
ENOTCONN	A send() or write() operation, or a sendto() or sendmsg() operation not specifying an address to which the message should be sent, was attempted on a socket on which a connect() operation had not already been performed.
EADDRINUSE	A bind() operation was attempted on a socket with a network address/port pair that has already been bound to another socket.
EADDRNOTAVAIL	A bind() operation was attempted on a socket with a network address for which no network interface exists.
EINVAL	A sendmsg() operation with a non-NULL msg_accrights was attempted.
EACCES	A bind() operation was attempted with a reserved port number and the effective user ID of the process was not the privileged user.
ENOBUFS	The system ran out of memory for internal data structures.

NAME

unistd - header file for symbolic constants

SYNOPSIS

```
#include <unistd.h>
```

DESCRIPTION

The unistd.h header file defines the symbolic constants and structures not already defined or declared in some other header file. The contents of this file are shown below.

The following symbolic constants are defined for the access function [see access(2)]:

R_OK	Test for read permission.
W_OK	Test for write permission.
X_OK	Test for execute (search) permission.
F_OK	Test for existence of file.

The constants F_OK, R_OK, W_OK and X_OK and the expressions R_OK | W_OK, R_OK | X_OK and R_OK | W_OK | X_OK all have distinct values.

Declares the constant

NULL	null pointer

The following symbolic constants are defined for the lockf function [see lockf(3C)]:

F_ULOCK	Unlock a previously locked region.
F_LOCK	Lock a region for exclusive use.
F_TLOCK	Test and lock a region for exclusive use.
F_TEST	Test a region for other processes locks.

The following symbolic constants are defined for the lseek [see lseek(2)] and fcntl [see fcntl(2)] functions (they have distinct values):

SEEK_SET	Set file offset to *offset*.
SEEK_CUR	Set file offset to current plus *offset*.
SEEK_END	Set file offset to EOF plus *offset*.

The following symbolic constants are defined (with fixed values):

_POSIX_VERSION	Integer value indicating version of the POSIX standard.
_XOPEN_VERSION	Integer value indicating version of the XPG to which system is compliant.

The following symbolic constants are defined to indicate that the option is present:

_POSIX_JOB_CONTROL	Implementation supports job control.
_POSIX_SAVED_IDS	The exec functions [see exec(2)] save the effective user and group.
_POSIX_VDISABLE	Terminal special characters defined in termios.h [see termio(7)] can be disabled using this character.

The following symbolic constants are defined for sysconf [see sysconf(3C)]:

_SC_ARG_MAX
_SC_CHILD_MAX
_SC_CLK_TCK
_SC_JOB_CONTROL
_SC_LOGNAME_MAX
_SC_NGROUPS_MAX
_SC_OPEN_MAX
_SC_PAGESIZE
_SC_PASS_MAX
_SC_SAVED_IDS
_SC_VERSION
_SC_XOPEN_VERSION

The following symbolic constants are defined for pathconf [see fpathconf(2)]:

_PC_CHOWN_RESTRICTED
_PC_LINK_MAX
_PC_MAX_CANON
_PC_MAX_INPUT
_PC_NAME_MAX
_PC_NO_TRUNC
_PC_PATH_MAX
_PC_PIPE_BUF
_PC_VDISABLE

The following symbolic constants are defined for file streams:

STDIN_FILENO	File number of stdin. It is 0.
STDOUT_FILENO	File number of stout. It is 1.
STDERR_FILENO	File number of stderr. It is 2.

The following pathnames are defined:

GF_PATH	Pathname of the group file.
PF_PATH	Pathname of the passwd file.

NOTES

The following values for constants are defined for this release of System V:

_POSIX_VERSION	198808L
_XOPEN_VERSION	3

SEE ALSO

access(2), exec(2), fcntl(2), fpathconf(2), lseek(2), termios(2), sysconf(3C), group(4), passwd(4), termio(7)

NAME

updaters - configuration file for Network Information Service (NIS) updating

SYNOPSIS

/var/yp/updaters

DESCRIPTION

The file /var/yp/updaters is a makefile [see make(1)] which is used for updating NIS databases. Databases can only be updated in a secure network, that is, one that has a publickey(4) database. Each entry in the file is a make target for a particular NIS database. For example, if there is a NIS database named publickey.byname that can be updated, there should be a make target named publickey.byname in the updaters file with the command to update the file.

