m4 — Command

Macro processor
m4 [file ...]

The command **m4** processes macros. It allows you to define strings for which **m4** is to search, and strings to replace them; **m4** then opens *file*, reads its contents, replaces each macro with its specified replacement string, and writes the results into the standard output stream.

m4 can also manipulate files, make conditional decisions, select substrings, and perform arithmetic. The tutorial *Introduction to the m4 Macro Processor* introduces **m4** in detail.

m4 reads its *file*s in the order given; if no *file* is named, then it reads the standard input stream. The file name '-' indicates the standard input.

m4 copies input to output until it finds a potential *macro*. A macro is a string of alphanumerics (letters, digits, or underscores) that begins with a non-digit character and is surrounded by non-alphanumerics. If **m4** does not recognize the *macro*, it simply copies it to the output and continues processing. If **m4** recognizes the *macro* and the next character is a left parenthesis '(', an *argument set* follows:

```
	macro(arg1,..., argn)
```

The arguments are collected by processing them in the same manner as other text (thus, an arguments may itself be another macro), and resulting output text is diverted into storage. **m4** stores up to nine arguments; any more will be processed but not saved. An argument set consists of strings of text separated by commas (commas inside quotation marks or parentheses do not terminate an argument), and must contain balanced parentheses that are free of quotation marks (i.e., that are *unquoted*). **m4** strips arguments of unquoted leading space (blanks, tabs, newline characters).

m4 then removes the *macro* and its optional argument set from the input stream, processes them, and replaces them in the input stream with the resulting value. The value becomes the next piece of text to be read.

Quotation marks, of the form ‘ ’, inhibit the recognition of *macro*. **m4** strips off one level of quotation marks when it encounters them (quotation marks are nestable). Thus, ‘*macro*’ is not processed, but is changed to *macro* and passed on.

m4 determines the *value* of a user-defined macro by taking the text that constitutes the macro's *definition* and replacing any occurrence within that text of '$*n*' (where *n* is '0' through '9') with the text of the *n*th argument. Argument 0 is the *macro* itself.

m4 recognizes the following predefined macros:

changequote[([*openquote***],[***closequote***])]**
Changes the quotation characters. Missing arguments default to ‘ for open or ’ for close. Quotation characters will not nest if they are defined to be the same character. Value is null.

decr[(*number***)]**
Decrement *number* (default, 0) by one and returns resulting value.

define(*macro,definition***)**
Define or redefine *macro*. If a predefined macro is redefined, its original definition is irrecoverably lost. Value is null.

divert[(*n***)]**
Redirects output to output stream *n* (default is zero). The standard output is zero, and one through nine are maintained as temporary files. Any other *n* results in output being thrown away until the next **divert** macro. Value is null.

divnum
Value is current output stream number.

dnl Delete to newline: removes all characters from the input stream up to and including the next newline. Value is null.

dumpdef[(*macros***)]**
Value is quoted definitions of all *macro*s specified, or names and definitions of all defined macros if no arguments.

errprint(*text***)**
Print *text* on standard error file. Value is null.

eval(*expression***)**
Value is a number that is the value of evaluated *expression*. It recognizes, in order of decreasing precedence: parentheses, **, unary + -, * / %, binary + -, relations, and logicals. Arithmetic is performed in **long**s.

ifdef(*macro,defvalue,undefvalue***)**
Return *defvalue* if *macro* is defined, and *undefvalue* if not.

ifelse(*arg1,arg2,arg3...***)**
Compares *arg1* and *arg2*. If they are the same, returns *arg3*. If not, and *arg4* is the last argument, return *arg4*. Otherwise, the process repeats, comparing *arg4* and *arg5*, and so on. Like other **m4** macros, this takes a maximum of nine arguments.

include(*file***)**
Value is the entire contents of the *file* argument. If *file* is not accessible, a fatal error results.

incr[(*number***)]**
Increments given *number* (default, zero) by one and returns resulting value.

index(*text,pattern***)**
Value is a number corresponding to position of *pattern* in *text*. If *pattern* does not occur in *text*, value is -1.

len(*text***)**
Value is a number that corresponds to length of *text*.

maketemp(*filenameXXXXXX***)**
Value is *filename* with last six characters, usually **XXXXXX**, replaced with current process id and a single letter. Same as the COHERENT system call **mktemp()**.

sinclude(*file***)**
Value is the entire contents of *file*. If *file* is not accessible, return null and continue processing.

substr(*text***[,***start***[,***count***]])**
Value is a substring of *text*. *start* may be left-oriented (nonnegative) or right-oriented (negative). *count* specifies how many characters to the right (if positive) or to the left (if negative) to return. If absent, it is assumed to be large and of the same sign as *start*. If *start* is omitted, it is assumed to be zero if *count* is positive or omitted, or -1 if *count* is negative.

syscmd(*command***)**
Pass *command* to the shell for execution. Value is null. Same as system call **system**.

translit(*text,characters***[,***replacements***])**
Replaces *characters* in *text* with the corresponding characters from *replacements*. If the *replacements* is absent or too short, replace *characters* with a null character. Value is *text* with specified replacements.

undefine(*macro***)**
Remove macro definition. Value is null. If a predefined macro is redefined, its original definition is irrecoverably lost.

undivert[(*stream***[,...])]**
Dumps each specified *stream* into the current output stream. With no arguments, **undivert** dumps all output streams in numeric order. **m4** will not dump any output stream into itself. At the end of processing, **m4** automatically dumps all diverted text to standard output in numeric order. Value is null.

See Also

commands, mktemp(), system
Introduction to the m4 Macro Processor

machine.h — Header File

Machine-dependent definitions
#include <sys/machine.h>

machine.h defines macros, constants, and structures that are specific to the machine upon which COHERENT is being run.

See Also

header files

Notes

This header file is obsolete, and will be dropped from a future release of COHERENT. Its use is strongly discouraged.

macro — Definition

A **macro** is a body of text that is given a name. When the name is used in a program, it is replaced with the text to which it refers; this is called macro *expansion*. For example, **getchar()** is a macro that consists of the function call **getc(stdin)**. The C preprocessor recognizes two varieties of macros: *object-like* and *function-like*.

When the C compiler performs macro substitution, all escape sequences and trigraphs have been resolved. After a macro has been expanded, the expanded text is scanned again to see if the expansion itself contains any macros (not including the original macro that has already been expanded). This re-scanning continues until no further replacement is possible.

Most macros are defined in C headers. The C preprocessor, however, defines some others.

See Also

#define, C preprocessor, m4, manifest constant, Programming COHERENT
ANSI Standard, §3.8.3

madd() — Multiple-Precision Mathematics (libmp)

Add multiple-precision integers
#include <mprec.h>
void madd(*a, b, c***)**
mint **a*, **b*, **c***;**

madd() sets the multiple-precision integer (or **mint**) pointed to by *c* to the sum of the the **mint**s pointed to by *a* and *b*.

See Also

libmp

mail — Command

Send or read mail
mail [-mpqrv] [-f *file***] [***user ...***]**

mail allows you to exchange electronic mail with other COHERENT system users, either on your own system or on other systems via UUCP. Depending upon its form, this command can be used either to send mail to other users or to read the mail that other users have sent to you.

Sending Mail

If you name one or more *user*s, **mail** assumes that you wish to send a mail message to each *user*. **mail** first prints the prompt

```
        Subject:
```

on the screen, requesting that you give the message a title.

mail then reads what you type on the standard input. A message is terminated by **<ctrl-D>**, by a line that contains only the character '.', or by a line that contains only the character '?'. Ending with a question mark prompts **mail** to feed the message into an editor for further editing. The editor used is the one named in the environmental variable **EDITOR**. If this variable is not defined, **mail** uses **ed**.

If you have defined environmental variable **ASKCC** to **YES**, **mail** asks you, after a message is ended, for a list of users to whom you wish to send a copy of the message.

Finally, **mail** prepends the date and the sender's name, and sends the result to each *user* named either on the command line or on the carbon-copy list with the **rmail** command.

Each *user* who has received mail is greeted by the message "You have mail." when she logs in. **mail** normally changes the contents of the mailbox as the user works with them; however, **mail** has options that allow the contents of the mailbox to remain unchanged if the user desires.

Reading Mail

If no *user* is named on its command line, **mail** reads and displays the user's mail, message by message. If environmental variable **PAGER** is defined, **mail** will "pipe" each message through the command it names. For example, the **.profile** command line:

```
export PAGER="exec /bin/scat -1"
```

invokes **/bin/scat** for each mail message with the command-line argument **-1** (the digit one).

While reading mail, the user can use any of the following commands to save, delete, or send each message to another user interactively.

d Delete the current message and print the next message.

m [*user* ...]
Mail the current message to each *user* given (default: yourself).

p Print the current message again.

q Quit, and update mailbox file to reflect changes.

r Reverse the direction in which the mailbox is being scanned.

s [*file* ...]
Save the current mail message with the usual header in each *file* (default: **$HOME/mbox**).

t [*user* ...]
Send a message read from the standard input, terminated by an end-of-file character or by a line containing only '.' or '?', to each *user* (default: yourself).

w [*file* ...]
Write the current message without the usual header in each *file* (default: **$HOME/mbox**).

x Exit without updating the mailbox file.

<newline>
Print the next message.

- Print the previous message.

EOF Quit, updating mailbox; same as **q**.

? Print a summary of available commands.

!*command*
Pass *command* to the shell for execution.

The following command line options control the sending and reading of mail.

-f *file* Read mail from *file* instead of from the default, **/usr/spool/mail/***user*.

-m Send a message to the terminal of *user* if he is logged into the system when mail is sent.

-p Print all mail without interaction.

-q Quit without changing the mailbox if an interrupt character is typed. Normally, an interrupt character stops printing of the current message.

-r Reverse the order of printing messages. Normally, **mail** prints messages in the order in which they were received.

-v Verbose mode. Show the version number of the **mail** program, and display expanded aliases.

If you wish, you can create a signature file, **.signature**, in your home directory. **mail** appends the contents of the signature file to the end of every mail message you send, as a signature. A signature can be your system's path name (for **uucp** messages), your telephone number, an amusing *bon mot*, or what you will.

Files

$HOME/dead.letter — Message that **mail** could not send
$HOME/mbox — Default saved mail
$HOME/.signature — Signature file
/etc/domain — Name of your system's domain
/etc/uucpname — Name of your system
/tmp/mail* — Temporary and lock files
/usr/spool/mail — Mailbox directory, filed by user name

See Also

aliases, ASKCC, commands, EDITOR, .forward, mkfnames, msg, nptx, PAGER, paths, rmail, smail, uux

Notes

Note that before you can send mail, either locally or to a remote site, you must run the program **uuinstall** and use its 'S' option to set the name of your local site and domain. Your local system must, of course, also have permission to log into any remote site to which you wish to send mail. See the tutorial and Lexicon articles on UUCP for details on using UUCP to exchange mail and files with remote sites.

mail — Overview

Electronic mail system

The COHERENT system includes a full-featured, UNIX-style mail system. This system consists of commands with which your system can send, receive, and forward mail; and configuration files, with which you can describe potential recipients of mail, either on your system or other systems. This article describes the design of the COHERENT mail system, and introduces the commands and files that compose it.

The COHERENT mail system has three major components: the *user agent* (also called the *mailer*); the *routing agents* (the commands **smail** and **rmail**); and the *delivery agents* (the commands **lmail** and **uux**).

This structure may seem overly complex (you may ask why all of this functionality could not be bundled into one program); however, experience has shown that it is best to organize each set of functions within its own program. One advantage this gives you is that you can replace one part of the mail system with another, superior program, without disturbing the operation of the system as a whole. For example, you may wish to replace the mailer **mail** with another mailer, such as **elm**. Because the mailer is its program, you can replace **mail** with **elm** without affecting the delivery or routing agents at all. You may never need to modify how the routing or delivery agents work, but studying the overall structure of the mail system will help you to decide intelligently on whether to replace a part of the mail system, and will also help you diagnose any problems that may crop up.

The following describes each set of agents in turn.

The User Agent

The user agent (also called the *mailer*) presents to you the messages that have been delivered into your mailbox. It also collects messages from you and hands them to the routing agent for delivery. This is the program you invoke when you wish to read mail or send a mail message.

COHERENT comes with one mailer, called **mail**. When you invoke it without any arguments, it reads the contents of file **/usr/spool/mail/***user* (where *user* is your login identifier), breaks its contents into individual messages, and presents the messages to you one by one. (File **/usr/spool/mail/***user* is also called your "mailbox", because that is where the mail system deposits your messages.) You can read a mail message; then, by giving commands to **mail**, you can reply to the message if you wish, then copy it into a file for archiving or throw it away.

If you wish to send mail, type the command

```
mail user
```

where *user* identifies the user to whom you wish to send the message. *user* can either be a user on your system, or on a remote system; if the latter, you must also name the site on which *user* resides. The syntax for naming a remote sites is described in further detail below. When you invoke **mail** to send a mail message, **mail** presents the prompt **Subject:**, upon which you should type the subject of the message. When you have typed the subject, press `<¢>`; you can then type the body of the message. Press `<¢>` when you come to the end of a line; the cursor jumps to the beginning of the next line on your screen, and you can continue typing. When you have finished typing, type **<ctrl-D>** or a '.' at the beginning of a line; this signals **mail** that you have finished entering the message. **mail** then passes the message to the routing agents for delivery, as described in the next section.

If, while you are typing the body of your message, you type the letter '?' at the beginning of a line, **mail** invokes an editor and copies your message into it. The editor it invokes is the set named by the environmental variable **EDITOR**. You can then use the editor to finish typing your message. When you have finished typing your message, exit from the editor; the message will then dispatched.

When the mail dispatches your message, it checks your home directory for a file named **.signature**. This is your *signature* file: the mailer appends the contents of this file onto the end of your message, as your signature. A signature can be any mass of text that you wish; usually, it gives a user's name and e-mail address, and sometimes includes a joke, motto, or slogan as a form of self-expression. It's generally considered bad form to have a signature that exceeds five lines of text, or that uses vulgar, obscene, or abusive language.

To mail a file to another user, use the shell's redirection operator '<'. For example, the command

```
mail stephen < bug.report
```

mails file **bug.report** to user **stephen**. The file will be prefixed with your address, and suffixed with your mail "signature", should you have one.

For details on how to use the mailer and its commands, see the Lexicon entry for the command **mail**.

Other mailers are also available for COHERENT; the most popular one is named **elm**. This mailer uses a visual interface to display the messages that are in your mailbox; you can use the arrow keys on your keyboard to move a cursor and select the message you want. Sources and binaries for **elm** are available on the MWC BBS and on other sites on the Internet.

Routing Agents

The routing agent, as term implies, figures out how to deliver a message to its destination, and dispatches it appropriately.

The routing agent **rmail** ("route mail") receives mail from another system. If the mail is intended for your local system, **rmail** passes the mail to the delivery agent **lmail** (described below) for delivery on your system. If, however, the mail is intended for forwarding to another system, **rmail** forwards it appropriately. **rmail** does most of its work in the background; you will seldom if ever will need to work with it directly.

The routing agent **smail** ("send mail") receives mail from you and dispatches to its target user, either on your system or on another system. **smail** actually is a large, complex program that handles mail correctly under a great variety of conditions. Under COHERENT, **rmail** actually is a link to **smail**.

When you mail a message to *user*, **smail** performs the following steps to route the message:

- **smail** first looks up *user* in the file **/usr/lib/mail/aliases**. For details on aliases, see the Lexicon entry **aliases**.
- If **smail** finds *user* in one of these files, it substitutes the alias for *user*. If the alias is of the form

    sys`!`*user*

or

    sys`! ... !`*user*

or

    user@sys[.domain]

    smail treats it as a remote destination, and invokes command **uux** to spool the message to *sys*. When **uux** has delivered the message, it becomes the responsibility of command **uuxqt** on *sys* to pass the message to *user*.

- If **smail** finds no match in **/usr/lib/mail/aliases**, it looks up *user* in the file **/etc/passwd**, to see if she is a user on your local system. If **smail** does not find *user* in **/etc/passwd**, it throws away the message and mails an error message to the sender.
- If **smail** does find *user* in **/etc/passwd**, it then looks for file **.forward** in *user*'s home directory. This file normally contains a list of addresses to which mail is to be forwarded.
- If *user*'s home directory does not contain a file named **.forward**, **smail** passes the message to **lmail** which writes the message into the file **/usr/spool/mail/***user* (the user's "mailbox").
- If, however, *user* does have file **.forward** in her home directory, **smail** reads it, and forwards the message to the address or addresses that that file contains.

For further information on **smail**, and on its suite of configuration files, see the Lexicon entry for **smail**.

Before you can send mail to a remote site, you must have set up a UUCP connect to that site, either directly or indirectly. That is, you must have set up UUCP to send mail to that site, or to a system that can forward the mail to some other that may have permission to access the site in question. See the tutorial and Lexicon articles on UUCP for details on using UUCP to exchange mail and files with remote sites.

Please note that the routing agents are the only components of the mail system that must run **setuid** to assume the privilege of the superuser **root**.

Delivery Agents

The delivery agents actually move messages to their destination.

The delivery agent **lmail** ("local mail") places messages into users' mailboxes. To discourage the forging of mail, **lmail** does not use **setuid**. It must be run by a privileged user (generally **root**) so that it will have permission to write mail into every user's mailbox. As a rule, **lmail** is invoked only by the routing agent; you seldom, if ever, will work directly with **lmail**.

The UUCP **uux** queues commands for execution on a remote system. The mail system uses **uux** as a delivery agent; in fact, on most systems, the delivery of mail is **uux**'s principal task. When a message is to be forwarded to a remote site, **smail** invokes **uux** to create two files in directory **/usr/spool/uucp/***site* (where *site* names the site to which mail is being sent). One file, which has the prefix 'C', contains the **rmail** command to be executed on the remote site; the other file, which has the prefix 'D', contains the body of the message that **rmail** is to route. The next time your system polls *site* (or is polled by *site*), those files are copied to *site*, where they are executed by *site*'s copy of the command **uuxqt**.

You can use **uux** directly to spool commands for execution. For details, see the Lexicon entries for **uux** and **uuxqt**.

uux uses **setuid** to assume the identity of user **uucp** in order to write into the necessary spool directories. Please note that it is very easy to use **uux** to forge messages to remote systems. Keep this in mind if you plan to use electronic mail for any kind of authorization system.

Setting Up a Mail Feed

One of the most useful tasks a personal computer can perform for you is let you exchange electronic mail with users on remote systems. The following describes how you can set up your mail system so you can plug into the Internet and begin to exchange mail with the outside world.

To begin, the COHERENT system at present can exchange mail with remote systems only via UUCP. To receive mail, you must find a site that has a connection to the Internet — either direct or indirect — and is willing to act as a UUCP feeder for you. Such a site may be a local college or university, or a commercial "Internet provider."

Once you have located such a site, set up a UUCP connection with that site, as described in this manual's tutorial on UUCP. If you do not have experience in setting up a UUCP site, read this tutorial carefully, as this can be rather tricky.

Next, edit file **/etc/domain**, and set your system's domain to the name of the system from which you will be receiving your feed. For example, if you have purchased Internet service from site **acme.com**, then **/etc/domain** should read:

```
acme.com
```

Finally, you must edit file **/usr/lib/mail/config** to tell **smail** to forward to the feeder system all messages that are bound for the outside world. You must change two attributes within this file:

domains
This attribute sets the domain name that **smail** writes into each mail message's header. This is required so that other users can reply to your messages. Set it to the name of your feeder system; it should be identical to the name you wrote into **/etc/domain**. For example, if you are purchasing your Internet service from system **acme.com**, then set the attribute **domains** to:

```
domains=acme.com
```

smart_path
Set this to the name of the remote site as you have set it in file **/usr/lib/uucp/sys**. For example, if you are purchasing Internet service from site **acme.com**, you may have used the name **acme** to name the site within file **/usr/lib/uucp/sys**. In this instance, you set the **smail** attribute **smart_path** as follows:

```
smart_path=acme
```

That's all there is to it. **smail**'s default configurations will handle the rest. Once you have the UUCP connection working properly, then any mail to a user who is not on your local system will be forwarded to the system that is providing your mail feed, and from there forwarded to the remote site to which you addressed it.

For details on setting up a UUCP feed, see the UUCP tutorial that appears earlier in this manual; also see the Lexicon entries for **UUCP**, **sys**, **dial**, and **port**. For more information on modifying **smail**'s configuration file, see the Lexicon entry for **config**.

Mailing to Networks

The following gives directions on how to send mail to users on popular networks:

America Online
Send mail to *user***@aol.com**.
Applelink
Send mail to *user***@applelink.apple.com**.
ATTMail
Send mail to *user***@attmail.com**.
BITNET
Send mail to *user*@*host***.bitnet** or to *user***%***host***.bitnet@***gateway*.
Compuserve
Send mail to *number***.***number***@compuserve.com**. Note that Compuserve addresses are usually given as *number***,***number*; you must convert the comma to a period.

FidoNet
This network uses an unusual addressing scheme. To send mail to John Doe at 1:123/456.0, use the following domain address:

```
f456.n123.z1.fidonet.org.
```

The **z1** comes from the **1:** at the front of the FidoNode address. Then, put the person's name in front of this, with the at-sign between them:

```
john.doe@f456.n123.z1.fidonet.org
```

If the host label does not end in **.0**, as in 1:123/456.4, use that digit with **p** prefixed to it, as follows:

```
john.doe@p4.f456.n123.z1.fidonet.org
```

MCIMail
Send mail to *user***@mcimail.com**. **MCIMail** usually includes a hyphen in *user*'s name; be sure to remove it.
UUnet
Send mail to *user***@host.uucp**, or to *user***%***host***.uucp@***gateway*, or to *user*@*domain*.

These directions assume that you have a UUCP link to another system that gives you access to the Internet or other intelligent network. For more information on sending mail to remote systems via UUCP, see the Lexicon entry for **UUCP**.

Files

$HOME/.forward — Forwarding instructions for inbound mail
$HOME/.signature — Personal signature
$HOME/dead.letter — Message that **mail** could not send
/etc/domain — Name of your system's domain
/etc/passwd — User identities

/etc/uucpname — Name of your system
/tmp/mail* — Temporary and lock files
/usr/lib/mail/aliases — Aliases of users
/usr/lib/mail/config — **smail** configuration file
/usr/lib/mail/fullnames — Short full name aliases of users
/usr/lib/mail/paths — Mail routing control file
/usr/lib/mail/routers — Information on routing mail to remote sites
/usr/lib/mail/transports — Information on mail-transportation programs
/usr/spool/mail — Mailbox directory, filed by user name

See Also

aliases, commands, config, cvmail, .forward, mail, mkfnames, msg, nptx, paths, rmail, smail, UUCP
Krol, E.: *The Whole Internet: User's Guide & Catalog.* Sebastopol, Ca.: O'Reilly & Associates, Inc., 1992. *Highly recommended.*

mailq — Command

Display information about spooled mail
mailq [-v]

Command **mailq** displays information about the mail currently spooled in directory **/usr/spool/mail** and its subdirectories. Command-line option **-v** tells **mailq** to display a per-message transaction log for each message, which shows what has happened to the message so far.

See Also

commands, mail [overview], smail

Notes

mailq is a link to command **smail**.

main() — C Language

Introduce program's main function

A C program consists of a set of functions, one of which must be called **main()**. This function is called from the runtime startup routine after the runtime environment has been initialized.

Programs can terminate in one of two ways. The easiest is simply to have the **main()** routine **return()**. Control returns to the runtime startup; it closes all open file streams and otherwise cleans up, and then returns control to the operating system, passing it the value returned by **main()** as exit status.

In some situations (errors, for example), it may be necessary to stop a program, and you may not want to return to **main()**. Here, you can use the library function **exit()**; it cleans up the debris left by the broken program and returns control directly to the operating system.

The system call **_exit()** quickly returns control to the operating system without performing any cleanup. This routine should be used with care, because bypassing the cleanup will leave files open and buffers of data in memory.

Programs compiled by COHERENT return to the program that called them; if they return from **main()** with a value or call **exit()** with a value (e.g., **EXIT_SUCCESS** or **EXIT_FAILURE**), **main()** returns that value to the program that invoked it (e.g., the shell). Programs that invoke other programs through the function **system()** check the returned value to see if these secondary programs terminated successfully. If you exit from **main()** without explicitly returning a value (e.g., by just letting **main()** simply conclude, or by invoking **exit()** without a return status, or by invoking **return** without a return value), **main()** returns whatever random value happens to have been in the register EAX.

