ChorusOS™ man pages section 1M: System Management Utilities

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part Number 806-3324
October 4, 1999

Contents

PREFACE

Overview

A man page is provided for both the naive user, and sophisticated user who is familiar with the ChorusOS operating system and is in need of on-line information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following is a list of sections in the man pages and the information it references:

- Section 1CC: User Utilities; Target Utilites
- Section 1M: System Management Utilities
- Section 2DL: System Calls; Data Link Services
- Section 2K: System Calls; Kernel Services
- Section 2MON: System Calls; Monitoring Services
- Section 2POSIX: System Calls; POSIX System Calls
- Section 2SEG: System Calls; Virtual Memory Segment Services
- Section 3FTPD: Libraries; FTP Daemon
- Section 3M: Libraries; Mathematical Libraries
- Section 3POSIX: Libraries; POSIX Library Functions
- Section 3RPC: Libraries; RPC Services
- Section 3STDC: Libraries; Standard C Library Functions
- Section 3TELD: Libraries; Telnet Services
- Section 4CC: Files

- Section 5FEA: ChorusOS Features
- Section 7P: Protocols
- Section 7S: Services
- Section 9DDI: Device Driver Interfaces
- Section 9DKI: Driver to Kernel Interface
- Section 9DRV: Driver Implementations

ChorusOS Man pages are grouped in Reference Manuals, with one reference manual per section

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and **man**(1) for more information about man pages in general.

NAME — This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . . Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, ‘ "`filename` . . ." .

| Separator. Only one of the arguments separated by this character can be specified at time.

	{ } Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.
PROTOCOL	This section occurs only in subsection 3R to indicate the protocol description file.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES.. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.
OPTIONS	This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output - standard output, standard error, or output files - generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each

condition is described in a separate paragraph under the error code.

USAGE

This section is provided as a guidance on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality:

Commands
Modifiers
Variables
Expressions
Input Grammar

EXAMPLES

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%` or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.

ENVIRONMENT VARIABLES

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

EXIT STATUS

This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion and values other than zero for various error conditions.

FILES

This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

SEE ALSO

This section lists references to other man pages, in-house documentation and outside publications.

DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.
BUGS	This section describes known bugs and wherever possible, suggests workarounds.

CHAPTER

System Management Utilities

NAME

arp – address resolution display and control

SYNOPSIS

arp –a hostname arp

arp –d hostname

arp –s hostname ether_addr [temp] [pub] [trail]

arp –f filename

DESCRIPTION

The `arp` program displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol *arp(7P)*. If no flags are set, the program displays the current ARP entry for `hostname`. The host may be specified by name or by number, using Internet dot notation.

Available options:

`–a` The program displays all of the current ARP entries.

`–d` A super-user may delete an entry for the host called `hostname` with the *-d* flag.

`–s` *hostname ether_addr* Create an ARP entry for the host called `hostname` with the Ethernet address *ether_addr*. The Ethernet address is given as six hex bytes separated by colons (":"). The entry will be permanent unless the word *temp* is used in the command. If the word *pub* is given, the entry will be "published"; this system will act as an ARP server, responding to requests for `hostname` even though the host address is not its own. The word *trail* indicates that trailer encapsulations may be sent to this host.

`–f` *filename* Causes the file *filename* to be read and multiple entries to be set in the ARP tables. Entries in the file should be of the form:

hostname ether_addr [temp] [pub] [trail]

with argument meanings as given above.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO **inet**(3STDC), **arp**(7P), **ifconfig**(1M)

NAME

chat – automated conversational script with a modem

SYNOPSIS

chat [*options*] *script*

DESCRIPTION

`chat` defines a conversational exchange between the computer and the modem. Its primary purpose is to establish the connection between the Point-to-Point Protocol daemon (`pppd`) and a remote `pppd` process.

OPTIONS

`chat` takes the following options:

`–e`
Start with the echo option turned on. Echoing may also be turned on or off at specific points in the chat script by using the `ECHO` keyword. When echoing is enabled, all output from the modem is echoed to `stderr`.

`–f` *chat_file*
Read the chat script from *chat_file*. If you use this option, then you cannot use *script* parameters as well. The user must have read access to the file. Multiple lines are permitted in the file. Spaces and horizontal tab characters are used to separate the strings.

`–r` *report_file*
Set the file for output of the report strings. If you use the keyword `REPORT`, the resulting strings are written to this file. If this option is not used and you still use the keyword `REPORT`, report strings are written to `stderr`.

`–S`
Do not use **syslog()**. By default, error messages are sent to **syslog()**. Using `–S` prevents both log messages from `–v` and error messages from being sent to **syslog()**.