The information necessary to make the update is passed to the update command through standard input. The information passed is described below (all items are followed by a NEWLINE, except for the actual bytes of key and actual bytes of date).

network name of client wishing to make the update (a string)

kind of update (an integer)

number of bytes in key (an integer)

actual bytes of key

number of bytes in data (an integer)

actual bytes of data

After getting this information through standard input, the command to update the particular database should decide whether the user is allowed to make the change. If not, it should exit with the status YPERR_ACCESS. If the user is allowed to make the change, the command should make the change and exit with a status of zero. If there are any errors that may prevent the updater from making the change, it should exit with the status that matches a valid NIS error code described in <rpcsvc/ypclnt.h>.

FILES

/var/yp/updaters

SEE ALSO

make(1), ypupdated(1M), ypupdate(3), publickey(4)

NAME

utmp, wtmp - utmp and wtmp entry formats

SYNOPSIS

```
#include <utmp.h>
```

DESCRIPTION

These files, which hold user and accounting information for such commands as who, write, and login, have the following structure, defined in utmp.h for the M88000 family of processors reference platform:

```
#define   UTMP_FILE     "/var/adm/utmp"
#define   WTMP_FILE     "/var/adm/wtmp"
#define   ut_name       ut_user

struct exit_status
  {
    short e_termination ;    /* Process termination status */
    short e_exit ;           /* Process exit status */
  };

struct     utmp {
   char    ut_user[8];       /* user login name */
   char    ut_id[4];         /* /etc/inittab id (created by */
                             /* process that puts entry in utmp) */
   char    ut_line[12];      /* device name (console, lnxx) */
   pid_t   ut_pid;           /* process id */
   short   ut_type;          /* type of entry */
#ifdef m88k
   short ut_pad ;            /* BCS 10.1 */
#endif /* m88k */
   struct  exit_status ut_exit; /* exit status of a process
                                 * marked as DEAD_PROCESS
                                 */
   time_t  ut_time;          /* time entry was made */
#ifdef m88k
   char ut_host[24];         /* hostname, if remote(BCS) */
#endif /* m88k */
};

/*  Definitions for ut_type  */

#define EMPTY           0
#define RUN_LVL         1
#define BOOT_TIME       2
#define OLD_TIME        3
#define NEW_TIME        4
#define INIT_PROCESS    5   /* process spawned by "init" */
#define LOGIN_PROCESS   6   /* a "getty" process waiting for login */
```

```
#define USER_PROCESS   7   /* a user process */
#define DEAD_PROCESS   8
#define ACOUNTING      9
#ifdef m88k
#define FTP            128
#define REMOTE_LOGIN   129
#define REMOTE_PROCESS 130
#endif /* m88k */

#ifdef m88k
#define UTMAXTYPE       REMOTE_PROCESS  /* Largest legal value of ut_type */
#endif /* m88k */
#ifdef m68k
#define UTMAXTYPE       ACCOUNTING      /* Largest legal value of ut_type */
#endif /* m68k */

/*  Below are special strings or formats used in the "ut_line" */
/*  field when  accounting for something other than a process.  */
/*  No string for the ut_line field can be more than 11 chars +  */
/*  a null character in length.  */

#define RUNLVL_MSG    "run-level %c"
#define BOOT_MSG      "system boot"
#define OTIME_MSG     "old time"
#define NTIME_MSG     "new time"
```

FILES

/var/adm/utmp
/var/adm/wtmp

SEE ALSO

login(1), who(1), write(1),
getut(3C)

NAME

utmpx, wtmpx - utmpx and wtmpx entry formats

SYNOPSIS

#include <utmpx.h>

DESCRIPTION

utmpx(4) is an extended version of utmp(4).