See Also

_exit(), argc, argv, C language, envp, exit(), EXIT_FAILURE, EXIT_SUCCESS
ANSI Standard, §5.1.2.2.1
POSIX Standard, §3.1.2.2

major number — Definition

Device numbering

A *major number* specifies the device driver associated with a given device name found in the directory **/dev**. COHERENT uses a device's the major number as an index into an internal table of device-driver pointers.

Every COHERENT device has a device number associated with it. This device number is of type **dev_t**, as defined in **<sys/types.h>**. The macro **major()** in **<sys/stat.h>** extracts the major number from a given device number.

See Also

device drivers, minor number, stat.h

make — Command

Program-building discipline
make [*option …***] [***argument …***] [***target …***]**

make helps you build things that consist of more than one file of source code. A "thing" can be a program, a report, a document, or anything else that is constructed out of something else.

Complex programs often consist of several *object modules*, each of which is the product of compiling a *source file*. A source file may refer to one or more **include** files, which can also be changed. Some programs may be generated from specifications given to program generators, such as **yacc**. Recompiling and relinking complicated programs can be difficult and tedious.

make regenerates programs automatically. It follows a specification of the structure of the program that you write into a file called **makefile**. **make** also checks the date and time that COHERENT has recorded for each source file and its corresponding object module; to avoid unnecessary recompilation, **make** will recompile a source file only if it has been altered since its object module was last compiled.

Options

The following lists the options that can be passed to **make** on its command line.

-d (Debug) Give verbose printout of all decisions and information going into decisions.

-e Force macro definitions in environment to override those in the **makefile**.

-f *file* *file* contains the **make** specification. If this option does not appear, **make** uses the file **makefile**, which is sought first in the directories named in the **PATH** environmental variable, and then in the current directory. If *file* is '-', **make** uses the standard input; note, however, that the standard input can be used *only* if it is piped.

-i Ignore all errors from commands, and continue processing. To invoke this behavior for an individual action within a **makefile**, prefix the action with the '-' flag. By default, **make** exits if a command returns an error.

-k Continue to update other targets that do not depend upon the current target if a non-ignored error occurs while executing the commands to bring a target up to date.

-n Test only; suppress execution of commands. To override this behavior for an individual action within a **makefile**, prefix the action with the '+' flag.

-p Print all macro definitions and target descriptions.

-q Return a zero exit status if the targets are up to date. Do not execute any commands.

-r Do not use the built-in rules that describe dependencies.

-S Terminate **make** if an error occurs while executing the commands to bring a target up to date. This is true by default, and the opposite of **-k**.

-s Do not print command lines when executing them. Commands preceded by '@' are not printed, except under the **-n** option.

-t (Touch option) Force the dates of targets to be the current time, and bypass actual regeneration. However, if the target is out-of-date, **make** will still execute an individual action if that action is prefixed with the '+' flag.

The Makefile

A **makefile** consists of three types of instructions: *macro definitions*, *dependency definitions*, and *commands*.

A macro definition simply defines a macro for use throughout the **makefile**; for example, the macro definition

```
FILES=file1.o file2.o file3.o
```

Note the use of the equal sign '='.

A dependency definition names the object modules used to build the target program, and source files used to build each object module . It consists of the *target name*, or name of the program to be created, followed by a colon ':' and the names of the object modules that build it. For example, the statement

```
example:  $(FILES)
```

uses the macro **FILES** to name the object modules used to build the program **example**. Likewise, the dependency definition

```
file1.o:  file1.c macros.h
```

defines the object module **file1.o** as consisting of the source file **file1.c** and the header file **macros.h**.

Finally, a command line details an action that **make** must perform to build the target program. Each command line must begin with a space or tab character. For example, the command line

```
    cc -o example $(FILES)
```

gives the **cc** command needed to build the program **example**. The **cc** command lists the *object modules* to be used, *not* the source files.

Note that if you prefix an action with a hyphen '-', **make** will ignore errors in the action. If the action is prefixed by '@', it tells **make** to be silent about the action — that is, do not echo the command to the standard output. The '+' flag is described below.

Finally, you can embed comments within a **makefile**. **make** ignores a pound sign '#' and all text that follows it. COHERENT's implementation of **make** recognizes the presence of quotation marks, and and does not treat a '#' as a comment if it appears between apostrophes or quotation marks, or prefixed by a backslash. Many other versions of **make** do not permit this, including the one specified by POSIX.2: *caveat utilitor*.

make searches for **makefile** first in directories named in the environmental variable **PATH**, and then in the current directory.

make Without a Makefile

Beginning with release 4.2 of COHERENT, you can also invoke **make** to build an object for which no **makefile** exists. In this case, **make** uses its default suffix rules to identify the objects it should construct and how it should construct them. If, for example, you type

```
make foo
```

make will search the local directory for any file named **foo** that has any of the suffices that **make** recognizes by default. If the local directory contains a file named **foo.c**, **make** invokes **cc** to compile it; whereas if it contains a file named **foo.o**, it invokes the linker **ld** to link it.

Note that if no **makefile** exists, **make** by default creates an executable named after the C source file, just as the command **cc** does.

Dependencies

The **makefile** specifies which files depend upon other files, and how to recreate the dependent files. For example, if the target file **test** depends upon the object module **test.o**, the dependency is as follows:

```
test: test.o
      cc -o test test.o
```

make knows about common dependencies, e.g., that **.o** files depend upon **.c** files with the same base name. The target **.SUFFIXES** contains the suffixes that **make** recognizes. (Note that you can use the command **makedepend** to build such a list dynamically. For details, see its entry in the Lexicon.)

make also has a set of rules to regenerate dependent files. For example, for a source file with suffix **.c** and a dependent file with the suffix **.o**, the target **.c.o** gives the regeneration rule:

```
.c.o:
      cc -c $<
```

The **-c** option to the **cc** commands tells **cc** not to link or erase the compiled object module. **$<** is a macro that **make** defines; it stands for the name of the file that causes the current action. The default suffixes and rules are kept in the files **/usr/lib/makemacros** and **/usr/lib/makeactions**.

Macros

To simplify the writing of complex dependencies, **make** provides a *macro* facility. To define a macro, write

NAME **=** *string*

string is terminated by the end-of-line character, so it can contain blanks. To refer to the value of the macro, use a dollar sign '$' followed by the macro name enclosed in parentheses or braces, e.g.:

$(*NAME*)
${*NAME*}

If the macro name is one character, parentheses are not necessary. **make** uses macros in the definition of default rules:

```
.c.o:
      $(CC) $(CFLAGS) -c $<
```

where the macros are defined as

```
CC=cc
CFLAGS=-V
```

The built-in macros are as follows:

$* The target's name, minus a '.'-delimited suffix.

$@ For regular targets, the target's full name. For targets that are library dependencies of the form *library***(***object***)**, this macro expands to the *library* part of the target.

$% For targets that are library dependencies of the form *library***(***object***)**, this macro expands to the *object* part of the target.

$? This expands to prerequisite files that are newer than the target.

$< For suffix-rules, this macro expands to the name of the prerequisite file that **make** chose as the implicit prerequisite of the target. Do not use this macro outside a suffix rule.

You can specify macro definitions in the **makefile**, in the environment, or as a command-line argument. A macro defined as a command-line argument always overrides a definition of the same macro name in the environment or in the **makefile**. Normally, a definition in a **makefile** overrides a definition of the same macro name in the environment; however, with the -e option, a definition in the environment overrides a definition in the **makefile**.

Each command line *argument* should be a macro definition of the form

```
OBJECT=a.o b.o
```

Arguments that include spaces must be surrounded by quotation marks, because blanks are significant to the shell **sh**.

Source File Path

make first looks for the file with the name given, which may be relative to the current directory when make was invoked. If it does not find the file, and if the name of the file is not an absolute path name, **make** removes any leading path information from the name and looks for the file in the current directory. If the file is not found in the current directory, **make** then searches for the file in the list of directories specified by the macro **$(SRCPATH)**. This allows you to compile a program in an object directory separate from the source directory. For example

```
export SRCPATH=/usr/src/local/me
make
```

or alteratively

```
make SRCPATH=/usr/src/local/me
```

builds objects in the current directory as specified by the **makefile** and sources in **/usr/src/local/me**. To test changes to a program built from several source files, copy only the files you wish to change to the current directory; make will use the local sources and find the other sources on the **$(SRCPATH)**.

Note that **$(SRCPATH)** can be a single directory, as in the above example, or a ':'-separated list of directories, as described in the Lexicon entry for the function **path()**.

Macros and Environmental Variables

The environmental variable **MAKEFLAGS** provides an alternative method of passing parameters to **make**. If this variable is defined, **make** processes the switches that it contains as if they were specified on the command line. **make** processes **MAKEFLAGS** before it processes any actual command-line parameters.

Either of the following two formats can be used for **MAKEFLAGS**:

```
MAKEFLAGS="-n -d"
MAKEFLAGS=nd
```

Either of the above passes to **make** the options **n** and **d**.

After it processes the switches named in **MAKEFLAGS**, **make** processes all options set on its command line. **make** then re-defines **MAKEFLAGS** to contain the full set of switches passed to it, and marks the macro for export. This means that recursive invocations of **make** are passed the same switch settings as the highest-level invocation of **make**. (See also the description of the '+' flag, below.)

make takes all other environment-variable settings passed to it and uses them to set the values of corresponding macros internally.

When **make** executes a command, it exports to that command all the environmental variables **make** imported from the initial environment, the **MAKEFLAGS** environmental variable, and the macros defined on the **make** command line.

Always Actions

If an action for rebuilding a target begins with the '+' flag, **make** executes the action even if the command line specifies the option **-n**. This is useful when dealing with recursive **makefiles**: when you pass the options **-p**, **-d**, or **-n** to the top-level invocation of **make**, the top-level **makefile** can still invoke the sub-**makefiles**, and pass them the same flags via the environmental variable **MAKEFLAGS**, as described above. This simplifies the debugging of **makefile**s for complex projects. This flag mainly affects **make**'s usage with the options **-q**, **-n**, and **-t**.

Library Dependencies

make interprets targets of the form *library***(***object***)** as referring to members of an archive created with the archiver **ar**. **make** can examine the archive's contents to determine whether the named member is present and what date it possesses.

When building such a target, **make** looks for suffix rules for use in building *object*, but with a target suffix of **.a** rather than the actual suffix of *object*.

For example, with the default **make** rules in effect, the target

```
libc.a(clock.o)
```

would be rebuilt from a source file **clock.c** by the suffix-rule **.c.a**. The default suffix rule (as supplied in file **/usr/lib/makeactions**) deals with building the *object* file and then uses the macros **AR** and **ARFLAGS** to move the resulting object file into the target archive.

Actions for library targets use macro definitions that differ slightly from those for normal actions. When it builds a library target, **make** sets the macro **$@** to the name of the *library* part of the target, and sets the special macro **$%** (defined only for use with library targets) to the name of the *object* part of the target.

Single-Suffix Rules

make can use an inference rule of the form:

```
suffix:
        actions
```

to infer an action from a target that does not have a suffix. When you use a target that has no explicit rule and no known suffix, **make** appends onto the target every known suffix in turn, and for each suffix searches for a file or rule for building the target. If **make** discovers a file that matches one of file names that it has built, it then tries to use a single-suffix rule to generate the target from **target***suffix*, with the actions given in the single-suffix rule.

For example, the default rules for **make** contain a single-suffix rule:

```
.c:
        $(CC) $(CFLAGS) $@ $<
```

Given the above rule and a file in the current directory or source path named **clock.c**, the target

```
clock:
```

results in the executable file **clock** being built by compiling the single source file **clock.c** and linking it.

Suffix-Rewriting Macro Expansion

You can use a special form of macro expansion

$(*macro***:***suffix***[=***value***])**

to simplify the use of macros that involve long lists of files names. When you request the above form of expansion, **make** searches the expansion of *macro*; for every word that ends in *suffix* it replaces *suffix* with the optional *value*.

For example, consider the following:

```
SOURCES = parse.c interpret.c builtin.c
OBJS = $(SOURCES:.c=.o)
```

This expansion of the macro **OBJS** is:

```
parse.o interpret.o builtin.o
```

When a **makefile** uses long lists of files, this facility not only saves typing, but eases maintenance because you need to change only one list of files.

Path-Oriented Macro Expansions

The following special-macro expansion forms perform path processing on the macro's contents:

$(*special***)**	Normal expansion
$(*special***F)**	Expand only file-name part
$(*special***D)**	Expand only directory part without trailing slash

where *special* is one of the following: **@**, **?**, **<**, *****, or **%**. These expansion forms allow rules (especially inference rules) to deal easily with path-oriented operations, without resorting to complex shell operations involving backquoting and the command **basename**. In particular, when expanding a macro with a file list such as **${?D}**, **make** processes all the entries in the file list as specified; otherwise, this would be extremely cumbersome.

Files

makefile
Makefile — List of dependencies and commands
/usr/lib/makeactions — Default actions
/usr/lib/makemacros — Default macros

See Also

as, cc, commands, ld, makedepend, srcpath, touch
The make Programming Discipline, tutorial

Diagnostics

make returns the following error messages:

; after target or macroname *(error)*
A semicolon appeared after a target name or a macro name.

Bad macro name *(error)*
A macro includes an illegal character, e.g., a control character.

= in or after dependency *(error)*
An equal sign '**=**' appeared within or followed the definition of a macro name or target file. For example, **OBJ=atod.o=factor.o** will produce this error.

Incomplete line at end of file *(error)*
An incomplete line appeared at the end of the **makefile**.

Macro definition too long *(error)*
The macro definition exceeds the limited designed into the preprocessor.

Multiple actions for *name (error)*
A target is defined with more than one single-colon target line.

Multiple detailed actions for *name (error)*
A target is defined with more than one single-colon target line.

Must use "::" for *name (error)*
A double-colon target line was followed by a single-colon target line.

Newline after target or macroname *(error)*
A newline character appears after a target name or a macro name.

'::' not allowed for *name* (error)
A double-colon target line was used illegally; for example, after single-colon target line.

::: or : in or after dependency list *(error)*
A triple colon is meaningless to **make**, and therefore illegal wherever it appears. A single colon may be used only in a target line (which is also called the *dependency list*), and nowhere else.

Out of core (adddep) *(error)*
This results from a system problem. Try reducing the size of your **makefile**.

Out of range number input. *(warning)*
You attempted to use a numeric value that is out of range.

Out of space *(error)*
System problem. Try reducing the size of your **makefile**.

Out of space (lookup) *(error)*
System problem. Try reducing the size of your **makefile**.

Syntax error *(error)*
The syntax of a line is faulty.

Too many macro definitions *(error)*
The number of macros you have created exceeds the capacity of your computer to process them.

= without macro name or in token list *(error)*
An equal sign '**=**' can be used only to define a macro, using the following syntax: "MACRO=*definition*". An incomplete macro definition, or the appearance of an equal sign outside the context of a macro definition, will trigger this error message.

: without preceding target *(error)*
A colon appeared without a target file name, e.g., :*string*.

Notes

Prior to release 4.2, COHERENT's implementation of **make** permitted users to use the macro **$<** outside of suffix rules. This non-standard behavior is no longer supported.

The order of items in **makemacros/.SUFFIXES** is significant. The consequent of a default rule (e.g., **.o**) must *precede* the antecedent (e.g., **.c**) in the entry **.SUFFIXES**. Otherwise, **make** will not work properly.

makeboot — Command

Create a bootable floppy disk
makeboot

The script **makeboot** automatically builds a bootable floppy disk. To use it, insert a scratch floppy disk into your system's drive 0 (drive A:), log in as the superuser **root**, and then type **makeboot**.

makeboot automatically formats the floppy disk, builds a file system on it, and copies onto the disk all files it needs to boot COHERENT. To abort **makeboot** at any time, press **<ctrl-C>**.

Because **makeboot** is a script you can — and should — edit it to suit your preferences. For example, by default

makeboot copies both the editors **vi** and **me** onto the floppy disk; you may wish to save space by copying just one or the other.

See Also

commands, floppy disk

Notes

makeboot reads file **/bin/mount** to discover whether floppy-disk drive 0 is 3.5 inches or 5.25 inches. This file was initialized when you last installed or updated COHERENT. If you have changed your A drive since then, **makeboot** may think it is working with one size of floppy disk when in fact it is working with another. To correct this error, abort **makeboot**, then edit **/etc/mount** so that it reflects your system's true configuration of disk drives.

makedepend — X Utility

Generate list of dependencies for a makefile
makedepend [-D*name***=***def***] [-D***name***] [-I***includedir***] [-Y***includedir***] [-a] [-f***makefile***] [-o***objsuffix***] [-p***objprefix***] [-s***string***] [-w***width***] [-v] [-m] [--***otheroption ...***--]** *sourcefile ...*

makedepend reads each *sourcefile*, and parses it as the C preprocessor does. It processes every **#include**, **#define**, **#undef**, **#ifdef**, **#ifndef**, **#endif**, **#if**, and **#else** directives so that it can correctly tell which **#include** directives should be used in a compilation. Any **#include** directive can reference a file that has other **#include** directives, and **makedepend** parses these files as well.

Every file that a *sourcefile* includes, directly or indirectly, is what **makedepend** calls a *dependency*. It writes these dependencies into a **makefile** in such a way that **make** will know which object files must be recompiled when a dependency has changed.

By default, **makedepend** writes its output into a file named **makefile**, if it exists; otherwise, it writes its output into **Makefile**. You can specify an alternate makefile with the option **-f**. **makedepend** first searches the makefile for the line

```
# DO NOT DELETE THIS LINE -- make depend depends on it.
```

or one provided with the option **-s** as a delimiter for the dependency output. If it finds the line, it deletes everything after after this line to the end of the makefile, and writes its output after this line. If **makedepend** does not find this line, it appends the string to the end of the makefile and writes the output after that. For each *sourcefile*, **makedepend** writes into the makefile a line of the form

```
sourcefile.o: dfile ...
```

where **sourcefile.o** is the name of the *sourcefile* with its suffix replaced **.o**, and **dfile** is a dependency that **makedepend** discovered in a **#include** directive as it parsed *sourcefile* or one of the files it includes.

Command-line Options

makedepend ignores any option it does not understand, so you can use the same arguments that you would for **cc**. It does recognize the following command-line options:

-D*symbol***[=***def***]**
Define *symbol* within **makedepend**'s internal symbol table. Without **=***def*, **makedepend** defines it as **1**.

-I*includedir*
Tell **makedepend** to prefix *includedir* onto the list of directories to search when it encounters a **#include** directive. By default, **makedepend** only searches only **/usr/include**.

-Y[*includedir***]**
Search *includedir* for header files instead of all of the standard header-file directories. If you omit to name an *includedir*, this option prevents searching of the standard header-file directories.

-a Append the dependencies to the end of the file instead of replacing them.

-f*file* Write output into *file* instead of into **makefile**.

-o*objsuffix*
Append *objectsuffix* to a *filename* instead of the default **.o**.

-p*objprefix*
Prefix the name of each object file with *objprefix*. This usually is used to designate a different directory for the object file. The default is the empty string.

-s*string* Use *string* as the starting-string delimiter within a **makefile**.

-w*width*
Set the width of a line of output to *width*. By default, **makedepend** limits a line of output to 78 characters.

-v Verbose: tell **makedepend** to write onto the standard output the list of files that each input file includes.

-m Warn about multiple inclusion. This option tells **makedepend** to warn if any input file includes another file more than once. In previous versions of **makedepend**, this was the default behavior; the default has been changed to better match the behavior of the C compiler, which does not consider multiple inclusion to be an error. This option is provided for backward compatibility, and to aid in debugging problems related to multiple inclusion.

-- option ... --
makedepend ignores every *option* that it does not recognize and that is bracked by two hyphens '--'. In this way, you can safely tell **makedepend** to ignore esoteric compiler arguments that might normally be found in a **CFLAGS** macro. **makedepend** processes normally all options between the pair of double hyphens that recognizes.

Algorithm

To speed its performance, **makedepend** makes two assumptions about the programs whose dependency it is mapping: first, that all files compiled by a single **makefile** will be compiled with roughly the same **-I** and **-D** options; and second, that most files in a directory include largely the same files. Given these assumptions, **makedepend** expects to be called once for each makefile, with all source files that that **makefile** maintains appearing on its command line.

makedepend parses each source and header file exactly once, and maintains an internal symbol table for each. Thus, the first file on the command line will take an amount of time proportional to the amount of time that a normal C preprocessor takes. However, on subsequent files, if **makedepend** encounters a header file that it has already parsed, it does not parse it again.

For example, imagine you are compiling two files, **file1.c** and **file2.c**. Assume, further, that each includes the header file **header.h**, and **header.h** in turn includes the files **def1.h** and **def2.h**. When you run the command

```
makedepend file1.c file2.c
```

makedepend parses **file1.c** and, therefore, **header.h** followed by **def1.h** and **def2.h**. It then decides that the dependencies for this file are

```
file1.o: header.h def1.h def2.h
```

When **makedepend** parses **file2.c** and discovers that it, too, includes **header.h**, it does not re-parse that file, but simply adds **header.h**, **def1.h**, and **def2.h** to the list of dependencies for **file2.o**.

Example

makedepend normally is used within a **makefile** target, so that typing the command

```
make depend
```

brings the dependencies up to date for the **makefile**. For example,

```
SRCS = file1.c file2.c ...
CFLAGS = -O -DHACK -I../foobar -xyz
depend:
        makedepend -- $(CFLAGS) -- $(SRCS)
```

See Also

cc, commands, make
X Windows Manual: **imake**

Notes

makedepend was written by Todd Brunhoff of Tektronix, Inc., and MIT Project Athena.

malloc() — General Function (libc)

Allocate dynamic memory
#include <stdlib.h>
char *malloc(*size***) unsigned** *size***;**

malloc() helps to manage a program's free-space arenas. It uses a circular, first-fit algorithm to select an unused block of at least *size* bytes, marks the portion it uses, and returns a pointer to it. The function **free()** returns allocated memory to the free memory pool.

Each arena allocated by **malloc()** is rounded up to the nearest even number and preceded by an **unsigned int** that contains the true length. Thus, if you ask for three bytes you get four, and the **unsigned** that precedes the newly allocated arena is set to four.

When an arena is freed, its low order bit is turned on; consolidation occurs when **malloc()** passes over an arena as it searches for space. The end of each arena contains a block with a length of zero, followed by a pointer to the next arena. Arenas point in a circle.

The most common problem with **malloc()** occurs when a program modifies more space than it allocates with **malloc()**. This can cause later **malloc()**s to crash with a message that indicates that the arena has been corrupted. You can use the function **memok()** to isolate these problems.