`–s`
Use `stderr`. All log messages from `–v` and all error messages are sent to `stderr`.

`–t` *timeout*
Set the timeout for the expected string to be received. If the string is not received within the time limit set by *timeout*, then the reply string is not sent. An alternate reply may be sent or the script will fail if no alternate reply string is available. Script failure causes `chat` to terminate with a non-zero error code.

–V	Request that the chat script be executed in stderr verbose mode. In stderr verbose mode `chat` logs the all text received from the modem, and the output strings sent to the modem, to `stderr`. `stderr` is usually the local console where `chat` or `pppd` is running.
–v	Request that the chat script be executed in verbose mode. In verbose mode `chat` logs the execution state of the chat script as well as all text received from the modem and the output strings sent to the modem. The default is to log through **syslog()**. The logging method may be altered using the –S and –s options. Logging is done to the `local2` facility at the `info` level for verbose tracking and the `err` level for errors only. Huh?
script	If the script is not specified in a file with the –f option, then the script is included as parameters to `chat`.

CHAT SCRIPT

The chat script defines the communications.

A script consists of one or more *expect-send* pairs of strings, separated by spaces, with an optional *subexpect-subsend* string pair, separated by a dash, as in the following example:

```
ogin:-BREAK-ogin: ppp ssword: hello2u2
```

This line indicates that `chat` should expect the string `ogin:`. If it fails to receive a login prompt within the time interval allotted, it is to send a break sequence to the remote process and then expect the string `ogin:`. If the first `ogin:` is received then the break sequence is not generated.

Once `chat` receives the login prompt, it sends the string `ppp` and then expects the prompt `ssword:`. When `chat` receives the prompt for the password, it sends the password `hello2u2`.

A carriage return is normally sent following the reply string. It is not expected in the *expect* string unless it is specifically requested using the `\r` character sequence.

The expect sequence should contain only what is needed to identify the string. Since the script is normally stored in a file on disk, it should not contain variable information. It is not generally acceptable to look for time strings, network identification strings, or other variable pieces of data as an expect string.

A very simple script might look like this:

```
ogin: ppp ssword: hello2u2
```

In other words, expect ...`ogin:`, send `ppp`, expect ...`ssword:`, send `hello2u2`.

In actual practice, simple scripts are rare. At the very least, you should include *sub-expect* sequences to handle cases where the original string is not received. For example, consider the following script:

```
ogin:--ogin: ppp ssword: hello2u2
```

This would be a better script that the simple one shown above. This script would look for the same `login:` prompt. However, if the prompt were not received, a single return sequence is sent, and then the `chat` would look for `login:` again. If line noise obscures the first login prompt, then sending the empty line usually generates a login prompt again.

COMMENTS

Comments can be embedded in the chat script. A comment is a line starting with the # character in column 1. `chat` ignores comment lines. If a # is expected as the first character of the expect sequence, then you must quote the expect string. If you want to wait for a prompt that starts with the # character, then you must write something like this:

```
# Now wait for the prompt and send the logout string
'# ' logout
```

ABORT STRINGS

Many modems report the status of a call as a string. Such strings may be CONNECTED, NO CARRIER, or BUSY. It is often desirable to terminate the script if the modem fails to connect to the remote process. The difficulty is that a script does not know exactly which modem string it may receive. On one attempt, it may receive BUSY while the next time it may receive NO CARRIER. Such ABORT strings may be specified in the script using the ABORT sequence. An ABORT sequence is written in the script as in the following example:

```
ABORT BUSY ABORT 'NO CARRIER' '' ATZ OK ATDT5551212 CONNECT
```

This sequence causes chat to expect nothing. chat then sends the string ATZ. The expected response is the string OK. When it receives OK, chat sends the modem ATDT5551212, telling it to dial the telephone. The expected string is CONNECT. If the string CONNECT is received, then the remainder of the script is executed. However, if the modem reaches a busy telephone, then it sends the string BUSY, causing the string to match the abort character sequence. The script then fails because it finds a match to the abort string. The string NO CARRIER causes it to abort for the same reason. Either string may be received. Either string terminates the chat script.

CLR_ABORT STRINGS

CLR_ABORT makes it possible to clear previously set ABORT strings. ABORT strings are kept in an array of pre-determined size (size determined at compilation time). CLR_ABORT reclaims the space so that new strings can use it.

SAY STRINGS

SAY strings allow the script to send strings to the user at the terminal via stderr. If chat is being run by pppd, and if pppd is running as a daemon detached from its controlling terminal, then stderr is normally redirected to the file /etc/ppp/connect-errors.

SAY strings must be enclosed in single or double quotes. If carriage return and line feed characters are needed as output in the string, you must explicitly add them to the string.