These files, which hold user and accounting information for such commands as who, write, and login, have the following structure as defined by utmpx.h:

```
#define   UTMPX_FILE   "/var/adm/utmpx"
#define   WTMPX_FILE   "/var/adm/wtmpx"
#define   ut_name     ut_user
#define   ut_xtime    ut_tv.tv_sec

struct utmpx  {
   char    ut_user[32];             /* user login name */
   char    ut_id[4];                /* inittab id */
   char    ut_line[32];             /* device name (console, lnxx) */
   pid_t   ut_pid;                  /* process id */
   short   ut_type;                 /* type of entry */
   struct exit_status ut_exit;      /* process termination/exit status */
   struct timeval ut_tv;            /* time entry was made */
   long    ut_session;              /* session ID, used for windowing */
   long    pad[5];                  /* reserved for future use */
   short   ut_syslen;               /* significant length of ut_host */
                                    /* including terminating null */
   char    ut_host[257];            /* remote host name */
  } ;

/* Definitions for ut_type */

#define   EMPTY           0
#define   RUN_LVL         1
#define   BOOT_TIME       2
#define   OLD_TIME        3
#define   NEW_TIME        4
#define   INIT_PROCESS    5  /* Process spawned by "init" */
#define   LOGIN_PROCESS   6  /* A "getty" process waiting for login */
#define   USER_PROCESS    7  /* A user process */
#define   DEAD_PROCESS    8
#define   ACCOUNTING      9

#define   UTMAXTYPE  ACCOUNTING  /* Largest legal value of ut_type */

/* Below are special strings or formats used in the "ut_line" */
/* field when accounting for something other than a process. */
/* No string for the ut_line field can be more than 11 chars + */
/* a null character in length. */

#define   RUNLVL_MSG     "run-level %c"
#define   BOOT_MSG       "system boot"
#define   OTIME_MSG      "old time"
#define   NTIME_MSG      "new time"
#define   MOD_WIN        10
```

FILES

/var/adm/utmpx
/var/adm/wtmpx

SEE ALSO

login(1), who(1), write(1) getutx(3C)

NAME

vfstab - table of file system defaults

SYNOPSIS

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

DESCRIPTION

The file /etc/vfstab describes defaults for each file system.

There are seven whitespace-separated fields in this table. Each field is described below.

The first field contains the block special device for mounting a local file system, a resource description if an RFS resource is to be mounted, or a remote directory (in the form host:directory-name) if an NFS mount is desired.

The second field should contain the character special device corresponding to the block special device in the first field if a local file system mount is specified, or a '-' if an RFS or NFS mount is specified.

The third field specifies the absolute path name of the mount directory.

The fourth field specifies the the file system type. For local file systems, this field should contain 's5', 'ufs', or 'bfs' for fast file system (UFS), system five file system (s5), and boot file system (BFS) mounts respectively. This field should contain the string 'rfs' or 'nfs' for RFS and NFS remote mounts respectively.

The fifth field specifies the fsck pass number. This field should contain a '-' for RFS and NFS mounts. Local mount requests may be grouped into passes, with all mounts in a given pass being checked by fsck before the next pass is performed. The pass numbers should start with one, and increase by one.

The sixth field specifies whether the mount request should be automatically initiated at boot time. This field should contain the string 'yes' or 'no'.

The seventh field specifies the mount options appropriate for the type of mount requested. Typically this field contains the string 'rw', which allows reading and writing of the mount, or 'ro', which specifies that the mount is read-only. Other values for this field are possible; for more information please refer to the appropriate mount man page listed in the "SEE ALSO" section below.

Empty lines and lines containing a '#' in the first column are ignored.

Each field in this file is also associated with a structure, defined in sys/vfstab.h:

```
struct vfstab {
      char   *vfs_special;
      char   *vfs_fsckdev;
      char   *vfs_mountp;
      char   *vfs_fstype;
      char   *vfs_fsckpass;
      char   *vfs_automnt;
      char   *vfs_mntopts;
};
```

The getvfsent(3C) family of routines are used to read and write to /etc/vfstab.

SEE ALSO

fsck(1M), mount(1M), setmnt(1M), mountall(1M), mount_ufs(1M), mount_s5(1M), mount_bfs(1M), mount_rfs(1M), mount_nfs(1M), getvfsent(3C).

NAME

ypfiles - the Network Information Service (NIS) database and directory structure

DESCRIPTION

The NIS network lookup service uses a distributed, replicated database of dbm files contained in the /var/yp directory hierarchy on each NIS server. A dbm database consists of two files, one has the filename extension .pag and the other has the filename extension .dir. For instance, the database named publickey, is implemented by the pair of files publickey.pag and publickey.dir.