Example

This example reads from the standard input up to *NITEMS* items, each of which is up to *MAXLEN* long, sorts them, and writes the sorted list onto the standard output. It demonstrates the functions **qsort()**, **malloc()**, **free()**, **exit()**, and **strcmp()**.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NITEMS 512
#define MAXLEN 256
char *data[NITEMS];
char string[MAXLEN];

main()
{
      register char **cpp;
      register int count;
      extern int compare();

      for (cpp = &data[0]; cpp < &data[NITEMS]; cpp++) {
            if (gets(string) == NULL)
                  break;
            if ((*cpp = malloc(strlen(string) + 1)) == NULL)
                  exit(1);
            strcpy(*cpp, string);
      }

      count = cpp - &data[0];
      qsort(data, count, sizeof(char *), compare);

      for (cpp = &data[0]; cpp < &data[count]; cpp++) {
            printf("%s\n", *cpp);
            free(*cpp);
      }
      exit(0);
}

compare(p1, p2)
register char **p1, **p2;
{
      extern int strcmp();
      return(strcmp(*p1, *p2));
}
```

See Also

alloca(), arena, calloc(), free(), libc, memok(), realloc(), setbuf(), stdlib.h
ANSI Standard, §7.10.3.3

POSIX Standard, §8.1

Diagnostics

malloc() returns NULL if insufficient memory is available.

Notes

The function **alloca()** allocates space on the stack. The space so allocated does not need to be freed when the function that allocated the space exits.

malloc.h — Header File

Definitions for memory-allocation functions
#include <sys/malloc.h>

malloc.h defines constants, structures, and macros used with COHERENT's memory-allocation functions. Note that this header does not declare the library's memory-allocation functions.

See Also

header files

Notes

This header file is obsolete, and will be dropped from a future release of COHERENT. Its use is strongly discouraged.

man — Technical Information

Manual macro package
nroff -man *file ...*

The macro package **tmac.an** formats UNIX-style manual pages. To invoke this package, use the argument **-man** with either **nroff** or **troff**

tmac.an includes the following macros:

.B [*string ...***]**
Use **boldface** font. If used with one or more *string*s, prints them in boldface. Otherwise, print all subsequent text in boldface, up to the next explicit change of font.

.BI boldtext *italictext* **boldtext** *italictext ...*
This macro prints its arguments in alternating **boldface** and *italic* fonts. It takes up to six arguments.

.BR boldtext romantext **boldtext** romantext ... **]**
This macro prints its arguments in alternating **boldface** and Roman fonts. It takes up to six arguments.

.CO Print the string "COHERENT".

.DE End a display. It is always used with the macro **.DS**, described below.

.DS Start a display. The text that follows, up to the macro **.DE**, is read into a diversion. It is not adjusted. When the display is closed, **nroff** checks whether the present page has enough space left to hold the text. If the page does not, **nroff** jumps to the next page and prints the text there.

.DT Set the default tab stops. **tmac.an** by default set a tab stop every five characters (half-inch).

.HE Help end — close a section of help messages.

.HP Hanging paragraph. The new paragraph is separated by a space from the text that came above it; however, unlike the macro **.PP**, the new paragraph keeps the current level of indentation.

.HS Help start. All text from here up to the macro **.HE** is assumed to form a special help message, and is ignored.

.I [*string ...***]**
Use **italic** font. If used with one or more *string*s, prints them in italic. Otherwise, print all subsequent text in italic, up to the next explicit change of font.

.IB *italictext* **boldtext** *italictext* **boldtext ...**
This macro prints its arguments in alternating *italic* and **boldface** fonts. It takes up to six arguments.

.IP [*string* **[***indentation***]]**
Indented paragraph. If it has no arguments, it drops a space and indents subsequent text to the current level of indentation. If the macro has one argument, it uses that argument as a stub, and indents the following text by another five characters (one-half inch). If it has two arguments, it uses the first as a stub, and indents the subsequent text by the value given in the second argument.

.IR *italictext* romantext *italictext* romantext ... **]**
This macro prints its arguments in alternating *italic* and Roman fonts. It takes up to six arguments.

.PD [*distance***]**
Set the default interparagraph distance to *distance*. If invoked without an argument, it resets the interparagraph distance to the default, which is kept in the number register **PD**.

.PP Paragraph. The macro inserts a space into the output, and indent subsequent text by the default amount, which is the value kept in the number register **IN**.

.R Use Roman font. If used with one or more *string*s, prints them in Roman. Otherwise, print all subsequent text in Roman, up to the next explicit change of font.

.RB romantext **boldtext** romantext **boldtext ...**
This macro prints its arguments in alternating Roman and **boldface** fonts. It takes up to six arguments.

.RE End relative indentation. Subsequent text is printed at the previous level of indentation.

.RI romantext *italictext* romantext *italictext ...*
This macro prints its arguments in alternating Roman and *italic* fonts. It takes up to six arguments.

.RS [*indentation***]**
Start relative indentation. The indentation of subsequent text is increased by *indentation*. If invoked without an argument, indentation is increased by the default amount, as set by the number register **IN**.

.SH [*text***]**
Section heading. Set *text* in bold as the title of the section. If it is invoked without an argument, this macro uses the first line of the subsequent text as the section's title. Subsequent text is indented by the default amount, as set by the number register **IN**.

.TH [*first second third fourth fifth***]**
Header. This is the first macro to appear in any manual page. Its optional arguments are used in the header and footer of the manual page, as follows:

first	The name of the manual page. It appears in the left and right corners of each page's header.
second	This argument gives the section of the UNIX manual that holds the manual page.
third	This argument appears in the center of each page's footer. It usually names the category of item that this manual page is documenting.
fourth	This appears in the lower-left corner of each page.
fifth	This appears in the center of each page's header.

.TP [*indentation***]**
Tagged paragraph. This macro resembles the macro **.IP**, except that it uses first line of subsequent text as the paragraph's stub.

tmac.an uses the following number registers to control its behavior. These are defined in the macro **.TH**; if you wish to reset them, do so *after* you have invoked macro **.TH**:

IN	The default indentation.
LL	The default line length.
PD	The default distance between paragraphs.

Finally, **tmac.an** sets the following strings:

R	The registered trademark symbol. This is equivalent to the special character **\(rg**.
Tm	The trademark symbol. This is equivalent to the special character **\(tm**.

Files

/usr/lib/tmac.an — Macro package

See Also

ms, nroff, troff, Using COHERENT
nroff, The Text Processing Language, tutorial

man — Command

Display Lexicon entries
man [-dw] [*page ...***]**

man prints each manual *page* onto the standard output. This normally is an entry from the COHERENT Lexicon, although it can be a manual page from any other source as well.

When used with the option **-w**, it prints the path name of the file instead of printing the document itself. When used with option **-d**, it dumps a list of all available manual pages to the standard output device, for your perusal.

By default, **man** uses the pager **more** to display text. To use another pager, e.g., **scat**, define the environmental variable **PAGER**:

```
	export PAGER="/bin/scat"
```

man normally searches for manual pages in the directory **/usr/man**. However, if the environmental variable **MANPATH** is set, **man** searches for manual pages in each directory that it names. **MANPATH** must name one or more directories, with directories separated by a colon ':'.

Index Files

To locate a manual page, **man** reads index files. It assumes that every file **/usr/man/*.index** is an index file; it then opens these files, and searches them for the manual entry you have requested.

Prior to release 4.2, an index file consisted of entries that had the format:

relative-path-name article-name

where *relative-path-name* gave the subdirectory and file in **/usr/man** that held the manual-page entry, *article_name* gave the name of the article as it appears in the Lexicon. Beginning with release 4.2, **man** uses index entries of the form:

relative-path-name article_name description

description gives a brief summary of the article. Fields must be separated by one more white-space characters. For example, entries

```
	COHERENT1/bc		bc		Interactive calculator with arbitrary precision
	LOCAL/chess		chess		Interactive chess program
```

associate manual-page file **/usr/man/COHERENT1/bc** with the Lexicon entry for the command **bc**. Likewise, rules for the user-written chess game **chess** are found in file **/usr/man/LOCAL/chess**.

man can read index entries prepared in either the "old" or the "new" form. We encourage you to use the new form, because this format also allows the index entries to be used by the command **apropos**.

Adding Manual-Page Entries

When writing new manual-page entries for COHERENT, we recommend that you place them into a subdirectory of **/usr/man**. This subdirectory should be uniquely named to avoid possible name-space collisions. A good rule of thumb is to name the subdirectory after the application with which it is associated. Also, when all manual-pages associated with a given application reside in a specific subdirectory, you can update the manual pages easily.

You should also add a uniquely named index file to directory **/usr/man** that identifies each of the newly added manual pages. This index file should use the "new" format described above; and its name should end with the suffix **.index**.

Files

/usr/man/* — Directories that hold manual pages **/usr/man/*.index** — Index files

See Also

apropos, commands, help, install, PAGER, Using COHERENT

Notes

The manual pages that are included with your release of the COHERENT system may include entries that have been corrected and updated since your COHERENT manual was printed. If there is a discrepancy between an on-line manual page and the printed COHERENT manual, you should assume that the on-line manual page is correct.

manifest constant — Overview

A **manifest constant** is a constant that is given a name so it can be defined differently under different computing environments. An example is **EOF**, the end-of-file marker, which has wildly different representations under different operating systems. Note, too, that numerals are manifest constants by definition.

The use of manifest constants in programs helps to ensure that code is portable by isolating the definition of these elements in a single header file, where they need to be changed only once.

The header file **limits.h** defines a set of macros that express certain numeric limits of COHERENT's implementation of C.

See Also

__DATE__, __FILE__, __LINE__, __STDC__, __TIME__, C preprocessor, EOF, EXIT_FAILURE, EXIT_SUCCESS, limits.h, macro, MB_CUR_MAX, NULL, RAND_MAX, portability, Programming COHERENT

math.h — Header File

Declare mathematics functions
#include <math.h>

math.h is the header file to be included with programs that use any of COHERENT's mathematics routines. It includes the following: definitions for mathematical functions; error return values, as used by the **errno** function; definitions of mathematical constants, e.g., **HUGE_VAL**; the definition of structure **cpx**, which describes complex variables; definitions of internal compiler functions; and, finally, prototypes of all mathematical functions.

See Also

header files, libm
ANSI Standard §7.5

MB_CUR_MAX — Manifest Constant

Largest size of a multibyte character in current locale
#include <stdlib.h>

MB_CUR_MAX is a manifest constant that is defined in the header **stdlib.h**. It expands into an expression that indicates the maximum number of bytes contained in a multibyte character in the current locale.

See Also

manifest constant
ANSI Standard, §7.10.7

mboot — Device Driver

Master boot block for hard disk

To be bootable, a COHERENT file system must contain a boot block (either **boot** or **mboot**). In addition, all hard disks must contain the master boot block **mboot** or an equivalent.

mboot is the master boot block for a hard-disk drive. It is compatible with, and therefore can replace, the IBM master boot block installed by the MS-DOS command **FDISK**. It must be installed in the first sector of the hard disk, as follows:

```
	/etc/fdisk -b /conf/mboot /dev/at0x
	/bin/sync
```

mboot searches its internal partition table (updated by the command **fdisk**) for an active partition. You can select an alternate partition by pressing 0 through 7 before the system selects the active partition. If the selected partition is of non-zero size with a valid partition boot block, COHERENT executes that partition's boot block. Otherwise, the prompt

```
	Select partition [0-7]
```

appear, and the system waits for you to select the partition you want.

Files

/conf/mboot — Hard-disk master boot block

See Also

boot, device drivers, fdisk, mkfs

mcd — Device Driver

Device driver for Mitsumi CD-ROM drives

mcd is a device driver for a Mitsumi CD-ROM drive, models FX001, FX001 high speed, FX001D, or LU005, that is plugged into the Mitsumi controller card. It has major number 16.

Normally, this device driver is included in the kernel when you install or update COHERENT. To configure this driver, log in as the superuser **root**, and execute script **/etc/conf/mcd/mkdev**. Then run the command

```
	/etc/conf/bin/idmkcoh -o coh.test
```

to build a test kernel that includes the driver.

Files

/dev/cdrom — Device applications read for CD-ROMs by default
/dev/rmcd0 — Character-special device for accessing Mitsumi CD-ROM

See Also

CD-ROM, device drivers, hai

mcmp() — Multiple-Precision Mathematics (libmp)

Compare multiple-precision integers
#include <mprec.h>
int mcmp(*a, b***)**
mint **a*, **b***;**

mcmp() compares the multiple-precision integers (or **mint**s) pointed to by *a* and *b*. It returns a signed integer less than, equal to, or greater than zero according to whether the value pointed to by *a* is less than, equal to, or greater than that pointed to by *b*.

See Also

libmp

mcopy() — Multiple-Precision Mathematics (libmp)

Copy a multiple-precision integer
#include <mprec.h>
void mcopy(*a, b***)**
mint **a*, **b***;**

mcopy() sets the multiple-precision integer (or **mint**) pointed to by *b* to the value pointed to by *a*.

See Also

libmp

mdevice — System Administration

Describe drivers that can be linked into kernel
/etc/conf/mdevice

File **mdevice** describes each device driver that can be linked into the COHERENT kernel. The command **idmkcoh** uses the information in this file when it builds and configures a new kernel.

mdevice contains one line for each driver or kernel component that can be linked into the kernel. Each line, in turn, consists of ten fields. The following describes the ten fields in order, from left to right:

1. *Name*
This field gives the name of the driver or component. Each name must uniquely identify the driver or component within the kernel. This field cannot be longer than eight characters.

2. *Function Flags*
This field holds a flag for each function (that is, entry point) within the driver or component. This field is used only by drivers or components that use the STREAMS or DDI/DKI interfaces; drivers that use the internal-kernel interface should place a hyphen '-' in this field. The legal flags are as follows:

o	Open
c	Close
r	Read
w	Write
i	Ioctl
s	Startup
x	Exit
I	Init
h	Halt
p	Poll — that is, **chpoll()**

3. *Miscellaneous Flags*
These flags give information about the device. They are set by most varieties of driver; the only exception is a STREAMS driver, for which only the flag **S** matters. The legal flags are as follows:

c	Character device
b	Block device
f	Driver conforms to the DDI/DKI
o	Driver has only one entry in **/etc/conf/sdevice**
r	Driver is required in all configurations of the kernel
S	STREAMS module; or STREAMS device when used with the 'c' flag
H	Device driver controls hardware
C	Driver uses interal-kernel (**CON**) interface

Note that the 'C' flag is unique to COHERENT, and cannot be used under other operating systems.

4. *Code Prefix*
This gives the "magic prefix" by which the kernel identifies the entry-point routines for this driver. In most instances, this is identical with the driver's name.

5. *Block Major-Device Number*
This gives the major-device number of this driver when it is accessed in block mode. In most instances, this and the following field are identical.

6. *Character Major-Device Number*
This gives the major-device number of this driver when it is accessed in character (raw) mode. In most instances, this and the preceding field are identical.

7. *Minor Device Numbers, Minimum*
This gives the smallest number that can be held by a minor-device number under this driver. Most drivers set this field to the lowest legal value, which is zero.

8. *Minor Device Numbers, Maximum*
This gives the largest number that can be held by a minor-device number under this driver. Most drivers set this field to the highest legal value, which is 255.

9. *DMA Channel*
This gives the DMA channel by which the device is accessed. Under COHERENT, this is always set to -1.

10. *CPU ID*
This gives the CPU that controls this driver, should the driver be running in a multiprocessor environment and be dedicated to a particular processor. Under COHERENT, this is always set to -1.

For an example of modifying this file, see the entry for **device drivers**.

Example

The following gives some example entries from **mdevice**:

```
1      2      3      4      5      6      7      8      9      10
###
# Example of an kernel components:  floating-point emulator and STREAMS
###
em87  -     -     em87  0     0     0     0     -1    -1
streams     I     -     streams     0     0     0     0-1 -1

###
# Example of a STREAMS driver:  note flags ‘c’ and ‘S’ both set in field 3
###
echo  -     cSf   echo  0     33    0     255   -1    -1

###
# Example DDI/DKI character driver:  Note that field 2 is initialized.
###
trace ocriI cfo   tr    0     34    0     255   -1    -1

###
# Example IK driver:  Note flag ‘C’ in field 3
###
at    -     CGHo  at    11    11    0     255   -1    -1
```

See Also

Administering COHERENT, device drivers, idmkcoh, mtune, sdevice, stune

mdiv() — Multiple-Precision Mathematics (libmp)

Divide multiple-precision integers
#include <mprec.h>
void mdiv(*a, b, q, r***)**
mint **a*, **b*, **q*, **r***;**

mdiv() divides the multiple-precision integer (or **mint**) pointed to by *a* with that pointed to by *b*. It writes the quotient and remainder into, respectively, *q* and *r*. *b* must not be zero. The results of the operation are defined by the following conditions:

1. $a = q * b + r$
2. The sign of *r* equals the sign of *q*
3. The absolute value of *r* is greater than the absolute value of *b*.

See Also

libmp

me — Command

MicroEMACS screen editor
me [-e *errorfile***] [-f** *bindfile***] [***textfile ...***]**

me is the command for MicroEMACS, the screen editor for COHERENT. With MicroEMACS, you can insert text, delete text, move text, search for a string and replace it, and perform many other editing tasks. MicroEMACS reads text from files and writes edited text to files; it can edit several files simultaneously, while displaying the contents of each file in its own screen window.

Screen Layout

Before you can use MicroEMACS, you must set the environmental variable **TERM** in your environment. If you do not set this variable explicitly in your **.profile** file, COHERENT sets it by default to **ansipc**. See the Lexicon entry **TERM** for details.

If the command **me** is used without arguments, MicroEMACS opens an empty buffer. If used with one or more file name arguments, MicroEMACS will open each of the files named, and display its contents in a window. If a file cannot be found, MicroEMACS will assume that you are creating it for the first time, and create an appropriately named buffer and file descriptor for it.

The last line of the screen is used to print messages and inquiries. The rest of the screen is portioned into one or more *windows* in which text is displayed. The last line of each window shows whether the text has been changed, the name of the buffer, and the name of the file associated with the window.

MicroEMACS notes its *current position*. It is important to remember that the current position is always to the *left* of the cursor, and lies *between* two letters, rather than at one letter or another. For example, if the cursor is positioned at the letter 'k' of the phrase "Mark Williams", then the current position lies *between* the letters 'r' and 'k'.

Commands and Text

The printable ASCII characters, from ' ' to '~', can be inserted at the current position. Control characters and escape sequences are recognized as *commands*, described below. A control character can be inserted into the text by prefixing it with **<ctrl-Q>** (that is, hold down the **<control>** key and type the letter 'Q').

There are two types of commands to remove text. *Delete* commands remove text and throw it away, whereas *kill* commands remove text but save it in the *kill buffer*. Successive kill commands append text to the previous kill buffer. Moving the cursor before you kill a line will empty the kill buffer, and write the line just killed into it.

Search commands prompt for a search string terminated by **<return>** and then search for it. Case sensitivity for searching can be toggled with the command **<esc>@**. Typing **<return>** instead of a search string tells MicroEMACS to use the previous search string.

Some commands manipulate words rather than characters. MicroEMACS defines a word as consisting of all alphabetic characters, plus '_' and '$'. Usually, a character command is a control character and the corresponding word command is an escape sequence. For example, **<ctrl-F>** moves forward one character and **<esc>F** moves forward one word.

MicroEMACS can handle blocks of text as well as individual characters, words, and lines. MicroEMACS defines a block of text as all the text that lies between the *mark* and the current position of the cursor. For example, typing **<ctrl-W>** kills all text from the mark to the current position of the cursor; this is useful when moving text from one file to another. When you invoke MicroEMACS, the mark is set at the beginning of the file; you can reset the mark to the cursor's current position by typing **<ctrl-@>**.

Using MicroEMACS with the Compiler

MicroEMACS can be invoked automatically by the compiler command **cc** to help you repair all errors that occur during compilation. The **-A** option to **cc** causes MicroEMACS to be invoked automatically when an error occurs. The compiler error messages are displayed in one window, the source code in the other, and the cursor is at the line on which the first error occurred. You can correct the errors one by one. To move to the next error in the list, type **<ctrl-X>>**; to move the previous error, type **<ctrl-X><**.

When have finished making corrections, exit from MicroEMACS by typing **<ctrl-Z>**, as usual; the compiler will automatically be re-invoked to re-compile the corrected source code. If more errors are found, MicroEMACS will be re-invoked with the new list of errors. This cycle will continue either until the file compiles without error, or until you break the cycle by typing **<ctrl-U> <ctrl-X> <ctrl-C>**.

The option **-e** to the **me** command allows you to invoke the error buffer by hand. For example, the commands

```
cc myprogram.c 2>errorfile
me -e errorfile myprogram.c
```

divert the compiler's error messages into **errorfile**, and then invokes MicroEMACS to let you correct them interactively.

The MicroEMACS Help Facility

MicroEMACS has a built-in help facility. With it, you can ask for information either for a word that you type in, or for a word over which the cursor is positioned. The MicroEMACS help file contains the bindings for all library functions and macros included with COHERENT.

For example, consider that you are preparing a C program and want more information about the function **fopen**. Type **<ctrl-X>?**. At the bottom of the screen will appear the prompt

```
Topic:
```

Type **fopen**. MicroEMACS will search its help file, find its entry for **fopen**, then open a window and print the following:

```
Open a stream for standard I/O
#include <stdio.h>
FILE *fopen (name, type) char *name, *type;
```

If you wish, you can kill the information in the help window and copy it into your program, to ensure that you prepare the function call correctly.

Consider, however, that you are checking a program written earlier, and you wish to check the call for a call to **fopen**. Simply move the cursor until it is positioned over one of the letters in **fopen**, then type **<esc>?**. MicroEMACS will open its help window, and show the same information it did above.

To erase the help window, type **<ctrl-X>1**.

Options

The following list gives the MicroEMACS commands. They are grouped by function, e.g., *Moving the cursor*. Some commands can take an *argument*, which specifies how often the command is to be executed. The default argument is 1. The command **<ctrl-U>** introduces an argument. By default, it sets the argument to four. Typing **<ctrl-U>** followed by a number sets the argument to that number. Typing **<ctrl-U>** followed by one or more **<ctrl-U>**s multiplies the argument by four.

Moving the Cursor

<ctrl-A>	Move to start of line.
<ctrl-B>	(Back) Move backward by characters.
<esc>B	Move backward by words.
<ctrl-E>	(End) Move to end of line.
<ctrl-F>	(Forward) Move forward by characters.
<esc>F	(Forward) Move forward by words.
<esc>G	Go to an absolute line number in a file. Same as **<ctrl-X>G**.
<ctrl-N>	(Next) Move to next line.
<ctrl-P>	(Previous) Move to previous line.
<ctrl-V>	Move forward by pages.
<esc>V	Move backward by pages.
<ctrl-X>=	Print the current position.
<ctrl-X>G	Go to an absolute line number in a file. Can be used with an argument; otherwise, it will prompt for a line number. Same as **<esc>G**.
<ctrl-X>[	Go to matching C delimiter. For example, if the cursor is positioned under the character ‘{’, then typing **<ctrl-X>[** moves the cursor to the next ‘}’. Likewise, if the cursor is positioned under the character }, then typing **<ctrl-X>[** moves the cursor to the first preceding ‘{’. MicroEMACS recognizes the delimiters [,], {, }, (,), /*, and */.
<ctrl-X>]	Toggle reverse-video display of matching C delimiters. For example, if reverse-video displaying is toggled on, then whenever the cursor is positioned under a ‘}’ MicroEMACS displays the first preceding ’{’ in reverse video (should it be on the screen). MicroEMACS recognizes the delimiters [,], {, }, (,), /*, and */.
<esc>!	Move the current line to the line within the window given by *argument*; the position is in lines from the top if positive, in lines from the bottom if negative, and the center of the window if zero.
<esc><	Move to the beginning of the current buffer.
<esc>>	Move to the end of the current buffer.

Killing and Deleting

<ctrl-D>	(Delete) Delete next character.
<esc>D	Kill the next word.

<ctrl-H> If no argument, delete previous character. Otherwise, kill *argument* previous characters.

<ctrl-K> (Kill) With no argument, kill from current position to end of line; if at the end, kill the newline. With argument set to one, kill from beginning of line to current position. Otherwise, kill *argument* lines forward (if positive) or backward (if negative).

<ctrl-W> Kill text from current position to mark.

<ctrl-X><ctrl-O>
Kill blank lines at current position.

<ctrl-Y> (Yank back) Copy the kill buffer into text at the current position; set current position to the end of the new text.

<esc><ctrl-H>
Kill the previous word.

<esc><DEL>
Kill the previous word.

<DEL> If no argument, delete the previous character. Otherwise, kill *argument* previous characters.

Windows

<ctrl-X>1 Display only the current window.

<ctrl-X>2 Split the current window into two windows. This command is usually followed by **<ctrl-X>B** or **<ctrl-X><ctrl-V>**.

<ctrl-X>N (Next) Move to next window.

<ctrl-X>P (Previous) Move to previous window.

<ctrl-X>Z Enlarge the current window by *argument* lines.

<ctrl-X><ctrl-N>
Move text in current window down by *argument* lines.

<ctrl-X><ctrl-P>
Move text in current window up by *argument* lines.

<ctrl-X><ctrl-Z>
Shrink current window by *argument* lines.

Buffers

<ctrl-X>B (Buffer) Prompt for a buffer name, and display the buffer in the current window.

<ctrl-X>K (Kill) Prompt for a buffer name and delete it.

<ctrl-X><ctrl-B>
Display a window showing the change flag, size, buffer name, and file name of each buffer.

<ctrl-X><ctrl-F>
(File name) Prompt for a file name for current buffer.

<ctrl-X><ctrl-R>
(Read) Prompt for a file name, delete current buffer, and read the file.

<ctrl-X><ctrl-V>
(Visit) Prompt for a file name and display the file in the current window.

Saving Text and Exiting

<ctrl-X><ctrl-C>
Exit without saving text.

<ctrl-X><ctrl-S>
(Save) Save current buffer to the associated file.

<ctrl-X><ctrl-W>
(Write) Prompt for a file name and write the current buffer to it.

<ctrl-Z> Save current buffer to associated file and exit.

Compilation Error Handling

<ctrl-X>> Move to next error.

<ctrl-X>< Move to previous error.

Search and Replace

<ctrl-R> (Reverse) Incremental search backward; a pattern is sought as each character is typed.

<esc>R (Reverse) Search toward the beginning of the file. Waits for entire pattern before search begins.

<ctrl-S> (Search) Incremental search forward; a pattern is sought as each character is typed.

<esc>S (Search) Search toward the end of the file. Waits for entire pattern before search begins.

<esc>% Search and replace. Prompt for two strings; then search for the first string and replace it with the second.

<esc>/ Search for next occurrence of a string entered with the **<esc>S** or **<esc>R** commands; this remembers whether the previous search had been forward or backward.

<esc>@ Toggle case sensitivity for searches. By default, searches are case insensitive.

Keyboard Macros

<ctrl-X>(Begin a macro definition. MicroEMACS collects everything typed until the next **<ctrl-X>)** for subsequent repeated execution. **<ctrl-G>** breaks the definition.

<ctrl-X>) End a macro definition.

<ctrl-X>E (Execute) Execute the keyboard macro.

<ctrl-X>M Bind a newly created keyboard macro to a given keystroke or set of keystrokes.

Flexible Key Bindings

<ctrl-X>R Replace one binding with another.

<ctrl-X>X Rebind the prefix (meta) keys, and the multiple-execution key **<ctrl-U>**.

<ctrl-X>S Prompt for a file name, and write all flexible keybindings and macros into it. This command also saves information about how you have configured MicroEMACS; for example, it notes whether you have turned on word-wrapping.

<ctrl-X>L Prompt for a file name, and read all flexible keybindings and macros from it.

<ctrl-X>I Rebind current macro to the initialization macro.

By default, MicroEMACS checks for the existence of file **$HOME/.emacs.rc** and executes it if found. The **-f** option lets you specify an alternate file of keybindings macros from the **me** command line. After loading the file, MicroEMACS then executes the initialization macro, if one exists. For example, to load the keybindings file **bindings** and edit file **textfile**, use the command:

```
me -f bindings textfile
```

Change Case of Text

<esc>C (Capitalize) Capitalize the next word.

<ctrl-X><ctrl-L>
(Lower) Convert all text from current position to mark into lower case.

<esc>L (Lower) Convert the next word to lower case.

<ctrl-X><ctrl-U>
(Upper) Convert all text from current position to mark into upper case.

<esc>U (Upper) Convert the next word to upper case.

White Space

<ctrl-I> Insert a tab. Default behavior is to move the cursor to the nearest 8's boundary; for example, if the cursor is in the 62nd column on the screen, pressing **<ctrl-I>** moves it to column 64.

When used with a positive argument, change the behavior of the tab key. For example, **<ctrl-U>4<ctrl-I>** commands MicroEMACS to insert enough spaces for a tab key to reach a four's boundary.

When used with a negative argument, change the behavior of the tab character. For example, **<ctrl-U>-4<ctrl-I>** says that a tab character on a file will take you to the nearest 4's boundary. Thus, if you have a file with tabs in it and you use '-4', the appearance of the file on the screen will change; but if you use '4' the appearance of the file on the screen will not change.

To change the default size of a tab, set the environmental variable **TABSIZE** to a value other than eight.

<ctrl-J> Insert a new line and indent to current level. This is often used in C programs to preserve the current level of indentation.

<ctrl-M> (Return) If the following line is not empty, insert a new line; if empty, move to next line.

<ctrl-O> Open a blank line; that is, insert newline after the current position.

<tab> With argument, set tab fields at every *argument* characters. An argument of zero restores the default of eight characters. Setting the tab to any character other than eight causes space characters to be set in your file instead of tab characters.

Send Commands to Operating System

<ctrl-C> Suspend MicroEMACS and execute a subshell. Typing **<ctrl-D>** returns you to MicroEMACS and allows you to resume editing.

<ctrl-X>! Prompt for a shell command and execute it.

These commands recognize the shell variable **SHELL** to determine the shell to which it should pass the command.

Setting the Mark

<ctrl-@> Set mark at current position.

<esc>. Set mark at current position.

Help Window

<ctrl-X>? Prompt for word for which information is needed.

<esc>? Search for word over which cursor is positioned.

<esc>2 Erase help window.

Miscellaneous

<ctrl-G> Abort a command.

<ctrl-L> Redraw the screen.

<ctrl-Q> (Quote) Insert the next character into text; used to insert control characters.

<esc>Q Quote a character by numeric value. When you type this command, MicroEMACS prompts you for a numeric value, in decimal. It then inserts into your text the character whose value you type. This command is useful when you wish to enter characters with the high bit set.

<ctrl-T> Transpose the characters before and after the current position.

<ctrl-U> Specify a numeric argument, as described above.

<ctrl-U><ctrl-X><ctrl-C>
Abort editing and re-compilation. Use this command to abort editing and return to COHERENT when you are using the **-A** option to the **cc** command.

<ctrl-X>H Use word-wrap on a region.

<ctrl-X>F Set word wrap to *argument* column. If argument is one, set word wrap to cursor's current position.

<ctrl-X><ctrl-X>
Mark the current position, then jump to the previous setting of the mark. This is useful when moving text from one place in a file to another.

Diagnostics

MicroEMACS prints error messages on the bottom line of the screen. It prints informational messages (enclosed in square brackets '[' and ']' to distinguish them from error messages) in the same place.

MicroEMACS manipulates text in memory rather than in a file. The file on disk is not changed until you save the edited text. MicroEMACS prints a warning and prompts you whenever a command would cause it to lose changed text.

See Also

commands, ed, elvis, ex, sed, TERM, vi

Notes

Because MicroEMACS keeps text in memory, it does not work for extremely large files. It prints an error message if a file is too large to edit. If this happens when you first invoke a file, you should exit from the editor immediately. Otherwise, your file on disk will be truncated. If this happens in the middle of an editing session, however, delete text until the message disappears, then save your file and exit. Due to the way MicroEMACS works, saving a file after this error message has appeared will take more time than usual.

MicroEMACS is based upon the public domain editor by David G. Conroy.

mem — Device Driver

Physical memory file

The special file **/dev/mem** permits a program to read and write to the physical memory of the host computer, just as it reads and writes into an ordinary file. The location where I/O will occur can be positioned to any valid byte address by a call to **lseek()**. Note that **ps** and related commands use **/dev/kmem**, which manipulates the kernel's data space.

Commands may examine or change addresses in physical memory. Addresses to use when changing the system itself normally are obtained from the system load module (**/coherent**) name list, so that they always reflect the currently running version of the system.

Files

/dev/mem

See Also

clock, cmos, core, device drivers, lseek, ps

Diagnostics

On an error, such as nonexistent memory location, **mem** returns -1.

memccpy() — String Function (libc)

Copy a region of memory up to a set character
#include <string.h>
char *memccpy(*dest***,** *src***,** *c***,** *n***)**
char ****dest*, ****src***; unsigned int** *c***,** *n***;**

memccpy() copies characters from *src* to *dest*, stopping when either it finds the first occurrence of character *c* or it has copied *n* characters. Unlike the routines **strcpy()** and **strncpy()**, **memcpy()** copies from one region to another. Therefore, it will not halt automatically when it encounters NUL.

memccpy() returns a pointer to the first location after character *c* in *dest*, or NULL if character *c* was not found.

See Also

libc, memcpy(), strcpy(), strncpy(), string.h

Notes

memccpy() is not part of the ANSI Standard. Use of this library routine may restrict portability.

If *dest* and *src* overlap, the behavior of **memccpy()** is undefined. *dest* should point to enough reserved memory to hold *n* bytes of data; otherwise, data corruption may result.

memchr() — String Function (libc)

Search a region of memory for a character
#include <string.h>
char *memchr(*region***,** *character***,** *n***)**
char ****region*; int** *character***; unsigned int** *n***;**

memchr() searches the first *n* characters in *region* for *character*. It returns the address of *character* if it is found, or NULL if it is not.

Unlike the string-search function **strchr()**, **memchr()** searches a region of memory. Therefore, it does not stop when it encounters a null character.

Example

The following example deals a random hand of cards from a standard deck of 52. The command line takes one argument, which indicates the size of the hand you want dealt. It uses an algorithm published by Bob Floyd in the September 1987 *Communications of the ACM*.

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define DECK 52

main(argc, argv)
int argc; char *argv[];
{
      char deck[DECK], *fp;
      int  deckp, n, j, t;

      if(argc != 2 ||
         52 < (n = atoi(argv[1])) ||
         1 > n) {
               printf("usage: memchr n # where 0 < n < 53\n");
            exit(EXIT_FAILURE);
      }

      /* exercise rand() to make it more random */
      srand((unsigned int)time(NULL));
      for(j = 0; j < 100; j++)
            rand();

      deckp = 0;
      /* Bob Floyd's algorithm */
      for(j = DECK - n; j < DECK; j++) {
            t = rand() % (j + 1);
            if((fp = memchr(deck, t, deckp)) != NULL)
                  *fp = (char)j;
            deck[deckp++] = (char)t;
      }

      for(t = j = 0; j < deckp; j++) {
            div_t card;
            card = div(deck[j], 13);
            t += printf("%c%c  ",
                  /* note useful string addressing */
                  "A23456789TJQK"[card.rem],
                  "HCDS"[card.quot]);

            if(t > 50) {
                  t = 0;
                  putchar('\n');
            }
      }

      putchar('\n');
      return(EXIT_SUCCESS);
}
```

See Also

libc, strchr(), string.h
ANSI Standard, §7.11.5.1

memcmp() — String Function (libc)

Compare two regions
#include <string.h>
int memcmp(*region1***,** *region2***,** *count***)**
char ****region1*; char ****region2***; unsigned int** *count***;**

memcmp() compares *region1* with *region2* character by character for *count* characters.

If every character in *region1* is identical to its corresponding character in *region2*, then **memcmp()** returns zero. If it finds that a character in *region1* has a numeric value greater than that of the corresponding character in *region2*, then it returns a number greater than zero. If it finds that a character in *region1* has a numeric value less than less that of the corresponding character in *region2*, then it returns a number less than zero.

For example, consider the following code:

```
      char region1[13], region2[13];
      strcpy(region1, "Hello, world");
      strcpy(region2, "Hello, World");
      memcmp(region1, region2, 12);
```

memcmp() scans through the two regions of memory, comparing **region1[0]** with **region2[0]**, and so on, until it finds two corresponding "slots" in the arrays whose contents differ. In the above example, this will occur when it compares **region1[7]** (which contains 'w') with **region2[7]** (which contains 'W'). It then compares the two letters to see which stands first in the character table used in this implementation, and returns the appropriate value.

memcmp() differs from the string comparison routine **strcmp()** in a number of ways. First, **memcmp()** compares regions of memory rather than strings; therefore, it does not stop when it encounters a NUL.

Also, you can use **memcmp()** to compare an **int** array with a **char** array, because **memcmp()** simply compares areas of data.

See Also

libc, strcmp(), string.h
ANSI Standard, §7.11.4.1

memcpy() — String Function (libc)

Copy one region of memory into another
#include <string.h>
char *memcpy(*region1***,** *region2***,** *n***)**
vaddr_t *region1***;**
vaddr_t *region2***;**
unsigned int *n***;**

memcpy() copies *n* characters from *region2* into *region1*. Unlike the routines **strcpy()** and **strncpy()**, **memcpy()** copies from one region to another. Therefore, it will not halt automatically when it encounters NUL.

memcpy() returns *region1*.

Example

The following example copies a structure and displays it.

```
#include <string.h>
#include <stdio.h>

struct stuff {
      int a, b, c;
} x, y;

main()
{
      x.a = 1;
      /* this would stop strcpy or strncpy. */
      x.b = 0;
      x.c = 3;

      /* y = x; would do the same */
      memcpy(&y, &x, sizeof(y));
      printf("a =%d, b =%d, c =%d\n", y.a, y.b, y.c);
      return(EXIT_SUCCESS);
}
```

See Also

libc, strcpy(), string.h
ANSI Standard, §7.11.2.1

Notes

If *region1* and *region2* overlap, the behavior of **memcpy()** is undefined. *region1* should point to enough reserved memory to hold *n* bytes of data; otherwise, code or data will be overwritten.

memmove() — String Function (libc)

Copy region of memory into area it overlaps
#include <string.h>
char *memmove(*region1*, *region2*, *count***)**
char **region1*, **char** **region2*, **unsigned int** *count*;

memmove() copies *count* characters from *region2* into *region1*. Unlike **memcpy()**, **memmove()** correctly copies the region pointed to by *region2* into that pointed by *region1* even if they overlap. To "correctly copy" means that the overlap does not propagate, not that the moved data stay intact. Unlike the string-copying routines **strcpy()** and **strncpy()**, **memmove()** continues to copy even if it encounters a NUL.

memmove() returns *region1*.

Example

The following example rotates a block of memory by one byte.

```
#include <string.h>
#include <stddef.h>
#include <stdio.h>

char *
rotate_left(region, len)
char *region; size_t len;
{
      char sav;

      sav = *region;
      /* with memcpy this might propagate the last char */
      memmove(region, region + 1, --len);
      region[len] = sav;
      return(region);
}
```

```
char nums[] = "0123456789";
main(void)
{
      printf(rotate_left(nums, strlen(nums)));
      return(EXIT_SUCCESS);
}
```

See Also

libc, string.h
ANSI Standard, §7.11.2.2

Notes

region1 should point to enough reserved memory to hold the contents of *region2*. Otherwise, code or data will be overwritten.

memok() — General Function (libc)

Test if the arena is corrupted
int
memok();

The library function **memok()** checks to see if the arena has been corrupted. It returns one if the arena is sound, and zero if it has been corrupted.

Example

The following example purposely corrupts the arena, to demonstrate **memok()**. Please note that this is not a recommended programming practice.

```
extern char *malloc();
main()
{
      char *p;

        p = malloc(2);                      /* get 2 bytes of memory */
        printf("Arena is %s\n", memok() ? "OK" : "bad");
        strcpy(p, "too long");              /* clobber memory */
        printf("Arena is %s\n", memok() ? "OK" : "bad");
}
```

See Also

arena, calloc(), libc, malloc(), realloc()

memset() — String Function (libc)

Fill an area with a character
#include <string.h>
char *memset(*buffer***,** *character***,** *n***)**
char ****buffer*; int** *character***; unsigned int** *n***;**

memset() fills the first *n* bytes of the area pointed to by *buffer* with copies of *character*. It casts *character* to an **unsigned char** before filling *buffer* with copies of it.

memset() returns the pointer *buffer*.

Example

The following example fills an area with 'X', and prints the result.

```
#include <stdio.h>
#include <string.h>
#define BUFSIZ 20

main()
{
      char buffer[BUFSIZ];
```

```
        /* fill buffer with 'X' */
        memset(buffer, 'X', BUFSIZ);

        /* append null to end of buffer */
        buffer[BUFSIZ-1] = '\0';

        /* print the result */
        printf("%s\n", buffer);
        return(EXIT_SUCCESS);
}
```

See Also

libc, string.h
ANSI Standard, §7.11.6.1

mesg — Command

Permit/deny messages from other users
mesg [ny]

Normally, a user can communicate with other users by using the commands **msg** and **write**.

In certain situations, it is useful to suppress messages from other users. Therefore, COHERENT supplies the command **mesg**, which, lets you permit or suppress messages from other users. The argument **y** allows messages, whereas argument **n** disallows messages. With no argument, **mesg** tells you whether you can receive messages (as **yes** or **no**) without changing the message state.

Files

/dev/*

See Also

commands, msg, write

Notes

The owner-execute mode bit of the user's **tty** indicates whether messages are allowed.

min() — Multiple-Precision Mathematics (libmp)

Read multiple-precision integer from stdin
#include <mprec.h>
void min(*a*)
mint ****a*;**

min() reads a multiple-precision integer (or **mint**) from the standard input and writes it at the address held by *a*. The base of the **mint** is indicated by the value held in the external variable **ibase**.

min() accepts leading blanks and an optional leading minus sign; the number is terminated by the first non-legal digit.

See Also

libmp

minit() — Multiple-Precision Mathematics (libmp)

Condition global or auto multiple-precision integer
#include <mprec.h>
void minit(*a*)
mint ****a*;**

minit() helps to create a multiple-precision integer (or **mint**). If a new **mint** is declared to be global or automatic, you must call **minit()** before using the variable. This prevents garbage values in the newly created **mint** structure from causing chaos. A **mint** conditioned by **minit()** has no value; however, it may be used to receive the result of an operation.

See Also

libmp

minor number — Definition

Device numbering

A *minor number* specifies the device or type of device to use. COHERENT uses the minor number of a given device in a driver-specific manner. For example, a hard-disk driver may use the minor number to select a disk drive and partition.

Every COHERENT device has a device number associated with it. It is of type **dev_t**, as defined in **<sys/types.h>**. The macro **minor()** in **<sys/stat.h>** extracts the minor number from a given device number.

See Also

device drivers, major number, stat.h

mintfr() — Multiple-Precision Mathematics (libmp)

Free a multiple-precision integer
#include <mprec.h>
void mintfr(*a***)**
mint **a***;**

mintfr() frees the memory used by a **mint**.

See Also

libmp

mitom() — Multiple-Precision Mathematics (libmp)

Reinitialize a multiple-precision integer
#include <mprec.h>
void mitom(*n*, *a***)**
mint **a***;* **int** *n***;**

mitom() reinitializes the existing multiple-precision integer (or **mint**) pointed to by *a* to *n.*

See Also

libmp

mkdbm — Command

Build a data base for smail
/usr/lib/mail/mkdbm [-d] [-f] [-n] [-o *output-file***] [-v] [-y] [***file ...***]**

The command **mkdbm** generates a data base for **smail**.

It forms the data key from the characters up to, but not including, a colon (':') or white-space character. The data after the colon or white-space character forms the value associated with the key. You can use **mkdbm** to produce data-base files that can then be read by a **smail** router **pathalias** or its director **alias-file**. By default, the router and director are configured to use the DBM file-access protocol. (For information on routers and directors, see the Lexicon entries for **routers** and **directors**.)

For some data bases, you can use **mkline** to form single-line records whose comments and extra white space are removed. The generated data base contains a single NUL character at the end of each key and value. It also generates a single record that contains a '@' as a key and value; it does so for compatibility with the Berkeley **sendmail** command's alias files.

mkdbm recognizes the following command-line options:

-d Suppress writing the extra '@' record.

-f Fold the key to lower case before storing it within the data base.

-n Suppress writing a NUL character at the end of each line. Please note that this option is incompatible with **smail**'s method of accessing the data-base file.

-o *output-file*
Write output into *output-file*. This option also sets the name for the data base. If you do not use this option, **mkdbm** names the output data base after its first *file* argument. If, in addition, the command line does not name an input *file*, **mkmf** names the output file **dbm**.

-v Write statistics to the standard output.

-y Create an output file that is compatible with the Sun Yellow Pages (YP) system. This obviates the need for keeping a copy of **sendmail** on your system to maintain a YP-alias data base.

If its command line does not name an input *file*, **mkdbm** reads the standard input. **mkdbm** also reads the standard input if a *file* is named '-'; in this way, it can mix data read from the standard input with material read from files.

Calling **mkdbm** with the arguments **-ynd** generates a data base that is compatible with regular YP data bases. Using just the argument **-y** generates a data base that is compatible with the YP **mail.alias** data base.

As it creates the data base, **mkdbm** builds temporary files in the same directory in which it eventually builds the output files. When it has completed its work, **mkdbm** removes all data-base files that have the target name, sleeps for one or two seconds, then moves the newly written temporary data-base files to the target names. This method of writing a data-base is not compatible with the locking method used by Berkeley command **sendmail**.

Example

As an example of the use of **mkdbm** consider a file named **paths**, which contains the routing information:

```
.COM sun!%s
Stargate.COM ames!cmcl2!uiucdcs!stargate!%s
ames ames!%s
.ATT.COM mtune!%s
mtune mtune!%s
```

Given this file, the command

```
mkdbm -f paths
```

produces a data base in the files **paths.pag** and **paths.dir** that contains the above entries but with the keys shifted into lower case. For example, one entry will contain the key **stargate.com** with an associated value of:

```
ames!cmcl2!uiucdcs!stargate!%s
```

Files

dbmXXXXXX.pag
dbmXXXXXX.dir
Temporary files, created in the same directory as the output files.

See Also

commands, libgdbm, mail [overview], mkline, pathalias, smail

Notes

Copyright © 1987, 1988 Ronald S. Karr and Landon Curt Noll. Copyright © 1992 Ronald S. Karr.

mkdbm is part of the **smail** package. For a full copyright statement, see file **COPYING**, which is included with source code to **smail**, or type **smail -bc** to see the distribution rights and restrictions associated with this software.

mkdir — Command

Create a directory
mkdir [-rp] *directory*

mkdir creates *directory*. Files or directories with the same name as *directory* must not already exist. **directory** will be empty except for the entries '.', the directory's link to itself, and '..', its link to its parent directory.

Option **-r** creates directories recursively. For example, the command

```
mkdir -r /foo/bar/baz
```

creates directory **foo** in **/**; then creates directory **bar** in the newly created directory **foo**; and finally creates directory **baz** in the newly created directory **bar**.

Option **-p** behaves exactly the same as **-r**. COHERENT includes it for use by scripts imported from other operating systems.

See Also

commands, mkdir(), rmdir

Diagnostics

mkdir fails and prints an error message if you do not have permission to write into directory in which you are attempting to create a new directory or if the directory in which you attempted to create a new directory does not exist.

mkdir() — System Call (libc)

Create a directory
#include <sys/types.h>
#include <sys/stat.h>
int mkdir(*path***,** *mode***)**
char ****path*;**
int *mode***;**

The COHERENT system call **mkdir()** creates the directory specified by *path* and gives it the file mode specified by *mode*. If *path* is relative (that is, it doesn't begin with a '/' character), **mkdir()** creates the directory relates to the current directory of the process that calls **mkdir()**. If *path* is absolute (i.e., begins with a '/'), **path** specifies a directory to be created relative to the root directory for this process. See Lexicon article **chroot()** for details. If *path* specifies more than one directory level, all parent names specified must exist, must be accessible by the calling process, and actually must be directories.

Argument *mode* is formed by logically OR'ing permissions constants found in header file **<sys/stat.h>**. These constants begin with **S_** and determine the permissions for the directory. See the Lexicon article **stat.h** for details.

If the directory is successfully created, **mkdir()** returns zero. If an error occurs, **mkdir()** returns -1 and sets **errno** to an appropriate value.

See Also

libc, mkdir, rmdir, rmdir(), stat.h
POSIX Standard, §5.4.1

mkfifo() — System Call (libc)

Create a FIFO
#include <sys/types.h>
#include <unistd.h>
int mkfifo(*path***,** *mode***)**
const char ****path*; mode_t** *mode***;**

mkfifo() calls **mknod()** to create a FIFO. *path* points to the full path name of the FIFO to create. *mode* gives the mode into which the FIFO is to be opened. **mkfifo()** ignores the bits in *mode* other than the file-permission bits. The file permission bits of *mode* are modified by the process's file-creation mask; for details, see the Lexicon entry for **umask()**.

mkfifo() sets the ownership of the file FIFO's to the process's effective user identifier, and sets the FIFO's group identifier to the process's effective group identifier.

If all goes well, **mkfifo()** returns zero. If an error occurs, it returns -1 and sets **errno** to an appropriate value.