SAY strings may be used to provide progress messages in sections of the script where you want to have ECHO OFF but still let the user know what is happening. An example is:

```
ABORT           BUSY
ECHO            OFF
SAY             "Dialing your ISP...\n"
```

```
''              ATDT5551212
TIMEOUT         120
SAY             "Waiting up to two minutes for connection...\n"
CONNECT         ''
SAY             "Connected. Now logging in...\n"
ogin:           account
ssword:         pass
SAY             "Logged in OK..."
```

The above example only presents the SAY strings to the user. All the details of the script remain hidden. For example, if the above script works, then the user sees:

```
Dialing your ISP...
Waiting up to two minutes for connection...
Connected. Now logging in...
Logged in OK...
```

REPORT STRINGS

REPORT strings are similar to ABORT strings. The difference is that the strings and all characters until the next control character, such as a carriage return, are written to a the report file.

REPORT strings may be used to isolate the transmission rate of the modem's connect string and return the value to the chat user. Analysis of the REPORT string occurs in conjunction with other string processing such as looking for the expect string. The use of the same string for a report and abort sequence is probably not very useful. However, it is possible.

REPORT strings do not change the completion code of the program.

REPORT strings may be specified in the script using the REPORT sequence. A REPORT sequence is written in the script as in the following example:

```
REPORT CONNECT ABORT BUSY '' ATDT5551212 CONNECT '' ogin: account
```

The above sequence expects nothing. It causes chat to send the modem ATDT5551212, telling it to dial the telephone. The expected string is CONNECT. If CONNECT is received, then the remainder of the script is executed. In addition, the program writes the string CONNECT and any characters which follow it, such as the connection rate, to the expect file.

CLR_REPORT STRINGS

CLR_REPORT makes it possible to clear previously set REPORT strings. REPORT strings are kept in an array of pre-determined size (size determined at compilation time). CLR_REPORT reclaims the space so that new strings can use it.

ECHO

The ECHO option controls whether the output from the modem is echoed to stderr. This option may be set using –e, but it can also be controlled using the ECHO keyword. The *expect-send* pair ECHO ON enables echoing, and ECHO OFF disables it. Using the ECHO keyword, you can select which parts of the conversation should be visible. For example, with the following script:

```
ABORT           BUSY
ABORT           'NO CARRIER'
OK\r\n          ATDT5551212
\r\n            \c
ECHO      ON
CONNECT   \c
ogin:     account
```

all output resulting from modem configuration and dialing remains hidden. However, starting with the CONNECT or BUSY message, everything is echoed.

HANGUP

The HANGUP option controls whether a modem hang up should be considered an error. This option is useful in scripts for dialing systems which hang up and call your system back. The HANGUP option can be ON or OFF.

When HANGUP is set to OFF and the modem hangs up, for example, after the first stage of logging in to a callback system, chat continues running the script, for example, expecting the incoming call and second stage login prompt. As soon as the incoming call is connected, you must use HANGUP ON to reinstate normal hang up signal behavior. Below is an example script:

```
ABORT                 BUSY
OK\r\n                ATDT5551212
\r\n                  \c
CONNECT               \c
'Callback login:'     call_back_ID
HANGUP                OFF
ABORT                 "Bad Login"
'Callback password:'  call_back_password
TIMEOUT               120
CONNECT               \c
HANGUP                ON
ABORT                 'NO CARRIER'
ogin:--BREAK--ogin:   real_account
```

TIMEOUT

The initial timeout value is 45 seconds. This may be changed using the –t option.

To change the timeout value for the next expect string, the following example may be used:

```
ATZ OK ATDT5551212 CONNECT TIMEOUT 10 ogin:--ogin: TIMEOUT 5 ssword: hello2u2
```

The above example changes the timeout to 10 seconds, during which the login prompt is expected. The timeout is then changed to five seconds, during which the password prompt is expected.

The timeout, once changed, remains in effect until it is changed again.

SENDING EOT

The special reply string EOT indicates that chat should send an EOT character to the remote process. The EOT character is normally the end-of-file character sequence. No return character is sent following the EOT. The EOT sequence may be embedded in the send string using the sequence ^D.

GENERATING BREAK

The special reply string BREAK causes a break condition to be sent. BREAK indicates a special signal on the transmitter. The normal processing on the receiver is to change the transmission rate. BREAK may be used to cycle through the available transmission rates on the remote until you are able to receive a valid login prompt. The break sequence may be embedded in to the send string using the \K sequence.

ESCAPE SEQUENCES

The expect and reply strings may contain escape sequences. All escape sequences are legal in the reply string. Many are legal in the expect string. Those that are not valid in the expect sequence are so indicated.