A dbm database served by the NIS is called a NIS *map*. A NIS *ypdomain* is a subdirectory of /var/yp containing a set of NIS maps. Any number of NIS domains can exist. Each may contain any number of maps.

No maps are required by the NIS lookup service itself, although they may be required for the normal operation of other parts of the system. There is no list of maps which NIS serves — if the map exists in a given domain, and a client asks about it, the NIS will serve it. For a map to be accessible consistently, it must exist on all NIS servers that serve the domain. To provide data consistency between the replicated maps, an entry to run ypxfr periodically should be made in the privileged user's crontab file on each server. More information on this topic is in ypxfr(1M).

NIS maps should contain two distinguished key-value pairs. The first is the key YP_LAST_MODIFIED, having as a value a ten-character ASCII order number. The order number should be the system time in seconds when the map was built. The second key is YP_MASTER_NAME, with the name of the NIS master server as a value. makedbm(1M) generates both key-value pairs automatically. A map that does not contain both key-value pairs can be served by the NIS, but the ypserv process will not be able to return values for "Get order number" or "Get master name" requests. See ypserv(1M). In addition, values of these two keys are used by ypxfr when it transfers a map from a master NIS server to a slave. If ypxfr cannot figure out where to get the map, or if it is unable to determine whether the local copy is more recent than the copy at the master, extra command line switches must be set when it is run.

NIS maps must be generated and modified only at the master server. They are copied to the slaves using ypxfr(1M) to avoid potential byte-ordering problems among NIS servers running on machines with different architectures, and to minimize the amount of disk space required for the dbm files. The NIS database can be initially set up for both masters and slaves by using ypinit(1M).

After the server databases are set up, it is probable that the contents of some maps will change. In general, some ASCII source version of the database exists on the master, and it is changed with a standard text editor. The update is incorporated into the NIS map and is propagated from the master to the slaves by running /var/yp/Makefile, see ypmake(1M). All default maps have entries in /var/yp/Makefile; if a NIS map is added, edit this file to support the new map. The makefile uses makedbm(1M) to generate the NIS map on the master, and yppush(1M) to propagate the changed map to the slaves. yppush is a client of the map ypservers, which lists all the NIS servers. For more information on this topic, see yppush(1M).

FILES

/var/yp
/var/yp/aliases
/var/yp/Makefile

SEE ALSO

makedbm(1M), ypinit(1M), ypmake(1M), yppoll(1M), yppush(1M), ypserv(1M), ypxfr(1M), dbm(3), publickey(4)

NAME

zero - source of zeroes

DESCRIPTION

A zero special file is a source of zeroed unnamed memory.

Reads from a zero special file always return a buffer full of zeroes. The file is of infinite length.

Writes to a zero special file are always successful, but the data written is ignored.

Mapping a zero special file creates a zero-initialized unnamed memory object of a length equal to the length of the mapping and rounded up to the nearest page size as returned by `sysconf`. Multiple processes can share such a zero special file object provided a common ancestor mapped the object `MAP_SHARED`.

FILES

`/dev/zero`

SEE ALSO

`fork`(2), `sysconf`(3C), `mmap`(2)

Permuted Index

MOTOROLA

The reference manual set for UNIX System V Release 4 for Motorola Processors is the definitive source for complete and detailed specifications for all System V interfaces. Retitled and reorganized, this edition makes finding the manual page you need fast and easy. The following table reflects these changes.

Commands Reference Manual Volumes 1 and 2

- General-purpose user commands
- Basic networking commands
- Form and Menu Language Interpreter (FMLI)
- System maintenance commands
- Enhanced networking commands
- Miscellaneous reference information related to commands

System Files and Devices Reference Manual

- System file formats
- Special files (devices)

Device Driver Interface/Driver-Kernel Interface Reference Manual

- Driver Data Definitions
- Driver Entry Point Routines
- Kernel Utility Routines
- Kernel Data Structures
- Kernel Defines

System Calls and Library Functions Reference Manual

- System calls
- BSD system compatibility library
- Standard C library
- Executable and linking format library
- General-purpose library
- Math library
- Networking library
- Standard I/O library
- Specialized library
- Miscellaneous reference information related to programming

Master Permuted Index

- Permuted index of all manual pages

ISBN 0-13-035874-6

90000

9780130358745

UNIX
PRESS

A Prentice Hall Title