See Also

libc, libsocket, named pipe, pipe(), unistd.h
POSIX Standard, §5.4.2

mkfnames — Command

Generate data base of user names
/usr/bin/mkfnames [*namefile ...***]**

The script **mkfnames** generates a data base of users' names and addresses. It reads the contents of *namefile*, which contains each user's names and her e-mail address; invokes the command **nptx** to generate permutations of the users' names; then sorts the output of **nptx** and writes the sorted output onto the standard output. If no *namefile* is named on the command line, **mkfnames** reads the file **/etc/passwd**, and parses its contents into the form required by command **nptx**.

mkfnames is usually used to generate the file **/usr/lib/mail/fullnames**, which the mail system uses to translate a person's name into her e-mail address. If more than one login account has the same part of a name (i.e., the same last name), the first login name in alphabetical order will be used.

See Also

commands, mail [overview], nptx, smail

mkfs — Command

Make a new file system
/etc/mkfs [-b *boot*] **[-d] [-f** *name*] **[-i** *inodes*] **[-m** *arg*] **[-n** *arg*] **[-p** *pack*] *filesystem proto*

mkfs makes a new file system. *filesystem* names the file (normally a block special file) where the new file system will reside. The contents of the newly created file system are described in *proto*. *proto* can be either a number or a file name.

If *proto* is a number, **mkfs** creates an empty file system (containing only a root directory) of the size in blocks given by *proto*. The number of i-nodes is calculated as a percentage of this number. The command

```
/etc/mkfs /dev/fha0 2400
```

creates a file system on a high-density, 5.25-inch diskette in drive 0. If the disk is a high-density, 3.5-inch diskette, use the command:

```
/etc/mkfs /dev/fva0 2880
```

If *proto* is a file name, however, the contents of that file will be used as a prototype for modeling the new file system. This prototype file must be laid out in the following manner:

```
bootstrap_file_name  file_system_name  device_name
no._of_blocks  no._of_i-nodes  n   m
%b XX XX XX
...
directory_name
        directory_name mode user_id group_id contents
        ...
        $
$
```

Each line is described below.

The first line has three fields. Field 1, *bootstrap_file_name*, contains the name of a file that holds the boot strap, which must fit into block 0 of the disk. Field 2, *file_system_name*, gives the name of the file system; and field 3, *device_name*, gives the name of file system's physical device (for example, **/dev/hd1**). Only the first six characters in field 2 and the first 11 in field 3 are significant; all characters after them are ignored.

The second line contains four fields. Field 1, *no._of_blocks*, gives the size of the file system in blocks; field 2, *no_of_i-nodes*, gives the number of i-nodes in the file system. Because each file or directory requires one i-node, this number represents the limit on the number of files that may be created in the file system. A ratio of seven blocks per i-node generally works well.

Fields 3 and 4 control free list interleaving on your disk. *n* is the size of a "virtual cylinder": **fsck** allocates all the blocks on one virtual cylinder before it advances to the next virtual cylinder. The value of *n* must be less than or equal to 255, and should evenly divide the actual size of a cylinder on the device. *m* tells the system how many blocks to skip each time it increments a free list block number, i.e., the free list "interleave"; *n* mod *m* must be zero. Choosing an optimal interleave value may improve system performance for the device. The optimal values for *n* and *m* are hardware-specific and can be determined by experimentation.

Next, the third line and following begin with **%b**. These list the bad blocks on your storage device. One or more block numbers may appear on each line, separated by white space. These blocks are allocated to the bad block file (i-node 1).

The remaining lines in the *proto* file define the names, modes, and contents of the directories and files in the file system. These lines are divided into fields separated by white space (blanks or tabs) as follows:

- The first field names the file or directory to be created. This field is missing on the first line, which describes the root directory of the file system.
- The second field describes the mode of the file, which is six characters long. The first character gives the file type, that is, whether the file is ordinary ('-'), directory ('d'), block special ('b'), or character special ('c'). The second character is 'u' for set user id on execution, and '-' otherwise. The third character is 'g' for set group id on execution, and '-' otherwise. Characters 4 through 6 specify permissions in octal; for example, **644** specifies read and write permission for the owner, read permission for other users from the same group, and read permission for users from other groups.
  If the above file type were a directory, subsequent files are recursively defined under that directory, until the current level of directory is terminated by a line containing a '$' character.
- The next two fields specify the owner's numerical user id and group id.
- The last field describes file contents. For a directory, it is not needed. For an ordinary file, it is the name of a COHERENT file that will be copied into the newly created file. For block or character-special files, there are two fields that specify the numbers of the major and minor devices.

Finally, each directory's description and the entire *proto* file must terminate with dollar signs '$'.

The *proto* file need not contain all of the above fields. However, it must contain the name of the boot block (line 1), the number of blocks and the number of i-nodes (line 2), the list of bad blocks, the name of at least one directory, and the dollar sign that ends the file.

Command-line Options

mkfs recognizes the following command-line options:

-b *boot*
Specifies the file to use as the "bootstrap" for the file system.

-d Preserve file dates and times on the new file system.

-f *name*
Label the file system with the given *name*. *name* must be less than seven characters in length.

-i *inodes*
Use *inodes* as the number of inodes for the file system.

-m *arg*
Set the number of blocks to skip when incrementing virtual block number. This is the same as the *m* option as set on line 2 of the prototype file. You can use this option if you choose not to use a prototype file.

-n *arg*
Set the size of a "virtual cylinder". This is the same as the *n* option as set on line 2 of the prototype file. You can use this option if you choose not to use a prototype file.

-p *pack*
Set the file system "pack name" to *pack*. *pack* must be less than seven characters in length.

Example

The following example specifies a proto file for a high-density, 5.25-inch floppy disk; note that this floppy disk is faulty and contains a number of bad blocks:

```
/conf/boot.fha
2400 100
%b 55
%b 185 86
d--755 3 1
       coherent ---644 3 1 /coherent
       tmp         d--777 3 1
       $
       bin         d--755 3 1
                   mail      -u-755 0 1 /bin/mail
       $
       dev         d--755 3 1
                   tty30     c--644 0 1 3 0
                   tty35     c--644 0 1 3 5
                   mt0       b--600 0 1 12 0
       $
$
```

You can use the command **badscan** to draw up the list of bad blocks on your disk and create a skeleton *proto* file.

See Also

badscan, chmod, commands, fsck, mount, restor

Notes

When the command **fsck** checks a file system, it stores files that it cannot decypher into directory **lost+found**. However, **fsck** cannot modify a file system during its work. This rule was adopted to prevent **fsck** from attempting to modify a corrupt file system, and so making matters worse. However, this means that (among other things) **fsck** cannot change the size of directory **lost+found**. Thus, if more files are detached from the file system than **lost+found** can hold, **fsck** must delete them outright. If your newly created file system will hold a large number of transient files (e.g., a news system), you should increase the size of **lost+found** so that it has a fighting chance of holding all detached files that **fsck** finds. For example, the following script expands **/lost+found** so it can hold up to 500 files:

```
su root
for i in `from 1 to 500`
do
      touch /lost+found/$i
done
rm /lost+found/*
```

Run this script for each file system whose **lost+found** directory you wish to expand. For example, if you have a file system mount on directory **/u**, run this script for directory **/u/lost+found** instead of for **/lost+found**.

mkhpath — Command

Build a pathalias data base from a hosts table
/usr/lib/mail/mkhostpath [-d] [-c *cost***] [-g** *gateway***] [-n** *netname***] [- |** *filename* **]**

The script **mkhpath** reads a hosts table and constructs routes to that network. *filename* holds the hosts table; if it is '-', **mkhpath** reads the standard input. If the command line does not name a *filename*, **mkhpath** reads file **/etc/hosts**.

mkhpath assumes its input to be in the format of the data-base file **/etc/hosts**. It ignores the first field (Internet address) and any domain-based name (any field containing a '.'). It also ignores the hosts **localhost** and **loghost**, and all comment lines (those that begin with a '#').

mkhpath recognizes the following command-line options:

-c *cost* Set the cost of accessing the gateway to be *cost*. *cost* can be any cost expression recognized by the command **pathalias**. **mkhpath** ignores this option if you do not also use option **-n**.

-d Print a line only if it contains a domain host name. This is useful for ignoring test lines.

-g *gateway*
Make *gateway* the gateway to the hosts.

-n *netname*
Form a network map instead a list of path aliases, and name the network *netname*. If you do not use this option, **mkhpath** assumes that your local host is within the network, and therefore inserts your local host into the network list. It also assumes that the cost of routes within the network is **LOCAL**, unless you use option **-c** to set the cost of the routes explicitly.

By default, **mkhpath** builds the route table in the same format as that generated by command **pathalias** with its option **-i**. Flag **-n** overrides this default.

If you use neither option **-g** nor **-n**, **mkhpath** constructs direct routes from your local host to each remote host.

If you use option **-g** but do not use option **-n**, **mkhpath** prefixes every route to the network (except for the route to the gateway) with the route *gateway***!**.

If you use the option **-n**, **mkhostpath** builds a pathalias map to the network this option names. The format of the map depends on whether you also use option **-g**.

If you use both options **-g** and **-n**, **mkhpath** establishes the route from your local host to *gateway*, and inserts the gateway into the network list. It does not add your local host to the network list, even if it appears as a site within the hosts table. It sets the cost of the link between your local host and the gateway is **LOCAL**; you can override this by using option **-c**. It fixes at **LOCAL** the cost of routes inside the network; this is not affected by the option **-c**.

See Also

commands, hosts, mail [overview], smail

Notes

Please note that because COHERENT does not yet support networking, this command is never used.

mkline — Command

Fold an alias file, paths file, or mailing list into one-line records
/usr/lib/mail/mkline [-ltn] [*file ...* **]**

Command **mkline** takes alias file, path file, or mailing-list file as input, and generates output records that contain one complete entry per line, and removes all comments and white space.

mkline recognizes the following command-line options:

-l Generate a list of addresses. Use this to generate a mailing list. If you use this option, **mkline** ignores options **-n** and **-t**.

-n Do not extract keys from the input. **mkline** passes all token through unchanged, although it still removes all comments and as much white space as it can without creating ambiguous output.

-t Separate the key from the data with a single tab character. The default is to use a colon ':'.

If its command line does not name an input *file*, **mkdbm** reads the standard input. **mkline** also reads the standard input if a *file* is named '-'; in this way, it can mix data read from the standard input with material read from files.

Examples

Consider the following alias file:

```
Postmaster:hustead                                      # Ted Hustead, jr.
UUCP-Postmasters: tron, chongo                          # namei contacts
yamato                                                  # kremvax contact
tron: tron@namei.uucp (Ronald S. Karr)
yamato: yamato@kremvax.ussr.comm (Yamato T. Yankelovich)
chongo: chongo@eek.uts.amdahl.com (Landon Curt Noll)
```

When it reads this file, **mkline** generates:

```
Postmaster:hustead
UUCP-Postmasters:tron,chongo yamato
tron:tron@namei.uucp
yamato:yamato@kremvax.ussr.comm
chongo:chongo@eek.uts.amdahl.com
```

As an example of using **mkline** to compress mailing lists, consider the mailing list:

```
tron@namei.uucp,tron@uts.amdahl.com                    # Ronald S. Karr
yamato@kremvax.ussr.comm                               # Yamato T. Yankelovich
chongo@eek.uts.amdahl.com                              # Landon Curt Noll
Wilt . (the Stilt) Chamberlain@NBA.US                  # RFC822 doc example
```

The command **mkline -l** generates the following:

```
tron@namei.uucp
tron@uts.amdahl.com
yamato@kremvax.ussr.comm
chongo@eek.uts.amdahl.com
Wilt.Chamberlain@NBA.US
```

See Also

commands, mail [overview], mkdbm, mksort, pathalias, smail

Notes

mkline leaves one space character if the concatenation of two tokens would otherwise cause ambiguity.

mkline frequently is used with the command **mksort**. For an example of using these commands together, see the Lexicon entry for **mksort**.

Copyright © 1987, 1988 Ronald S. Karr and Landon Curt Noll. Copyright © 1992 Ronald S. Karr.

mkdbm is part of the **smail** package. For a full copyright statement, see file **COPYING**, which is included with source code to **smail**, or type **smail -bc** to see the distribution rights and restrictions associated with this software.

mklost+found — Command

Make an enlarged lost+found directory
/etc/mklost+found *directory* **[***slots***]**

When the command **fsck** checks your file system, it copies all “orphaned” files into directory **lost+found** in the root directory of the file system being checked. Normally, this works well; however, if a great number of files are orphaned, directory **lost+found** may not be able to hold them all. This is because a directory is itself a file that holds information about the files it contains; normally, COHERENT expands the size of a directory file when it needs more space to hold files, but because **fsck** is forbidden to modify any file, it cannot enlarge **lost+found**. Thus, orphaned files that cannot be copied into **lost+found** are deleted.

Script **mklost+found** lets you build an enlarged **lost+found** directory within *directory*. It initializes the **lost+found** directory to be able to hold *slots* files. If you do not specify how many files you want **lost+found** to be able to hold, **mklost+found** initializes it to hold 250 files.

Example

The following command creates a **lost+found** directory that can hold 1,000 files for the file system that is mounted on directory **/news**:

```
/etc/mklost+found /news 1000
```

See Also

commands, fsck

Notes

Only the superuser **root** can run this script.

mknod — Command

Make a special file or named pipe
/etc/mknod [-f] *filename type major minor*
/etc/mknod [-f] *filename* **p**

In the first form, **mknod** creates a *special file,* which provides access to a device by the *filename* specified. Special files are conventionally stored in the **/dev** directory.

type can be either ‘b’ (for block-special file) or ‘c’ (for character-special file). Block-special files tend to be devices such as disks or magnetic tape, upon which COHERENT uses an elaborate buffering strategy. Character-special files are unstructured (character at a time) devices such as terminals, line printers, or communications devices. Character-special files may also be random-access devices; this circumvents system buffering, allowing transfers of arbitrary size directly between the user and the hardware.

The *major* device number uniquely identifies a device driver to COHERENT. The *minor* device number is a parameter interpreted by the driver; it might specify the channel of a multiplexor or the unit number of a drive.

The caller must be the superuser.

In the second form, **mknod** creates a named pipe with the given *filename.* Named pipes can be used for communication between processes.

The **-f** option to **mknod** forces the creation of a new node, even if one of the same name already exists.

Files

/dev/*

See Also

commands, mount

mknod() — System Call (libc)

Create a special file
#include <sys/ino.h>
#include <sys/stat.h>
mknod(*name, mode, addr***)**
char ****name*; int** *mode, addr***;**

mknod() is the COHERENT system call that creates a special file. A *special file* is one through which a device is accessed, or a named pipe.

mode gives the type of special file to be created. It can be set to **IFBLK**, for a block-special device, such as a disk driver; to **IFCHR**, for a character-special device, such as a serial-port driver; to **IFDIR**, for a directory; or to **IFPIPE**, for a named pipe. *mode* also contains permission mode bits.

address is a parameter interpreted by the driver; it might specify the channel of a multiplexor or the unit number of a drive. Note that this is not used with named pipes.

If all goes well, **mknod()** returns zero. If an error occurs, it returns a negative value and sets **errno** to an appropriate value.

See Also

libc, device drivers, named pipe, pipe()

Notes

Only the superuser **root** can use **mknod()**. This is a security feature.

mkpath — Command

Create a pathalias output file
**/usr/lib/mail/mkpath [-v] [-V] [-x] [-e] [-n] **
        [-t *trace***] [***path_config***]**

The script **mkpath** is a wrapper for the command **pathalias**, which generates a set of paths among computers.

File *path_config* holds the data that are passed to **pathalias**; if it is set to '-', then **pathalias** reads the standard input. If no *path_config* is named on the command line, **mkpath** by default uses file **/usr/lib/mail/maps/mkpath.conf**.

mkpath recognizes the following command-line options:

-e Tell **mkpath** to stop when it encounters a syntax error, or if a command that it invokes exits with a non-zero status.

-n Disable the execution of any commands useful with the Bourne shell's option **-v**, and disables its own options **-e**, **-t**, **-V**, and **-x**.

-t *tracefile*
Copy into *tracefile* all data passed to **pathalias**.

-V Invoke command **pathalias** with its option **-v**.

-v Verbose mode. Its commands are executed with the Bourne shell's option **-v**, which echoes each command as it is read.

-x Verbose mode. Its commands are executed with the Bourne shell's option **-x**, which echoes each command as it is executed.

See Also

commands, mail [overview], pathalias, smail

mksort — Command

Sort the standard input, allowing arbitrarily long lines
/usr/lib/mail/mksort [-f] [*file ...* **]**

The command **mksort** reads lines of text, sorts them by the first field in each line, then writes them to the standard output. It usually is used by system administrators to help prepare the data files used by **smail**. Unlike the COHERENT command **sort**, **mksort** can read and process an arbitrarily long line of text.

The first field within a line of input is delimited either by a white-space character or a colon ':'. A line can be of any length, as long as the entire input can be stored in memory. Command-lind option **-f** (for "fold") tells **mksort** to ignore case when it sorts its input; with this option, the letter 'A' equals the letter 'a', and 'a' is always less than 'B'.

If its command line does not name an input file, **mksort** reads the standard input. A file name of '-' indicates the standard input; this permits **mksort** to mingle the contents of one or more files with what it reads from the standard input.

Example

The following example demonstrates how to use **mksort** with **mkline**. Consider file **aliases**, which contains the following aliasing information:

```
Postmaster:hustead # Ted Hustead, jr.
UUCP-Postmasters: tron, chongo   # namei contacts
    yamato  # kremvax contact
tron:      tron@namei.uucp (Ronald S. Karr)
yamato:    yamato@kremvax.ussr.comm (Yamato T. Yankelovich)
chongo:    chongo@eek.uts.amdahl.com (Landon Curt Noll)
```

Given this file, the command

```
mkline aliases | mksort -f
```

yields:

```
chongo:chongo@eek.uts.amdahl.com
Postmaster:hustead
tron:tron@namei.uucp
UUCP-Postmasters:tron,chongo yamato
yamato:yamato@kremvax.ussr.comm
```

See Also

commands, mail [overview], mkline, mkdbm, pathalias, smail

Notes

This command is not used by COHERENT's implementation of **smail**.

Copyright © 1987, 1988 Ronald S. Karr and Landon Curt Noll. Copyright © 1992 Ronald S. Karr.

For details on the distribution rights and restrictions associated with this software, see file **COPYING**, which is included with the source code to the **smail** system; or type the command: **smail -bc**.

mktemp() — General Function (libc)

Generate a temporary file name
char *mktemp(*pattern***) char ****pattern***;**

mktemp() generates a unique file name. It can be used, for example, to name intermediate data files. *pattern* must consist of a string with six **X**'s at the end. **mktemp** replaces these **X**'s with the five-digit process id of the requesting process and a letter that is changed for each subsequent call. **mktemp** returns *pattern*. For example, the call

```
	char template[] = "/tmp/sortXXXXXX";
	mktemp(template);
```

might return the name **/tmp/sort01234a**.

It is normal practice to write temporary files into the directory **/tmp**. The start of the file name identifies the originator of the file.

See Also

libc

Notes

Because **mktemp()** writes on the argument template, passing it a quoted string causes a segmentation violation. To avoid this, either compile the module that contains the call to **mktemp()** with the compiler option **-VPSTR** (to put the quoted string into segment **.data** rather than into segment **.text**) or, preferably, move the string into the data segment. For example, use

```
	char template[] = "/tmp/sortXXXXXX";
	mktemp(template);
```

rather than:

```
	mktemp("/tmp/sortXXXXXX");
```

mktime() — Time Function (libc)

Turn broken-down time into calendar time
#include <time.h>
time_t mktime(*timeptr***)**
struct tm ****timeptr*;**

mktime() reads broken-down time from the structure pointed to by *timeptr* and converts it into calendar time of the type **time_t**. It does the opposite of the functions **localtime()** and **gmtime()**, which turn calendar time into broken-down time.

mktime() manipulates the structure **tm** as follows:

1. It reads the contents of the structure, but ignores the fields **tm_wday** and **tm_yday**.
2. The original values of the other fields within the **tm** structure are not restricted. This allows you, for example, to increment the member **tm_hour** to discover the calendar time one hour hence, even if that forces the value of **tm_hour** to be greater than 23, its normal limit.
3. When calculation is completed, the values of the fields within the **tm** structure are reset to within their normal limits to conform to the newly calculated calendar time. The value of **tm_mday** is not set until after the values of **tm_mon** and **tm_year**.
4. The calendar time is returned.

If the calendar time cannot be calculated, **mktime** returns -1 cast to **time_t**.

Example

This example gets the date from the user and writes it into a **tm** structure.

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define BAD_TIME ((time_t)-1)

/* ask for a number and return it. */
int askint(msg)
char *msg;
{
      char buf[20];

      printf("Enter %s ", msg);
      fflush(stdout);

      if(gets(buf) == NULL)
            exit(EXIT_SUCCESS);
      return(atoi(buf));
}

main()
{
      struct tm t;

      for(;;) {
            t.tm_mon  = askint("month") - 1;
            t.tm_mday = askint("day");
            t.tm_year = askint("year") - 1900;
            t.tm_hour = t.tm_min = t.tm_sec = 1;

            if(BAD_TIME == mktime(&t)) {
                  printf("Invalid date\n");
                  continue;
            }

            printf("Day of week is %d\n", t.tm_wday);
            break;
      }
      return(EXIT_SUCCESS);
}
```

See Also

clock(), difftime(), libc, time [overview]
ANSI Standard, §7.12.2.3
POSIX Standard, §8.1

Notes

The above description may appear to be needlessly complex. However, the Committee intended that **mktime()** be used to implement a portable mechanism for determining time and for controlling time-dependent loops. This function is needed because not every environment describes time internally as a multiple of a known time unit.

MLP_COPIES — Environmental Variable

Set default number of copies to print

When the command **lp** spools a job for printing, it reads the environmental variable **MLP_COPIES** to find how many copies are to be printed. The default is one copy; the maximum is 99.

See Also

environmental variables, lp, lpadmin, lpsched, printer

MLP_FORMLEN — Environmental Variable

Set default page length

When the command **lp** spools a job for printing, it reads the environmental variable **MLP_FORMLEN** to find the length, in lines, of the form on which the job is to be printed. In the United States, a *line* is defined to be one pica high (that is, one sixth of an inch). The default is length 66 lines (11 inches). (NB, the COHERENT command **units** gives a handy way to convert from picas or inches into metric units.)

The printer daemon **lpsched** uses this information to help it count pages of input — so you can specify the range of pages that it should print. Unfortunately, **lpsched** identifies a page by counting lines of input, so this feature works only it prints "straight" text. It does *not* work correctly with "cooked" input, such as files of PostScript or PCL.

See Also

environmental variables, lp, lpadmin, printer

MLP_LIFE — Environmental Variable

Set default life for print jobs

When the command **lp** spools a job for printing, it reads the environmental variable **MLP_LIFE** to set the job's "life expectancy". **MLP_LIFE** must be one of the following:

T	Temporary: live in the queue for two hours.
S	Short-term: live in the queue for 48 hours.
L	Long-term: live in the queue for 72 hours.

The default life expectancy is **S**.

To change the default values for life-expectancies, edit the file **/usr/spool/mlp/control**. For details, see the Lexicon article **controls**.

See Also

environmental variables, lp, lpadmin, printer

MLP_PRIORITY — Environmental Variable

Set default priority for print spooling

When the command **lp** spools a job for printing, it reads the environmental variable **MLP_PRIORITY** to find what priority it is to give the job. **MLP_PRIORITY** must a numeral, from **0** to **9**: **0** assigns highest priority, **9** lowest. The default priority is **2**.

See Also

environmental variables, lp, lpadmin, printer

MLP_SPOOL — Environmental Variable

Pass user-specific information to print spooler

When the command **lp** spools a job for printing, it reads the environmental variable **MLP_SPOOL** to find user-specific information for this job. **MLP_SPOOL** must have the following layout:

Offset	*Length*	*Description*
0	10	Type of document (user-specific)
10	3	Page length, lines per page (default, 66)
13	14	Name of data base (user-specific)
28	14	Name of program (user-specific)
42	60	Title (user-specific)

With the exception of page length, **lp** uses none of these fields itself; rather, it makes them available to whatever program the user (or system administrator) has selected to process text before it is printed.

See Also

environmental variables, lp, lpadmin, printer

mmu.h — Header File

Definitions for memory-management unit
#include <sys/mmu.h>

The header file **mmu.h** defines functions that manipulate the memory-management unit (MMU) of the Intel 80X86 family of microprocessors.

See Also

header files

mneg() — Multiple-Precision Mathematics (libmp)

Negate multiple-precision integer
#include <mprec.h>
void mneg(*a, b***)**
mint **a*, **b***;**

mneg() negates the value of the multiple-precision integer (or **mint**) pointed to by *a*, and writes the result into the **mint** pointed to by *b*.

See Also

libmp

mnttab — System Administration

Mount table
/etc/mnttab

File **/etc/mnttab** holds the COHERENT system's mount table. It consists of an array of type **mnttab**, which is defined in header file **mnttab.h**.

See Also

Administering COHERENT, mnttab

mnttab.h — Header File

Structure for mount table
#include <mnttab.h>

mnttab.h defines the structure for the mount table maintained by the functions **/etc/mount** and **/etc/umount**.

File **/etc/mnttab** is an array of these structures.

See Also

header files, mount, umount

modem — Technical Information

The word *modem* is an abbreviation for "modulation/demodulation device". With the COHERENT system, you can attach a modem to your computer either to dial out for remote communication, to let others dial into your COHERENT system, or both. With your modem, too, you can use COHERENT's UUCP commands to exchange mail and files with remote sites automatically, and to download news and files from networks.

This article gives a summary of how to connect your modem to your computer, describe it to the COHERENT system, and set it up for UUCP connections. It also discusses some problems that may crop up when you attempt to use your modem.

Internal vs. External Modems

You can use internal and external modems with COHERENT. You must plug an external modem into a serial port on your system, whereas you must jumper an internal modem to use one of your system's COM ports. Be sure to use a COM port that is not already used on your system, or problems will result. See the Lexicon entry for **asy** for details on how COHERENT handles COM ports.

It is more difficult to diagnose problems with an internal modem because you have no status lights to indicate operation; otherwise, they operate almost identically. The rest of this article assumes that you are working with an external modem.

Plugging in an External Modem

A modem must be hooked up to a serial port on your computer. To plug your modem into the computer, simply take a normal serial-port cable, one with an RS-232 plug of the appropriate gender at each end, plug one end into your modem and the other into the serial port you wish to use. The Lexicon article **RS-232** describes the wiring of the RS-232 plug in detail; but if you are not skilled with a soldering iron, you are well advised simply to purchase a cable from your local electronics store and be done with it.

Serial Ports

The COHERENT system supports up to four serial ports; the devices for these are named **/dev/com1r** through **/dev/com4r**. If you are not sure which port you have plugged your modem into, perform the following test: First, turn on the modem. Then, type the following command:

```
echo FOO >/dev/com1l
```

If the **TX** light on the modem blinks, then you know the modem is plugged into **com1**. If it does not, try the command again for **/dev/com2l**, and so on through **com4l** until you find the appropriate port. If no command works, check the wiring on your cable and make sure that the plugs are securely inserted.

Edit /etc/ttys

If you intend to use your modem with UUCP, you must edit file **/etc/ttys** to tell COHERENT how you want it to handle that serial port. You must know (1) whether you want the port enabled or disabled; (2) the baud rate of the port (as set by your modem); and (3) the name of the port (which you just determined).

If a port is enabled, remote users can log into the system, either via a terminal directly plugged into the port or via a modem. COHERENT sends a login prompt to every enabled port. The COHERENT system also restricts permissions on all enabled serial ports, so that only the superuser **root** can read and write to the port. This prevents other users who may be using the system from accessing the serial port. If a port is disabled, you can dial out or use a direct-connect UUCP connection via that disabled port. To dial out on an enabled port, you must first use the command **disable** to disable the port. When you have finished dialing out, run the command **enable** to re-enable the port. (Note that UUCP automatically disables and re-enables a port when it dials out to poll a remote system.) Before you can use these commands with a port, the port must first be described in the file **/etc/ttys**.

See the Lexicon article on **ttys** for details on how to edit this file. Note that a modem is a remote device, and must be so described in **/etc/ttys**, or it will not work correctly.

After you have made your changes, type the command

```
kill quit 1
```

to make COHERENT re-read **/etc/ttys** and implement your changes.

Remote-Access Passwords

If you intend to let people dial into your computer, you are well advised to set the remote-access password. This will require that people who dial in know a special password in addition to whatever password their personal account may have.

If you wish, you can set a different remote-access password for each group of users who log into your system, as organized by the program invoked upon logging in. For example, you can give one password to the users who log in and invoke **uucico**; and another to the users who log in and use the interactive shells **ksh** or **sh**. For details on how to do this, see the Lexicon entries for **d_passwd** and **dialups**.

Edit /usr/lib/uucp/dial

Once you have edited file **/etc/ttys** and have set the remote-access password, check the file **/usr/lib/uucp/dial** and see if it holds a description that matches your modem. The commands **cu** and **uucico** read the descriptions in **dial** to control how they talk to modems. **dial** already contains descriptions for many commonly used modems; but you may find that you must edit an existing entry to match your modem's features exactly; for example, the existing entry may assume that you have a Touch-Tone telephone whereas you actually have a pulse telephone. The Lexicon entry on **dial** will walk you through this process.

When you have completed editing this entry, write it down, for you will need to insert it elsewhere.

Edit Port

If you intend to use your modem with UUCP, you must insert an entry for it into your the file **/usr/lib/uucp/port**. This file links a modem, as described in file **/usr/lib/uucp/dial**, with a port on your system. This arrangement permits UUCP to use one description with several modems of the same type, each plugged into a different port.

See the Lexicon entry **port** for details.

Walking Through UUCP Configuration

The following description walks you through the task of configuring your modem to handle UUCP. It is adapted from a posting to **comp.os.coherent** by Rob Schofield (schofld@mebv.mhs.compuserve.com).

First, decide whether you want outsiders (including outside UUCP sites) to log into your COHERENT system. If you do, then you must add to file **/etc/ttys** the name the incoming device — that is, the device that the remote users will log into. If you do not want incoming logins, you do not need to have an incoming device installed in **/etc/ttys** and you can safely omit it.

As described above, an entry in **/etc/ttys** consists of three one-character fields, followed by the name of the device:

- The first field indicates whether the device is enabled (that is, gets a login prompt) or disabled (that is, does not get a login prompt).
- The second field indicates whether the device is in "raw" mode or whether it "cooks" its input (that is, handles backspaces correctly, and so on). You should use 'l' (for cooked input).
- The third field gives the speed of the port; see the Lexicon entry **/etc/ttys** for a list of recognized codes.
- The device has the name **/dev/com?r**. The '?' in this name stands for the number of the COM port into which you've plugged your modem, from '1' to '4'. The 'r' in the device name stands for the "remote" (i.e., modem) device. If your modem is high speed (i.e., faster than 9600 baud) then use the hardware-handshaking version of the remote device (i.e., **/dev/com?fr**).

For example, if you have plugged a 14.4-kilobaud modem into serial port 3, insert the following line into file **/etc/ttys**:

```
1lQcom3fr
```

Once you have inserted this line into **/etc/ttys**, type the command:

```
kill quit 1
```

This forces COHERENT to re-read **/etc/ttys** and so recognize your change.

If you wish to dial out on your modem via programs **cu** or **ckermit**, or if you wish to have your UUCP system dial other, remote sites, those systems must use the *local* **/dev/com?l** on the same port number as your modem. If it is high speed, again use the 'f' version **/dev/com?fl**, which enables hardware handshaking. This sounds may sound strange (after all, why use a terminal-type device on a modem?), but there's a reason for it. When you use the UNIX or COHERENT system call **open()** on a **com?r** port, the function call does not return until it detects a "true" value on DCD — and that occurs only when someone has dialed in and the modems have connected. By using a **com?r** device, you are only setting up the system for a **getty** to detect someone dialing in; if you're dialing out, you do not need to detect DCD, hence the use of a terminal device. Hence, **cu**, UUCP, and **ckermit** should all be used with the outgoing port device, and not the incoming.

Do *not* add this port to **/etc/ttys**; rather, add it to the configuration files used by the applications. In the case of **ckermit**, use its command **set speed**. You can type this command either by hand, when you invoke **ckermit**; or you can add it to file **.kermrc** in your home directory. For details, see the Lexicon entry for **ckermit**. In the case of **cu** and UUCP, the device must be named in the file **/usr/lib/uucp/port**. For example, to dial out via our 14.4-kilobaud modem plugged into COM 3, add the following entry to **/usr/lib/uucp/port**:

```
port exampleport
type modem
device /dev/com3fl
baud 19200
dialer exampledialer
```

The device is **/dev/com3fl**, not the device **/dev/com3fr** we added to **/etc/ttys**. The 'r' version of a port is used exclusively for dialing in; the 'l' version for dialing out.

Last little trick is to link the device you are using to a pseudo device used by a few communication packages:

```
ln -f /dev/com?fl /dev/modem
```

Be sure to substitute the number of the port you're using (from '1' through '4') for the '?' in the above example.

Modem Maladies

This section discusses problems that have arisen with remote login via modem, as diagnosed by the technical support staff of Mark Williams Company.

Difficulty in logging in from a remote site via modem can be the result of problems in one or more of the following: cabling; enabling/disabling the port; flaws in the contents of file **/etc/ttys**; incorrect configuration of the modem; and setting the port to an incorrect state. See Lexicon articles **terminal** and **UUCP** for additional information. The following paragraphs discuss the above-named items in detail.

RS-232 Cabling

When attaching an external modem to your computer, it is important to use a modem cable that supports "full modem control". COHERENT relies on modem-control signals when operating a modem for remote access purposes. When attaching a terminal directly to a serial port, a "null modem" cable must be used. When attaching a modem, a "straight through" cable must be used. See Lexicon articles **RS-232** and **terminal** for further details on cabling.

Enabled vs. Disabled Ports

A serial port can be either enabled or disabled for remote access. Enabling a port allows a user on a remote terminal or modem to log into your COHERENT system. Disabling a port permits a user to dial out or use a direct connect UUCP connection via that disabled port.

If a port is enabled for remote logins and you will use it to call out, you must use the command **disable** to disable the port before you access the port. UUCP automatically disables and re-enables a port.

The port name supplied to an **enable** or **disable** command must *exactly* match the last part of a line in the **/etc/ttys** file (see below). For example, for the command **enable com2pr** to work, there must be an entry in the file **/etc/ttys** which ends with **com2pr**.

When a port is enabled, the first character for the port in file **/etc/ttys** is set to a '1' (one), the permissions for the port are changed so that only the superuser **root** can read and write to the port (to prevent other users on the system from accessing the port while a remote session is in progress), and a login prompt is sent to the port.

ttys *Problems*

This file should have permissions of 644 (-rw-r--r--) and belong to owner and group **root**. Review the Lexicon entry for **ttys** to ensure that the format of your version of **/etc/ttys** is correct.

Leaving blanks at the end of a line in **/etc/ttys** usually results in error messages stating that a device could not be found.

You do not need to edit the initial '0' or '1' in entries in **/etc/ttys**; this digit is updated by the commands **enable** and **disable**. See the Lexicon entries for **enable** and **disable** for more information.

Constant Flickering

Another problem is a constant flickering of send/receive LEDs and an unexplained continual access of the hard drive. This occurs when the port is enabled and the modem is set in echo mode: COHERENT sends the login prompt to the modem, the modem echoes it back to COHERENT, COHERENT then thinks the modem is trying to talk to it and sends the password prompt, and so on *ad infinitum*.

To fix this problem, place the modem into no-echo mode, and turn off the display of result codes. The following section discusses this in more detail.

Modem Configuration

A modem that fails to answer an incoming call, hangs up before locking onto the remote carrier, becomes stuck in a loop echoing characters sent to it from the computer, or fails to operate at the expected baud rate probably is configured improperly. To remedy this situation, send the appropriate control string to the modem.

We offer some guidelines here for modem settings. Be warned, however, that modems from different manufacturers usually behave differently, regardless of claims of Hayes compatibility: you must check the manual for your modem.

- Echo should be OFF (usually by setting "E0").
- Result codes should be OFF (usually by setting "Q1").
- Modem status "DCD" should follow true carrier detect status, rather than being always on (usually by setting "&C1").
- Auto answer should be ON (usually obtained by setting register S0 to a nonzero value equal to the number of rings before answer).
- The delay value for "Wait for Carrier/Dial Tone" (usually register S7) should not be too short.

The scripts below show typical initialization for a "Hayes-compatible" modem that runs at 2400 baud and is plugged into port **/dev/com3r**. It is only an example; your modem may need something different. Please note that the commands **sleep** and **stty** are necessary in the first example so that the command string will be sent to the modem at 2400 baud; otherwise, the string is sent at the default port speed, which is 9600 baud.

```
# initialize 2400-baud Hayes-compatible modem
sleep 3 > /dev/com3l &
stty 2400 < /dev/com3l
echo 'AT E0 Q1 V0 S0=1 &C1 M3' > /dev/com3l
sleep 3
```

The following gives a similar script for a Trailblazer modem that runs at 9600 baud and is plugged into port **/dev/com2r**:

```
# initialize 9600 baud internal Trailblazer on com2
/etc/disable com2r
sleep 3 > /dev/com2l &
stty 9600 < /dev/com2l
echo 'AT E0 T V0 X3 H0' > /dev/com2l
echo 'AT S0=1 S7=60 S48=1 S51=252 S52=0 S54=3 S58=2' > /dev/com2l
/etc/enable com2r
```

Modem Control

This section describes the modem-control protocol used by the driver **asy**, which COHERENT uses to control serial ports. *Modem control* describes how COHERENT handles RS-232 signals other than "Receive Data" and "Transmit Data".

Many processes can open a device at the same time. *First open* occurs if a process opens a device when no process has opened the device. *Last close* occurs when a process closes the port and no other remaining process has the port open. On first open, RTS and DTR are asserted by the computer, regardless of whether the specified device used modem control. If modem control is used (the high-order bit in minor number set to zero), **open()** does not complete until CD is true. Once an **al[01]** device has been opened with modem control, loss of CD to that port causes **SIGHUP** to be sent to all processes in the group keeping the port open.

See Also

Administering COHERENT, dial, RS-232, terminal, UUCP

Notes

One final bit of hard-won wisdom: once you have something working, write down what you did, and store it in a place where you won't lose it.

modf() — General Function (libc)

Separate integral part and fraction
#include <math.h>
double modf(*real***,** *ip***)**
double *real, *ip***;**

modf() is the floating-point modulus function. It returns the fractional part of its argument *real*, and stores the integral part in the location to which *ip* points. These numbers satisfy the equation *real* = *f* + **ip*.

Example

This example prompts for a number from the keyboard, then uses **modf()** to calculate the number's fractional portion.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main()
{
      double real, fp, ip;
      char string[64];

      for (;;) {
            printf("Enter number: ");
            if (gets(string) == 0)
                  break;
            real = atof(string);
            fp = modf(real, &ip);
            printf("%lf is the integral part of %lf\n",
                  ip, real);
            printf("%lf is the fractional part of %lf\n",
                  fp, real);
      }
}
```

See Also

atof(), ceil(), fabs(), floor(), frexp(), ldexp(), libc
ANSI Standard, §7.5.4.6
POSIX Standard, §8.1

Notes

In releases prior to version 4.0, the COHERENT implementation of **modf()** handled negative numbers by returning a integral part less than *real*, and a positive fraction. Now, it returns an integral part greater than *real*, and a negative fraction. For example, the old version of **modf()** would transform -1.9 into an integer of -2.0 and a fraction of 0.1; whereas the current version transforms -1.9 into an integer of -1.0 and a fraction of -0.9.

The behavior of **modf()** was changed to conform to the ANSI Standard.

modulus — Definition

Modulus is the operation that returns the remainder of a division operation. For example, 12 modulus four equals zero, because when 12 is divided by four it leaves no remainder. The term “modulo” also refers to the product of a modulus operation; in the above example, the modulo is zero. In C, the modulus operation is indicated with a percent sign ‘%’; therefore, 12 modulus 4 is written **12%4**.

The modulus operation often is used to trim numbers to a preset range. For example, if you wanted to create a list of single-digit random numbers, you would use the command:

```
      rand()%10
```

This is demonstrated by the following example.

Example

This example prints a list of 20 single-digit random numbers. The random-number table is seeded with a portion of the current system time.

```
#include <stdio.h>
#include <stdlib.h>

main()
{
      long nowhere;     /* place to put unused data */
      int counter;

      srand((int)time(&nowhere));
      for (counter = 0; counter <20; counter++)
            printf("%d\n", rand()%10);
}
```

See Also

operator, Programming COHERENT

Notes

The implementation of C defines how a modulus operator behaves when it operates upon numbers with different signs. On the i8086,

```
      10 % -4
```

yields -2. This is not mathematical modulus, which is +2.

mon.h — Header File

Read profile output files
#include <mon.h>

mon.h is used with programs that read the profiling routines' output files.

See Also

header files

moo — Command

Greatly amusing numeric guessing game
/usr/games/moo [*numdigits* **]**

moo is a guessing game of numbers, typically four digits, all different.

The game randomly selects a number that consists of *numdigits* unique digits. Obviously, *numdigits* cannot exceed ten; the default is four. **moo** then prompts you to guess the number it has selected. When you type your guess, **moo** responds with one of two possible answers. If you guess the number correctly, i.e., win, **moo** responds with "Right!". If any of the digits that you guessed were correct digits, but in the wrong place, you get a "cow." If you guess a digit correctly and in the correct place, you get a "bull." If the number of "bulls" is the same as the number of digits in the guess, you win. **moo** typically responds with a count of "bulls" and "cows," as in:

```
2 bulls, 1 cow.
```

See Also

commands

Notes

moo is sometimes also called **mastermind**.

It will never replace **DOOM**.

more — Command

Display text one page at a time
more [-cdflsu] [*-window_size* **] [+***line_number* **] [+/***pattern* **] [** *file ...* **] [-]**

more is a filter for paging through text one screenful at a time. *file* is a text file; the operator **-** tells **more** to read and display the standard input.

Command-line Options

more reads options from the command line and from the environmental variable **MORE**. In case of a conflict, the options given on the command line take precedence. Every cluster of options must be preceded with a hyphen '-', even if passed via the environmental variable **MORE**.

more recognizes the following options:

-c Paint the screen from the top line down. **more** normally repaints the screen by scrolling from the bottom of the screen.

-d Prompt the user at the end of each screen with the message:

```
[Press space to continue, 'q' to quit.]
```

The default is to not issue a prompt.

-f Count actual lines from the input file rather than screen lines. This option is useful when the input contains escape sequences that **more** does not recognize.

-l Do not treat the formfeed character **<ctrl-L>** as special. By default, **more** pauses at each formfeed character, as if a full screen had been displayed.

-s Squeeze consecutive blank lines into one blank line. This is useful for looking at **nroff** output, such as manual pages.

-u Display backspaces as control characters and leave the carriage return-linefeed (CR-LF) pair alone. By default, **more** displays backspaces that appear adjacent to an underscore character as underlined text; backspaces that appear between two identical characters as emboldened text; and compresses CR-LF sequences.

+/*pattern*
Search for *pattern* before displaying a file. *pattern* is a regular expression, as recognized by commands **ed** or **egrep**. *pattern* should be escaped to avoid being processed by the shell.

-*window_size*
Set the size of the window that **more** displays to *window_size*, which is a decimal integer less than or equal to the number of lines on your terminal. The default window size is read from the **termcap** description for your terminal.

+*line_number*
Make *line_number* the beginning line to display in *file*. *line_number* is a decimal integer less than the number of lines in *file*.

Commands

The following describes **more**'s interactive commands. These commands are based on those for the UNIX editor **vi**. Some commands may optionally be preceded by a decimal number. If you enter an invalid command, **more** will beep at you.

h
? Help: display a summary of these commands.

*[N]***<space>**
Display the next *N* lines of text (default, one screenful).

*[N]***z**
If *N* is not specified, display the next screenful. Otherwise, display *N* lines and set the default scrolling size to *N* for all subsequent **<space>** and **z** commands.

*[N]***<ctrl-F>**
*[N]***f**
Scroll forward *N* screenfuls (default, one screenful). If *N* is more than the screen size, only the final screenful is displayed.

*[N]***<ctrl-B>**
*[N]***b**
Scroll backward *N* screenfuls (default, one screenful). If *N* is more than the screen size, only the final screenful is displayed.

*[N]***s**
Skip forward *N* lines (default, one line) and display one screenful.

*[N]***<return>**
*[N]***<enter>**
Scroll forward *N* lines (default, one). Display all *N* lines, even if *N* is more than the screen size.

*[N]***<ctrl-D>**
*[N]***d**
Scroll forward *N* lines (default, one half of the screen size). If *N* is specified, it becomes the new default for subsequent **d** and **<ctrl-D>** commands.

<ctrl-L>
Redraw the screen.

' (Apostrophe) Return to the position in the current file where the previous search command started, or to the beginning of the file if no search commands have occurred. This information is lost when a new file is examined.

*[N]***/***pattern*
Search forward for the *N*-th line that contains *pattern* (default, one). *pattern* is a regular expression, as recognized by **ed** or **egrep**. The search starts at the second line displayed.

n Repeat previous search.

:f Display the name of the current file with the current line number.

*[N]***:n**
Examine the *N*-th file after the current file, as given in the command line (default, the next file).

*[N]***:p**
Examine the *N*-th file previous to the current file, as given in the command line (default, the previous file).

! *command*
:! *command*
Pass *command* to the shell specified by environment variable **SHELL** for execution. The default shell is **/bin/sh**.

v Invoke an editor to edit the current file. The editor is set by the environment variables **VISUAL** and **EDITOR**, in that order. If these variables are not set, use **vi**.

= Display the current line number.

q
:q
Q
:Q Quit.

Environment

more uses the following environment variables:

EDITOR	Specify default editor.
MORE	Set default options for **more**
SHELL	Specify the shell being used (normally set at login time).
TERM	Specify the type of terminal you are using. **more** uses this variable to read from **/etc/termcap** the terminal characteristics needed to manipulate the screen.
VISUAL	Specify default visual editor.

See Also

commands, egrep, scat, vi, zmore

Author

This software is derived from software contributed to Berkeley by Mark Nudleman. **more** is copyright © 1988,1990 by The Regents of the University of California. Copyright © 1988 by Mark Nudleman. All rights reserved.

motd — System Administration

File that holds message of the day
/etc/motd

The file **motd** holds the message of the day. Its contents are displayed by the script **/etc/profile**, which is executed whenever you log in.

Only the superuser can alter the contents of this file.

See Also

Administering COHERENT, login

mount — Command

Mount a file system
/etc/mount [*device directory* **[-ru]]**

The command **mount** mounts a file system from *device* onto *directory*. In effect, it grafts the root directory of file system on *device* onto *directory*.

If you invoke **mount** without any arguments, it displays information about the file systems that are now mounted.

If you use option **-r**, **mount** mounts the specified file system in read-only mode. This is useful if you wish to read a file system without changing it in any such way, such as when you are backing it up. Note, however, that when a file system is mounted in read-only mode, COHERENT does not update file-system information, such the time a file was last accessed.

The option **-u** tells **mount** to write an entry into the mount-table file **/etc/mtab** without actually mounting the file system. When this is done, COHERENT will hereafter mount the file system automatically whenever you boot COHERENT.

Please note that unlike every other COHERENT or UNIX command ever devised, **mount** requires that its options *follow* the file names, rather than precede them. The COHERENT version of **mount** follows this convention in order to conform to this established UNIX practice.

To un-mount a file system, use the command **umount**. (NB, this is not a typographical error — this command's name contains only one 'n'.)

The script **/bin/mount** calls **/etc/mount**, and provides convenient abbreviations for commonly used devices. For example,

```
mount f0
```

executes the command:

```
/etc/mount /dev/fha0 /f0
```

You should edit this script to reflect the devices that you use on your system.

Files

/etc/mtab — Mount table
/etc/mnttab — Mount table
/bin/mount — Shell script that calls **/etc/mount**

See Also

commands, fsck, mkfs, mknod, umount

Diagnostics

Errors can occur if *device* or *directory* does not exist or if you do not have permission to access *device*.

The message

```
/etc/mtab older than /etc/boottime
```

indicates that **/etc/mtab** has probably been invalidated by booting the system.

Attempting to mount a block-special file that does not contain a COHERENT file system (e.g., a tape device) can have disastrous consequences. *Caveat utilitor!* To build a file system on a block-special device, use the command **/etc/mkfs**. For details, see its entry in the Lexicon.

mount.h — Header File

Define the mount table
#include <sys/mount.h>

mount.h defines the structures and constants that comprise the COHERENT system's mount table. It also declares functions that are used internally by routines that manipulate the mount table.

See Also

header files, mount

mount() — System Call (libc)

Mount a file system
#include <sys/mount.h>
#include <sys/filsys.h>
int mount (*device***,** *name***,** *flag***)**
char ****device*, ****name***; int** *flag***;**

mount() is the COHERENT system call that mounts a file system. *device* names the physical device that through which the file system is accessed. *name* names the root directory of the newly mounted file system. *flag* controls the manner in which the file system is mounted, as set in header file **sys/mount.h**.

See Also

fd, libc, mount, mount.h

mount.all — System Administration

Mount file systems at boot time
/etc/mount.all

The file **/etc/mount.all** holds a set of **mount** commands to mount all COHERENT file systems on hard disk. It is invoked by the script **/etc/rc**, which COHERENT reads and executes at boot-time.

When you add a new COHERENT partition to your system, you should insert an appropriate entry into this file, to ensure that the new partition is mounted whenever you reboot your system. You should also insert an entry into **/etc/checklist**, to ensure that the utility **fsck** examines and corrects the file system on this new partition before the system mounts it.

See Also

Administering COHERENT, checklist, mount, rc

mout() — Multiple-Precision Mathematics (libmp)

Write multiple-precision integer to stdout
#include <mprec.h>
void mout(*a***)**
mint **a***;**

mout() writes the multiple-precision integer (or **mint**) pointed to by *a* onto the standard output. The base of the output is set by the value of the external variable **obase**.

See Also

libmp

mprec.h — Header File

Multiple-precision arithmetic
#include <mprec.h>

The header file **mprec.h** declares a set of routines used to perform multiple-precision arithmetic. It also declares the structure **mint**, which holds multiple-precision integers.

See Also

header files, libmp

mrand48() — Random-Number Function (libc)

Return a 48-bit pseudo-random number as a long integer
long mrand48();

Function **mrand48()** generates a 48-bit random number, then returns its high-order 32 bits in the form of a **long**. The value returned is (or should be) uniformly distributed throughout the range of -2^31 through 2^31.

See Also

libc, srand48()

ms — Technical Information

Manuscript macro package
nroff -ms *file ...*

The **nroff** macro package **ms** formats manuscripts. The tutorial on **nroff** describes the **ms** macros in detail.

ms includes the following macros:

.AB	Begin the abstract portion of a document's title page.
.AE	End the abstract
.AI	Indicate author's institution on a document's title page.
.AU	Name the author on the title page of a document.
.B	Boldface font: set the following argument in boldface. If the argument is longer than one word, it must be enclosed in quotation marks. Anything on the line after the argument is thrown away.
.BD	Block-centered display. Take a portion of text; do not adjust it or break it between two lines, but center it as a whole.
.BT	Bottom title. This controls the printing of the footer title, should you want one. It uses three strings, all or any of which can be defined by the user: **LF**, for left-hand portion; **CF**, for center portion; and **RF**, for right-hand portion. **CF** has the default definition of printing the page number; the other two strings are undefined.
.CD	Centered display. Center individually every line within a display.
.DA	Set the date.
.DE	Mark the end of a display. Do *not* use after the macros **.LD**, **.CD**, or **.RD**.
.DS	Mark the beginning of a display. Do *not* use for displays longer than one page.
.FE	Mark the end of a footnote entry.
.FS	Mark the beginning of a footnote entry.
.I	Italic font. Used like **.B**, above.
.ID	Indent a display 1/2 inch before printing.
.IP	Indent a paragraph of text before printing. This macro can take two arguments: argument 1 is used as a **tag** that is printed to the left of the first line of the paragraph; argument 2 indicates how far to indent the paragraph, in characters (the default is five characters, or 1/2 inch).
.KE	Indicate the end of a *keep*, or a portion of text that must not be broken between two pages.
.KF	Start floating keep.
.KS	Indicate the beginning of a *keep*.
.LD	Set a display flush left; used with displays that are longer than one page.
.NH	Set a numbered heading. This macro takes one argument: the *depth* of numbering. For example, a '4' here would yield a number of the format "1.1.1.1". No number higher than five is accepted here. The following line gives the text of the heading.
.PP	Begin a new paragraph.
.QE	Mark the end of a quoted paragraph.
.QP	Quoted paragraph. Used like **.IP**, above.
.QS	Mark the beginning of quoted text; text is indented by five characters (1/2 inch).
.R	Roman font. Used like **.B**, above.
.RE	Mark the end of a relative indentation.
.RS	Mark the beginning of a relative indentation. A relative indentation is a block of text that is indented five characters (1/2 inch) more than the text before it.
.SH	Subheading. One line of space is inserted, and the following line of text is set boldface and flush left.
.TA	Set tabs, in characters.
.TL	Title: format the title entry on the cover page of a document.

Files

/usr/lib/tmac.s

See Also

man, nroff, troff, Using COHERENT *Introduction to nroff, Text Processing Language*, tutorial

MS-DOS — Technical Information

That other operating system

MS-DOS is the native operating system of the IBM-AT and compatible computers. As such, it needs no introduction to most users. Many customers have asked, however, how MS-DOS and COHERENT compare in terms of their capabilities; and many have also asked for a chart that maps familiar MS-DOS commands to their COHERENT equivalents. This article attempts to fulfill these requests.

MS-DOS vs. COHERENT

MS-DOS differs significantly from COHERENT in practically every aspect of its design. For example, its file system is incompatible with COHERENT; its shell **command.com** differs significantly from COHERENT's suite of shells; the manner in which it loads and executes a program differs completely from COHERENT's.

The most noticeable difference in design, however, is that MS-DOS is a single-user, single-process operating system, whereas COHERENT is a multi-user, multi-tasking operating system.

Single-user means that only one user can use MS-DOS at any given time: whoever sits at the keyboard "owns" the machine and all its facilities. *Multi-user* means, of course, that more than one user can use COHERENT at any given time, via terminals or modems plugged into the computer's serial ports. The number of users who can use your COHERENT system at once is limited only by your computer's speed, available memory, and by the number of serial ports that can be plugged into your computer.

Single-tasking means that MS-DOS can do only one task at a time: it loads a program into memory, runs it to completion, then awaits your request to execute another program. *Multi-tasking* means that COHERENT can execute more than one program at a time.

To grasp how multi-tasking can simplify some work, consider the task of formatting floppy disks. Under MS-DOS, you pop the floppy disk into the drive, invoke the MS-DOS program **format**, answer its queries, then go get a cup of coffee while the machine grinds away. Formatting a box of high-density floppy disks ties up your machine for the better part of an hour, which is largely wasted time for you. Under COHERENT, however, you can format a floppy disk in the *background* — that is, you can tell COHERENT to execute the disk-format program unsupervised, and let you work with another program. For example, if you wish to low-level format a 5.25-inch, high-density floppy disk in drive 0 (that is, drive A), use the following command:

```
	/etc/fdformat -v /dev/fha0 &
```

Try it. You'll notice that the COHERENT prompt returns immediately: while COHERENT is formatting your disk for you, you can edit a file, play a video game, dial out to a remote system, or even format a second disk in your machine's B drive (should you have one).

Multi-tasking also means that you can program COHERENT to execute programs untended, even while you are away from your machine. The UUCP system is a good example of this feature. UUCP lets you exchange mail and files with remote systems via modem; once the system is set up, it runs automatically, without requiring that you sit at the keyboard to run it.

This discussion only gives you a taste of the advantages COHERENT enjoys over an obsolete system like MS-DOS. The following documents contain information that MS-DOS users will find helpful:

- The tutorial *Using the COHERENT System* introduces COHERENT to new users. If you are new to COHERENT and have not yet read this tutorial, you should do so before you continue any farther.
- The Lexicon articles **floppy disks** and **hard disk** discuss the in's and out's of using mass-storage device with COHERENT. The article **floppy disks** in particular discusses in detail all the steps required to format and manipulate MS-DOS-style floppy disks under COHERENT.
- The Lexicon articles **modem**, **printer**, and **terminal** discussion how to connect these devices to COHERENT, and introduce the set of commands with which you can manipulate them under COHERENT.

- The Lexicon article **execution** describes in detail how COHERENT loads and executes a program. This article is aimed at the technically knowledgeable, but neophytes may find parts of it helpful.
- The Lexicon article **commands** summarizes all commands available under the COHERENT system. This article will help you grasp the scope of COHERENT's suite of commands, and will help you explore them systematically.
- The following Lexicon articles describe COHERENT commands for manipulating MS-DOS files and disks:

doscp	Copy files to/from an MS-DOS file system.
doscat	Concatenate a file on an MS-DOS file system.
doscp	Copy a file to/from an MS-DOS file system.
doscpdir	Copy directories to/from an MS-DOS file system.
dosdel	Delete files from an MS-DOS file system.
dosdir	Show the contents of an MS-DOS directory.
dosformat	Write an MS-DOS file system onto a floppy disk.
doslabel	Label an MS-DOS floppy disk. The MS-DOS file system can reside on a floppy disk or an MS-DOS portion of a hard disk.
dosls	List contents of an MS-DOS file system.
dosmkdir	Create a directory on an MS-DOS file system.
dosrm	Remove a file on an MS-DOS file system.
dosrmdir	Remove a directory from an MS-DOS file system.

COHERENT Equivalents to MS-DOS Commands

The following table lists the most commonly used MS-DOS commands, and gives COHERENT equivalents.

Note that often there is no single COHERENT command that equates to a given MS-DOS command. COHERENT often offers several alternatives, and you can select the one that best suits your needs. Every COHERENT command has its own article in the COHERENT Lexicon; look there first for details on how to use the command.

BACKUP

This command copies a directory's files to a formatted floppy disk to back them up. To do so under COHERENT, use the command:

```
find . -print | cpio -ocm > /dev/rfha0
```

Note that **cpio** requires a formatted, defect free floppy disk, however you do not need to create a filesystem on the floppy disk prior to using **cpio**.

Note that if you want COHERENT to prompt you before it backs up a file, use the command:

```
find . -print | cpio -ocmr > /dev/rfha0
```

See the article on the archiving command **cpio** for details on this command — especially important if you expect to retrieve your backed-up files.

Note, too, that the device **/dev/rfha0** corresponds to a 5.25-inch, high-density floppy disk in drive 0 (drive A). See the article **floppy disks** for a list of the devices that correspond to different sizes and configuration of floppy disks.

BREAK

Abort a command. Aborting a command under COHERENT varies, depending upon whether the command is running in the foreground or the background. The keystroke

```
<ctrl-c>
```

aborts most commands that are running in the foreground. To abort a command that is running in the background, you must use the **kill** command. See its Lexicon entry for details on how to use it.

CHDIR or **CD**

Change to another directory. To do so under COHERENT, use the command

```
cd dir
```

where *dir* is the directory to which you wish to go. The directories '.' and '..' are used by both COHERENT and MS-DOS; since MS-DOS "borrowed" its directory structure from UNIX (of which COHERENT is an implementation), the similarity should not be surprising.

Note that MS-DOS requires that before you can change to directory on another physical device or partition, you must first switch to that device by typing its name before you use the **chdir** command. COHERENT has no such restriction.

CHKDSK

Check the integrity of a file system. Under COHERENT, use the command:

```
/etc/fsck [option] [filesystem]
```

Read the Lexicon entry on **fsck** *before you attempt to run it!*

COMP Compare the contents of two files. To do so under COHERENT, use the following command to compare two binary files:

```
cmp [option] file1 file2
```

cmp displays the bytes which differ between the files.

To compare the contents of two text files, use the command:

```
diff [option] file1 file2
```

COPY Copy the contents of one file into another; create the target file if it does not already exist. Under COHERENT, say:

```
cp oldfilename newfilename
```

To copy a set of files into a directory without changing their names, use the following form of the command:

```
cp file1 ... fileN directory
```

DATE Reset the current date and time. Under COHERENT, use the command:

```
date yymmddhhmm.ss
```

Only the superuser can reset the system's date and time. When **date** is used without an argument, it prints the date and time on the standard output.

DIR Type the contents of a directory. Under COHERENT, use the command:

```
ls -l
```

DIR/W List a directory's contents in columnar form. Under COHERENT, use either the command:

```
lc
```

or the command:

```
ls -C
```

DISKCOPY

Copy one floppy disk track-by-track to another floppy disk. COHERENT has no exact equivalent to this command; however, you can copy the contents of one disk to another by using the following set of commands.

First, place a write-protect tab on your source disk; insert the disk into drive 0 (drive A), then type the following command:

```
dd if=/dev/fha0 of=/tmp/filename
```

This copies the contents of the 5.25-inch, high-density floppy disk in drive 0 into file **/tmp/filename**. For a table of devices that correspond to other sizes and configurations of floppy disks, see the Lexicon article **floppy disks**.

Second, insert formatted destination diskette into drive 0, and then type the command:

```
dd if=/tmp/filename of=/dev/fha0
```

This command copies the files in directory **/tmp/filename** onto the target floppy disk. Note that the target disk must be formatted before it can receive files; see the Lexicon article **floppy disks** for information on how to do this.

EDLIN Perform simple-minded editing of text files. Under COHERENT, the **ed** editor performs line editing, but is much more sophisticated than **edlin**. COHERENT also includes the **vi** and MicroEMACS screen editors, which are more useful still.

ERASE or **DEL**
Remove a file or a directory. To erase a file, use the command:

```
rm file1 [... fileN ]
```

To erase a directory, use the command:

```
rmdir directory
```

To erase a directory and all files and directories below it, use the command:

```
rm -r directory
```

FIND Find a pattern within a text file. Under COHERENT, use the command:

```
egrep [option] pattern [file ...]
```

egrep is an extremely useful command; see its Lexicon entry for details on how to use it.

FORMAT
Format a floppy disk. To format a floppy disk for MS-DOS, use the command **dosformat**. To format a floppy disk for COHERENT, use the command **fdformat**. For details, see the respective Lexicon entries for these commands. Under COHERENT, use the command

MEM Find how much space is left free on your hard disk. Under COHERENT, say:

```
df [options]
```

See the Lexicon entry on **df** for details.

MKDIR Create a new directory. Under COHERENT:

```
mkdir directory ...
```

MODE Set parameters for terminals and ports. Under COHERENT, use the command **stty**. This command comes with many options; see its Lexicon entry for details. The default speeds of all ports and terminals reside in file **/etc/ttys**. The superuser can use a text editor to edit this file to change any or all default settings.

MORE Display text a screenful at a time. Under COHERENT, use the commands **more** or **scat**.

PRINT Print files via a serial port. To print a file on a dot-matrix printer, use the command:

```
lpr file1 [ ... fileN ]
```

To print a file on a Hewlett-Packard LaserJet printer, use the command

```
hpr file1 [ ... fileN ]
```

Note that before these commands can be used, the appropriate devices must be linked to your system. See the Lexicon article on **printer** for details.

Note, too, that COHERENT uses a spooling system to manage the printing of files; thus, attempting to print a non-existent file will not hang the system.

PROMPT
Change the **command.com** prompt. The COHERENT shells store the prompt format within the environmental variable **PS1**. This variable is usually defined in each user's **.profile** file; this file holds commands that are executed whenever the user logs in. To change the definition of your prompt, edit **.profile** to define **PS1** to suit your preference, then log in again.

Note that the information that can be embedded within the prompt varies between the Bourne and Korn shells. See the Lexicon articles **sh** and **ksh** for details on those shells and their prompts.

RENAME
Rename a file. Under COHERENT, use the command:

```
mv  oldfile newfile
```

mv can also be used to move files from one directory or file system to another.

RESTORE
Restore a file saved with the **BACKUP** command. Under COHERENT, insert the floppy disk upon which the **cpio** utility saved its backup archive; then type the command:

```
cpio -icv < /dev/rfha0
```

Note that this command assumes you are using **/dev/rfha0**, which describes a 5.25-inch, high-density floppy disk in drive 0 (drive A). For a table of devices that correspond to other sizes and configurations of floppy disks, see the Lexicon article **floppy disks**.

TREE List all directories on a file system. Under COHERENT, use the command:

```
find / -type d | more
```

To list all files and directories that are subordinate to the current directory, use the command:

```
find . | more
```

The COHERENT command **ls -lR** also lists a directory tree, in a somewhat different output format.

MS-DOS 6.0 and COHERENT

Release 6.0 of MS-DOS offers a feature of dynamic file compression that creates some difficulties for machines that have both COHERENT and MS-DOS on their systems.

To begin, MS-DOS 6.0 assumes that it is the only operating system on your computer. When you install MS-DOS 6.0, by default it overwrites the COHERENT master boot block. If at all possible, you should install MS-DOS 6.0 onto your system first, then install COHERENT so that its Master Bootstrap is in control of your machine.

Second, MS-DOS 6.0 offers a compression utility called **dblspace**, which compresses MS-DOS file systems on the fly. The COHERENT **dos** commands do not understand compressed MS-DOS file systems created by the MS-DOS 6.0 utility **dblspace** or by such programs as **Stacker**. If you are running MS-DOS 6.0 with file compression, you must copy files to an uncompressed file system (for example, to an uncompressed floppy disk or to the uncompressed host for a compressed file system) to make them accessible to the COHERENT **dos** commands.

See Also

COHERENT, doscat, doscp, doscpdir, dosdel, dosdir, dosformat, doslabel, dosls, dosmkdir, dosrmdir, floppy disks, hard disk, modem, printer, terminal, Using COHERENT

msg — Kernel Module

Kernel module for messages

The kernel module **msg** enables System V-style messages. It is called a *kernel module* because you can link it into your kernel or exclude it, as you wish, just like a device driver; yet it is not a true device devicer because it does not perform I/O with a peripheral device.

See Also

device drivers, kernel, msgctl()

msg — Command

Send a brief message to other users
msg *user*
message

The command **msg** prints the one-line *message* on the screen of *user*.

The message is sent as soon as you type **<return>** on the *message* line. If *user* is not logged in or is not known to the system, **msg** prints an error message on your screen.

See Also

commands

msg.h — Header File

Definitions for message facility
#include <sys/msg.h>

msg.h defines the structures and constants used with the COHERENT message facility.

See Also

header files, msgget()

msgctl() — General Function (libc)

Message control operations
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(*id, command, buffer***)**
int *id***; int** *command***; struct msqid_ds ****buffer***;**

The function **msgctl()** controls the COHERENT's system's messaging facility. This facility permits processes to pass messages from one another.

Each message queue is controlled by a structure of type **msqid_ds**, which is defined in header file **<sys/msg.h>**. This structure points to the first and last messages in the queue, gives the size of the queue and the number of messages in the queue, and names who can manipulate it and how. The messages themselves consist of a linked list of structures of type **msg**, which is also defined in **msg.h**. When the function **msgget()** creates a message queue, it assigns to that queue an identification number and returns that number to the calling process. For details on this process, see the Lexicon entry for **msgget()**.

id identifies the message queue to be manipulated. This value must have been returned by a call to **msgget()**.

command names the operation that you want **msgctl()** to perform. **msgctl()** recognizes the following *commands*:

IPC_STAT	Copy the message-queue structure identified by *id* into the structure pointed to by *buffer*. This command lets you gather information about a message queue without actually manipulating the queue.
IPC_SET	This command sets permissions for this queue. It does so by copying fields **msg_perm.uid**, **msg_perm.gid**, **msg_perm.mode** (low nine bits only), and **msg_qbytes** from the message-queue structure point to by *buffer* to structure identified by *id*. Only the superuser **root** and the user who owns the process that created structure *id* can execute this *command*. Note that only the superuser can raise the value of field **msg_qbytes**, which gives the size of space occupied by the queue, in bytes.
IPC_RMID	Remove the structure identified by *id*, and destroy its queue. Only the superuser **root** and the user who owns the process that created structure *id* do this.

If any of the following conditions occur, **msgctl()** returns -1 and sets **error** to the value in parentheses:

- *id* is not a valid message-queue identifier (**EINVAL**).
- *command* is not a valid command (**EINVAL**).
- *command* equals **IPC_STAT**, but the owner of the calling process lacks permission to execute this command (**EACCES**).
- *command* equals **IPC_RMID** or **IPC_SET**, but the owner of the calling process lacks permission to execute the command (**EPERM**).
- A process owned by someone other than the superuser **root** attempted to increase field **msg_qbytes** (**EPERM**).
- *buffer* points to an illegal address (**EFAULT**).

If all went well, **msgctl()** returns zero.

Example

For an example of this function, see the Lexicon entry for **msgget()**.

Files

/usr/include/sys/ipc.h
/usr/include/sys/msg.h

See Also

libc, msgget(), msgrcv(), msgsnd()

Notes

For information on other methods of interprocess communication, see the Lexicon entries for **semctl()**, **shmctl()**, and **libsocket**.

msgget() — General Function (libc)

Create or get a message queue
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
msgget(*key***,** *flag***)**
key_t *key***; int** *flag;*

The function **msgget()** gets or creates a message queue. If necessary, it can create a message queue and its control structure, and link them to the identifier *key*.

key is an identifier that your application generates to identify its message queues. To guarantee that each key is unique, you should use the function call **ftok()** to generate keys.

When it creates a message queue, **msgget()** also creates a copy of structure **msqid_ds**, which the header file **<sys/msg.h>** defines as follows:

```
struct msqid_ds {
	struct ipc_perm msg_perm;		/* operation permission struct */
	struct msg *msg_first;		/* ptr to first message on queue */
	struct msg *msg_last;		/* ptr to last message on queue */
	unsigned short msg_cbytes;		/* current # bytes on queue */
	unsigned short msg_qnum;		/* # of messages on queue */
	unsigned short msg_qbytes;		/* max # of bytes on queue */
	unsigned short msg_lspid;		/* pid of last msgsnd() */
	unsigned short msg_lrpid;		/* pid of last msgrcv() */
	time_t msg_stime;			/* last msgsnd() time */
	time_t msg_rtime;			/* last msgrcv() time */
	time_t msg_ctime;			/* last change() time */
};
```

The messages themselves consist of a linked list of structures of type **msg**. Fields **msg_first** and **msg_last** point to, respectively, the first and last messages in the list. Header file **<sys/msg.h>** defines structure **msg** as follows:

```
struct msg {
	struct msg *msg_next;		/* pointer to next message on queue */
	long msg_type;			/* message type */
	short msg_ts;			/* message text size */
	short msg_spot;			/* message text map address */
};
```

Field **msg_perm** is a structure of type **ipc_perm**, which header file **<sys/ipc.h>** defines as follows:

```
struct ipc_perm {
	unsigned short uid;		/* owner's user id */
	unsigned short gid;		/* owner's group id */
	unsigned short cuid;		/* creator's user id */
	unsigned short cgid;		/* creator's group id */
	unsigned short mode;		/* access modes */
	unsigned short seq;		/* slot usage sequence number */
	key_t key;			/* key */
};
```

msgget() initializes **msqid_ds** as follows:

- It sets the fields **msg_perm.cuid**, **msg_perm.uid**, **msg_perm.cgid**, and **msg_perm.gid** to, respectively, the effective user ID and effective group ID of the calling process.
- It sets the low-order nine bits of **msg_perm.mode** to the low-order nine bits of *flag*. These nine bits define access permissions: the top three bits give the owner's access permissions (read, write, execute), the middle three bits the owning group's access permissions, and the low three bits access permissions for others.
- It sets **msg_ctime** is set to the current time.
- It sets **msg_qbytes** to the value of kernel variable **NMSQB**, which sets the maximum number of bytes available to the message queue.

If any of the following error conditions is true, **msgget()** returns -1 and sets **errno** to the value within parentheses:

- *key* already has a message queue, but the owner of the process that called **msgget()** does not have permission to read it (**EACCES**).
- *key* does not have a message queue associated with it, but *flag* is does not request that one be created (i.e., *flag* **& IPC_CREAT** is false) (**ENOENT**).
- *flag* requests that **msgget()** create a message queue, but the system's maximum number of message queues (as set by the kernel variable **NMSQID**) already exists (**ENOSPC**).
- *key* already has a message queue, but *flag* requests that a queue be created exclusively (i.e., (*flag* **& IPC_CREAT) && (***flag* **& IPC_EXCL**) is true) (**EEXIST**).

If all goes well **msgget()** returns the message-queue identifier, which is always a non-negative integer. Otherwise, it returns -1 and sets **errno** to an appropriate value.

Example

The following program, **samplemsg.c**, gives an example of the COHERENT message facility. One server process accepts user keyboard input, and sends it client 1 if the first character is an upper-case letter, or to client 2 if the first character is not an upper-case letter.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/signal.h>
#include <sys/types.h>
#include <sys/wait.h>

/* Maximum size of messages in this example.
 * The default maximum size is 2048. */
#define     MAX_MSG_SIZE 80

/* template for a message */
struct my_msg {
      long mtype;
      unsigned char mtext[MAX_MSG_SIZE];
};

struct my_msg sndmsg; /* message we will send */
struct my_msg rcvmsg; /* message we will receive */

key_t key;             /* key for getting our message queue */
int id;                /* message queue id returned by msgget() */
long msgtype;          /* type of the message */

main()
{
      /* Generate unigue key */
      if ((key = ftok("./samplemsg", 'A')) == -1)
            fprint (stderr, "samplemsg does not exist.\n");
            exit(EXIT_FAILURE);
      }
      /* get our message queue, abort on error */
      if( -1 == (id = msgget(key, IPC_CREAT|0660))){
            printf("Error obtaining message queue\n");
            exit(EXIT_FAILURE);
      }

      printf("To end this demonstration, type 'end'.\n"
            "Enter the message -> ");
      fflush(stdout);
      msgtype = 1;      /* 1st client receives messages of type 1 */

      /* fork() to produce our 1st client processes. */
      if (fork()) { /* we are parent process (server) */
            msgtype = 2; /* 2nd client receives messages of type 2 */
            /* fork() again to produce our 2nd client processes. */
            if (fork()) { /* we are parent process (server) */
                  send_messages(); /* server */
            } else
                  receive_messages(); /* second client */
      } else
            receive_messages(); /* 1st client */
      exit (EXIT_SUCCESS);
}

/* Get a message from user and send it to client or child processes */
send_messages()
{
      for (;;) {
            /* get our message to send */
            gets(sndmsg.mtext);

            /* if 'end' was entered, send message to BOTH clients,
             * as this is a flag for them to terminate themselves.
             * Otherwise, just send the message.
             */
            if (!strcmp(sndmsg.mtext,"end")) {
                  sndmsg.mtype = 1;
                  msgsnd(id, &sndmsg, strlen(sndmsg.mtext)+1, 0);
                  sndmsg.mtype = 2;
                  msgsnd(id, &sndmsg, strlen(sndmsg.mtext)+1, 0);
                  printf("Thank you. Bye.\n");
                  break;
            }

            /* Determine the type of message this will be.
             * if the first character is upper case letter,
             * then this is a type-1 message; otherwise,
             * this is a type-2 message.
             */
            if (isupper(sndmsg.mtext[0]))
                  sndmsg.mtype = 1L;
            else
                  sndmsg.mtype = 2L;

            if (msgsnd(id, &sndmsg, strlen(sndmsg.mtext)+1, 0) < 0) {
                  perror("send");
                  break;
            }
      }

      while (wait(NULL) > 0)  /* Wait for the children */
            ;
      msgctl(id, IPC_RMID,0); /* remove message queue */
      return;
}
```

```
/* receive_messages(). */
receive_messages()
{
      char clntbuf[20];

      sprintf(clntbuf, "Client %ld", msgtype);

      for (;;) {
            if (msgrcv(id, &rcvmsg, MAX_MSG_SIZE, msgtype, 0) < 0) {
                  perror(clntbuf);
                  exit(EXIT_FAILURE);
            }

            printf("%s received: '%s'\n", clntbuf, rcvmsg.mtext);
            if (!strcmp(rcvmsg.mtext,"end"))
                  break;
            printf("Enter next message -> ");
            fflush(stdout);
      }
      exit(EXIT_SUCCESS);
}
```

Files

/usr/include/sys/ipc.h
/usr/include/sys/msg.h

See Also

ftok(), ipcrm, ipcs, libc, libsocket, msgctl(), msgrcv(), msgsnd()

Notes

Prior to release 4.2, COHERENT implemented semaphores through the driver **msg**. In release 4.2, and subsequent releases, COHERENT has implemented semaphores as a set of functions that conform in large part to the UNIX System-V standard.

msgrcv() — General Function (libc)

Receive a message
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
msgrcv(*id***,** *buffer***,** *size***,** *type***,** *flag***)**
int *id***,** *size***,** *flag***; long** **buffer***; long** *type***;**

The function **msgrcv()** reads a message from the queue associated with identifier *id*, and writes it into the user-defined chunk of memory to which *buffer* points. The memory to which *buffer* points has a layout similar to a structure with the following members (if we pretend **mtext[]** is legal C):

```
struct msgbuf {
       long mtype;                      /* message type */
       char mtext[];                    /* message text */
};
```

mtype gives the message's type, as specified by the sending process. **mtext** gives the text of the message.

size gives the size of the message's text, in bytes. **msgrcv()** silently truncates the received message to *size* if it more than *size* bytes long and (*flag* **& MSG_NOERROR**) is true.

type gives the type of message being requested. **msgrcv** obeys the following rules when it reads the message queue:

- If *type* equals **0L**, it reads the first message in the queue.
- If *type* is greater than **0L**, it reads the first message of *type*.
- If *type* is less than **0L**, it reads the first message whose type is less than or equal to the absolute value of *type*.

If the message queue contains no message of the desired type, the behavior of **msgrcv()** is determined by the value of *flag*. If *flag* contains the value **IPC_NOWAIT** (i.e., *flag* **& IPC_NOWAIT** is true) then **msgrcv()** sets **errno** to **ENOMSG** and returns -1. If, however, *flag* does not contain **IPC_NOWAIT**, then **msgrcv()** suspends execution until one of the following occurs:

1. A message of the desired type appears on the queue.

2. *id* is removed from the system. **msgrcv()** sets **errno** to **EIDRM** and returns -1.

3. The calling process receives a signal. **msgrcv()** sets **errno** to **EINTR** and returns -1. The calling process then resumes execution in the manner by signal received. For information on what given signals mean, see the Lexicon entry for **signal()**.

msgrcv() also fails and returns no message if any of the following is true:

- *id* is not a valid message-queue identifier. **msgrcv** sets **errno** to **EINVAL**.
- The calling process lacks operation permission (**EACCES**).
- *size* is less than zero (**EINVAL**).
- The message's size is greater than *size* bytes long and (*flag* **& MSG_NOERROR**) is false (**E2BIG**).
- *buffer* points to an illegal address (**EFAULT**).

When **msgrcv()** has successfully received its message, it modifies the data structure associated with *id* in the following ways:

- It decrements field **msg_qnum** by one.
- It sets **msg_lrpid** to the identifier of the process that called **msgrcv()**.
- It sets **msg_rtime** to the current time.

When it completes successfully, **msgrcv()** returns the number of bytes written into the field **mtext** of the structure pointed to by *buffer*.

Example

For an example of this function, see the Lexicon entry for **msgget()**.

Files

/usr/include/sys/ipc.h
/usr/include/sys/msg.h

See Also

libc, msgctl(), msgget(), msgsnd()

msgs — Command

Read messages intended for all COHERENT users
msgs [*-q*] [*number*]

msgs selects and displays messages that are intended to be read by all COHERENT users. Messages are mailed to the login **msgs**. They should contain information meant to be read once by most users of the system.

The command **msgs** normally is in a user's **.profile**, so that it is executed every time he logs in. When invoked, it prompts the user with the identifier of the user who sent the message and the message's size. **msgs** then asks the user if he wishes to see the rest of the message. The user should reply with one of the following:

y	Display the message.
<return>	Display the message.
n	Skip this message and go to the next one.
-	Redisplay the last message.
q	Quit **msgs**.
number	Display message *number*; then continue.

If environmental variable **PAGER** is defined, **msgs** will "pipe" each message through the command specified in **PAGER**. For example, the **.profile** command line:

```
export PAGER="exec /bin/scat -1"
```

would invoke **/bin/scat** for each message with the command line argument **-1** (the digit one).

msgs writes into the file **$(HOME)/.msgsrc** the number of the next message the user will see when he invokes

msgs. **msgs** keeps all messages in the directory **/usr/msgs**; each message is named with a sequential number, which indicates its message number. The file **/usr/msgs/bounds** contains the low and high numbers of the messages in the directory; **msgs** determines whether a user has not read a message by comparing the information in **$(HOME)/.msgsrc** with that in **/usr/msgs/bounds**. If the contents of **/usr/msgs/bounds** are incorrect, the problem can be fixed by removing that file; **msgs** will create a new **bounds** file the next time it is run.

When the contents of a message are no longer needed, simply remove that message. Avoid removing the **bounds** file and the highest numbered message at the same time.

msgs accepts the following command-line options:

-q Query whether there are messages; print "There are new messages" if there are, and "No new messages" if not. The command **msgs -q** is often used in profile scripts.

number Start at message *number* rather than at the message recorded in **$(HOME)/.msgsrc**. If *number* is greater than zero, then start with that message; if *number* is less than zero, then begin *number* messages before the one recorded in **$(HOME)/.msgsrc**.

Files

/usr/spool/mail/msgs — Mail messages file
/usr/msgs/[1-9]* — Data base
/usr/msgs/bounds — File that contains message number bounds
$(HOME)/.msgsrc — Number of next message to be presented

See Also

commands, mail, PAGER, scat

msgsnd() — General Function (libc)

Send a message
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
msgsnd(*id*, *buffer*, *size*, *flag***)**
int *id*, *size*, *flag*; **long** **buffer*;

The function **msgsnd()** inserts a message into the queue associated with identifier *id*.

buffer points to a user-defined buffer that holds a code that defines the type of the message, and the text of the message. *buffer* can be described by a structure something like the following (if we pretend **mtext[]** is legal C):

```
struct msgbuf {
	long mtype;			/* message type */
	char mtext[];			/* message text */
};
```

Field **mtype** is a positive long integer that gives the type of message this is. Function **msgrcv()** examines this field to see if this message is of the type that it seeks. The text of the message immediately follows **mtype** in memory, for *size* bytes. *size* can range from zero to a maximum defined in the kernel variable **NMSC**.

If any of the following error conditions occurs, **msgsnd()** does not send the message, sets **errno** to the value given in parentheses, and returns -1:

- *id* is not a valid message queue identifier (**EINVAL**).
- The calling process does not have permission to manipulate this queue (**EACCES**).
- Field **mtype** in the structure pointed to by *buffer* is less than one (**EINVAL**).
- *size* is less than zero or greater than the system-imposed limit (**EINVAL**).
- *buffer* points to an illegal address (**EFAULT**).

Sending a message may exceed a system-defined limit. There are two such limits: one limits the size of a queue, and the other sets the total number of messages available to your system. The maximum size of this queue is given in the field **msg_qbytes** of the structure **msqid_ds** that controls that queue. If issuing a message *size* bytes long would push the total size of the queue's messages past the value of **msg_qbytes**, then an error occurs. Likewise, an error occurs if the system already holds the maximum maximum number of message available to it, as set by the kernel variable **NMSG**.

flag indicates how **msgsnd()** is to react to either of the above conditions. If *flag* is OR'd to include value **IPC_NOWAIT**, then **msgsnd()** reacts as it does with any other error: it does not send the message, it returns -1, and it sets **errno** to an appropriate value (in this case, **EAGAIN**). If, however, *flag* is *not* OR'd to include **IPC_NOWAIT**, then **msgsnd()** waits until any of the following happens:

1. The error condition resolves. In this case, **msgsnd()** sends the message and returns normally.
2. The message queue identified by *id* is removed from the system. In this case, **msgsnd()** does not send the message; it sets **errno** to **EIDRM**; and it returns -1.
3. The process that issued the call to **msgsnd()** receives a signal. In this case, **msgsnd()** does not send the message, sets **errno** to **EINTR**, and returns -1. The calling process then executes the action requested by the signal. For information on the behavior that each signal invokes, see the Lexicon entry for **signal()**.

msgsnd() successfully sends a message, returns zero and modifies the message queue in the following manner:

- It increments by one the value in field **msg_qnum**.
- It sets field **msg_lspid** to the process ID of the calling process.
- It sets **msg_stime** to the current time.

Example

For an example of this function, see the Lexicon entry for **msgget()**.

Files

/usr/include/sys/ipc.h
/usr/include/sys/msg.h

See Also

libc, msgctl(), msgget(), msgrcv()

msig.h — Header File

Machine-dependent signals
#include <signal.h>

The header file **msig.h** defines the machine-dependent signals that the COHERENT system uses to communicate with its processes. The header file **signal.h** declares constants for the machine-independent signals, and includes **msig.h**.

See Also

header files, signal.h

Notes

This header file is obsolete, and will be dropped from a future release of COHERENT. Its use is strongly discouraged.

msqrt() — Multiple-Precision Mathematics (libmp)

Compute square root of multiple-precision integer
#include <mprec.h>
void msqrt(*a, b, r***)**
mint **a*, **b*, **r*;

msqrt() sets the multiple-precision integer (or **mint**) pointed to by *b* to the integral portion of the positive square root of the **mint** pointed to by *a*. It sets the **mint** pointed to by *r* to the remainder. The value pointed to by *a* must not be negative. The result of the operation is defined by the condition

$a = b * b + r.$

See Also

libmp

msub() — Multiple-Precision Mathematics (libmp)

Subtract multiple-precision integers
#include <mprec.h>
void msub(*a, b, c***)**
mint **a*, **b*, **c***;**

msub() subtracts the multiple-precision integer (or **mint**) pointed to by *a* from the **mint** pointed to by *b*, and writes the result into the **mint** pointed to by *c*.

See Also

libmp

mtab.h — Header File

Currently mounted file systems
#include <mtab.h>

The file **/etc/mtab** contains an entry for each file system mounted by the command **mount**. This does not include the root file system, which is already mounted when the system boots.

Both the commands **mount** and **umount** use the following structure, defined in **mtab.h**. It contains the name of each special file mounted, the directory upon which it is mounted, and any flags passed to **mount** (such as read only).

```
#define     MNAMSIZ     32
struct      mtab {
      char  mt_name[MNAMSIZ];
      char  mt_special[MNAMSIZ];
      int   mt_flag;
};
```

Files

/etc/mtab

See Also

header files, mount, umount

mtioctl.h — Header File

Magnetic-tape I/O control
#include <sys/mtioctl.h>

mtioctl.h defines constants and structures used by routines that control magnetic-tape I/O.

See Also

header files

mtoi() — Multiple-Precision Mathematics (libmp)

Convert multiple-precision integer to integer
#include <mprec.h>
int mtoi(*a***)**
mint **a***;**

mtoi() returns an integer equal to the value of the multiple-precision integer (or **mint**) pointed to by *a*. The value pointed to by *a* should be in the range allowable for a signed integer.

See Also

libmp

mtos() — Multiple-Precision Mathematics (libmp)

Convert multiple-precision integer to string
#include <mprec.h>
char *mtos(*a***) mint** **a***;**

mtos() converts the multiple-precision integer (or **mint**) pointed to by *a* to a string. It returns a pointer to the string it creates. The string is allocated by **malloc()**, and may be freed by **free()**. The base of the string is set by the value of the external variable **obase**.

See Also

libmp

mtune — System Administration

Define tunable kernel variables
/etc/conf/mtune

File **mtune** defines all of the tunable variables within the kernel. These variables let you configure some aspects of your kernel, without having to modify the kernel's drivers or recompile the kernel.

Command **idmkcoh** reads this file when it builds a new kernel, and uses its contents to help patch the newly build kernel. A **mkdev** script (kept in a subdirectory of **/etc/conf**) also sets appropriate variables within this file, based on your answers to its questions.

Each line within **mtune** defines one tunable parameter. A line consists of four fields, as follows:

1. *Name*
This field names the parameter. It cannot exceed 20 characters.

2. *Minimum Value*
The legal minimum value of this parameter.

3. *Default Value*
The default value for this parameter. This value can be overridden by an entry in file **/etc/conf/stune**.

4. *Maximum Value*
The legal maximum value of this parameter.

Note that under UNIX System V, fields 2 and 3 are reversed. A line that begins with the pound sign '#' is a comment, and is ignored by **idmkcoh** when it builds a new kernel.

For details on the parameters that this file sets, read the comments within this file.

See Also

Administering COHERENT, device drivers, mdevice, sdevice, stune

Notes

mtune contains comments that describe the kernel variables that you can tune. If you wish to tune the kernel, you should read this file and modify it appropriately. The variables are documented in this file rather than in the COHERENT manual to ensure that you have exactly accurate information about the variables that reside in the version of the kernel on your system.

mtype() — General Function (libc)

Return symbolic machine type
#include <mtype.h>
char *mtype(*type***)**
int *type***;**

mtype() takes an integer machine *type* and returns the address of a string that contains the symbolic name of the machine. The header file **mtype.h** defines the possible machine types. For example,

```
mtype(M_PDP11)
```

returns the address of the string **PDP-11**.

Files

<mtype.h>

See Also

libc

Diagnostics

mtype() returns NULL to indicate that it doesn't recognize the type of machine requested.

mtype.h — Header File

List processor code numbers
#include <mtype.h>

The header file **mtype.h** assigns a code number to each of the processors supported by Mark Williams C compilers and operating systems. These include the Intel i8086, i8088, i80186, i80286, and i80386; the Zilog Z8001 and Z8002; the DEC PDP-11 and VAX; the IBM 370, and the Motorola 68000.

See Also

header file

mult() — Multiple-Precision Mathematics (libmp)

Multiply multiple-precision integers
#include <mprec.h>
void mult(*a, b, c***)**
mint **a*, **b*, **c***;**

mult() multiplies the multiple-precision integers (or **mint**s) pointed to by *a* and *b*, and writes the product into the **mint** pointed to by *c*.

See Also

libmp

mv — Command

Rename files or directories
mv [-f] *oldfile* **[***newfile***]**
mv [-f] *file ... directory*

mv renames files. In the first form above, it changes the name of *oldfile* to *newfile*. If *newfile* already exists, **mv** replaces it with the file being moved; if not, **mv** creates it. If *newfile* is a directory, **mv** places *oldfile* under that directory.

In the second form, **mv** moves each *file* so that it resides under *directory*. If a file with the new name exists but is unwritable, **mv** will not delete it unless the **-f** option is specified.

mv will not copy directories between devices and will not remove directories that occupy the destination of the command.

Normally, **mv** creates a link to the old file with the new file and then removes the old file. If it cannot create the link (e.g., because the new file is on a different file system than the old), **mv** performs a copy and then removes the old file.

See Also

commands, cp, ln, mvdir

Notes

mv tests the validity of directory moves by means of search permission. The superuser always has search permission and thus can use **mv** incorrectly.

mvdir — Command

Rename a directory
/etc/mvdir *olddir newdir*

The COHERENT command **mvdir** renames directory *olddir* to *newdir*. Both can be path names.

For obvious reasons, *olddir* cannot be a subset of *newdir*. Both *olddir* and *newdir* must reside on the same file system.

See Also

commands, mv

Notes

mvdir is a link to **mv**.

mvfree() — Multiple-Precision Mathematics (libmp)

Free multiple-precision integer
#include <mprec.h>
void mvfree(*a***)**
mint ****a*;**

mvfree() frees the space allocated to an automatic multiple-precision integer (or **mint**). You should call **mvfree()** before exiting the function that uses the **mint**, or the storage used by the **val** field of the **mint** structure will never be reclaimed.

See Also

libmp

mwcbbs — Command

Download files from the Mark Williams bulletin board
mwcbbs [-cp] [-d*path***]** *directory*

The command **mwcbbs** lets you select one or more files from **mwcbbs**, the bulletin board maintained by Mark Williams Company. It displays the contents of **Contents** files that you download from the bulletin board, lets you select one or more files interactively, then constructs a **uucp** command and requests the files from Mark Williams. If all goes well, the files will be delivered to directory **/usr/spool/uucppublic** on your system. **mwcbbs**. In this way, you can obtain the latest versions of COHERENT software, sources for public-domain software that has been ported to COHERENT, and exchange mail with MWC developers and support personnel.

Options

mwcbbs recognizes the following options:

-c Force **uucp** to telephone the Mark Williams bulletin board when you exit from **mwcbbs**.

-d*path* Use *path* in place of the default receive path.

-p Print the **Contents** file. You can print all entries in a **Contents** file, or entries newer than a specified date.

mwcbbs looks for the file **.mwcbbs** in the current directory. This file contains the interface variables **DOWNPATH** and **DATAPATH**. The former names names the directory into which **uucp** is to write the requested files; and the latter names the directory where you keep data files. For example:

```
DOWNPATH=/usr/spool/uucppublic/
DATAPATH=/usr/lib/mwcbbs
```

Please note that path names are limited to 45 characters.

When you invoke **mwcbbs**, it displays a menu with the following items:

0. Contents.down
List public-domain software and shareware available for COHERENT release 3.*N* (COHERENT 286).

1. Contents.32bit
List public-domain software and shareware available for COHERENT release 4.*N* (COHERENT 386).

2. Contents.news
List news items and other informative postings from MWC.

3. Contents.UPD
List updates to COHERENT.

4. Maillist
List the mail sites available through **mwcbbs**.

5. Net_Maps
Show available network maps of world-wide UNIX sites.

6. Quit Exit from **mwcbbs**.

Downloading Files

If you select items 0 through 3 from the main menu, **mwcbbs** displays the names of files, 100 at a time. These names are read from a **Contents** file that is stored in a directory you name either with the option **-d** or the variable **DATAPATH**.

You can select one or more of these files for downloading to your system. Note that when you invoke **mwcbbs** for the first time, the only files displayed are those of the **Contents** files themselves; you must download them first, before you can begin to download other files. This is because the **Contents** are continually being updated, and also to test your UUCP with the Mark Williams bulletin board before you attempt to download a large number of source files.

To select a file for downloading, use the arrow keys to move the cursor to that file (or use the **vi** cursor-movement keys **h**, **j**, **k**, and **l**). **mwcbbs** lets you enter any of the following commands:

-s Select highlighted file name for more information or downloading. Pressing (¢) also selects the file.

-n Go to next screen (if there are more than 100 files).

-p Go to the previous screen.

-q Quit **mwcbbs**.

When you select a highlighted file, **mwcbbs** displays the following information about it:

- A summary of the file.
- The date it was added or last updated
- Other files that are required for compilation or use of selected file name.
- Other miscellaneous notes that may be of interest.
- The system commands to be generated to download the selected file from the Mark Williams bulletin board.

If a file is more than 50,000 bytes long, **mwcbbs** downloads it in parts. When a file is to be received in parts you must concatenate the parts into one file, which should be given the name of the file you selected.

Lists and Networks

Item 4 on the main menu (**Maillist**) gives you information about electronic mail sites throughout the United States. When you select this option, **mwcbbs** displays the names of the 50 states. When you select a state, as with the file lists, **mwcbbs** displays information about the mail sites in that state.

Item 5 (**Net_Maps**) gives you information about networks. When you select this option, **mwcbbs** displays a menu with the following items:

0. Net_Maps.WORLD
Network maps of UNIX sites across the world.

1. Net_Maps.USA
Network maps of UNIX sites in the United States, by area code.

2. Net_Maps.CAN
Network maps of UNIX sites in Canada, by area code.

3. Quit Return to main menu.

Options 0 through 2 display maps of available networks. You can select a map interactively, as with the file options.

Printing a Contents File

When you invoke **mwcbbs** with its **-p** option, it lists the four **Contents** files. When you select one, **mwcbbs** asks you to enter a search date. If you enter a search date, **mwcbbs** prints only those entries that are dated later than that date. If you do not enter a date, it prints every entry in the **Contents** file entries.

Entries are printed to the file **mwcbbs**. When the entries have been printed, **mwcbbs** automatically exits to the shell.

See Also

commands, UUCP
COHERENT Tutorial: **UUCP, Remote Communications Utility**

Notes

mwcbbs does not work correctly until you have correctly configured UUCP to contact the Mark Williams bulletin board. For details on how to do so, see the tutorial *UUCP, Remote Communications Utility* in the front of the COHERENT manual.

The charges for downloading a large set of files via a long-distance telephone call can be quite heavy. Much depends upon the speed of your modem and the time you place your call. *Caveat utilitor!*