''	Expect or send a null string. If you send a null string then the return character is also sent. This sequence may be either a pair of apostrophe or quote characters.
\\	Expect or send a backslash character.
\b	Expect or send a backspace character.
\c	Suppress the newline at the end of the reply string. This is the only way to send a string without a trailing return character. \c must be at the end of the send string. For example, the sequence hello\c simply sends the characters h, e, l, l, o. (*Not valid in expect sequences.*)
\d	Delay for one second. chat uses threadDelay. (*Not valid in expect sequences.*)
ddd	Collapse the octal digits *ddd* to a single ASCII character, and send that character. (*Some characters are not valid in expect sequences.*)
\K	Insert a BREAK. (*Not valid in expect sequences.*)

`\N`	Send a null character. The same sequence may also be written as `\0`. (*Not valid in expect sequences.*)
`\n`	Send a newline or linefeed character. (*Not valid in expect sequences.*)
`\p`	Pause for one-tenth of a second. (*Not valid in expect sequences.*)
`\q`	Suppress writing the string to `syslogd`. The string `??????` is written to the log in its place. (*Not valid in expect sequences.*)
`\r`	Expect or send a carriage return.
`\s`	Expect or send a space character. This may be used when it is not desirable to quote the strings that contain spaces. For example, `'HI SCOTT'` and `HI\sSCOTT` are equivalent.
`\t`	Expect or send a tab character.
`^`*C*	Substitute the sequence with the control character represented by *C*. For example, the character DC1 (17) is shown as `^Q`. (*Some characters are not valid in expect sequences.*)

TERMINATION CODES

`chat` terminates with the following completion codes:

`0`	The normal termination of the program. This indicates that the script was executed without error to the normal conclusion.
`1`	One or more of the parameters are invalid, or an expect string was too large for the internal buffers. This indicates that the program was not properly executed.
`2`	An error occurred during the execution of the program. This may be due to a read or write operation failing for some reason, or to `chat` receiving a signal such as `SIGINT`.
`3`	A timeout event occurred when there was an expect string without a subsend string. This may

	mean that you did not program the script correctly for the condition or that some unexpected event has occurred and the expected string could not be found.
4	The first string marked as an ABORT condition occurred.
5	The second string marked as an ABORT condition occurred.
6	The third string marked as an ABORT condition occurred.
7	The fourth string marked as an ABORT condition occurred.
...	Other termination codes indicate the occurrence of strings marked as ABORT conditions.

Using the termination code, it is possible to determine which event terminated the script. It is possible to determine, for example, if the string received from the modem was BUSY as opposed to 'NO DIAL TONE'.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

pppd(1M), **slattach**(1M)

NAME

chorusNS, chorusNSsite, chorusNSinet – ChorusOS name servers

SYNOPSIS

chorusNSsite [user-sites-file] chorusNSinet

DESCRIPTION

The *chorusNS* utility is a family of ChorusOS Name Server daemons which answer local and remote site naming requests issued using *getsitebyname* (3CSAFE) or *getsitebyaddr* (3CSAFE). It inserts a CHORUS port into a static user group (see *grpAllocate* (2K)), continuously waits for messages on that port, and then replies to clients. The daemons can be started and killed using the arun and akill *C_INIT* (1M) commands.

The *chorusNSsite* searches sequentially from the beginning of the local CHORUS site data base until a matching site name or site address is found, or until end-of-file is reached. If the optional *user-sites-file* parameter is omitted, the */etc/sites* data base is used.

The *chorusNSinet* uses the *gethostbyaddr* (3CSAFE) and *gethostbyname* (3CSAFE) functions to retrieve information from an Internet Name Server. It is assumed that a mapping exists between the IP address set and the CHORUS site address set. In this case, the CHORUS symbolic site name is equal to the IP symbolic name. An inetNS daemon must be running somewhere on the network.

The *CHORUSNS* environment parameter may be used to reference the site where a *chorusNS* server is located if it is not local.

No more than one name server can be run on a given site.

FILES

/etc/sites

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

inetNS(1M) , **sites**(4CC)

NAME

chorusNS, chorusNSsite, chorusNSinet – ChorusOS name servers

SYNOPSIS

chorusNSsite [user-sites-file] chorusNSinet

DESCRIPTION

The *chorusNS* utility is a family of ChorusOS Name Server daemons which answer local and remote site naming requests issued using *getsitebyname* (3CSAFE) or *getsitebyaddr* (3CSAFE). It inserts a CHORUS port into a static user group (see *grpAllocate* (2K)), continuously waits for messages on that port, and then replies to clients. The daemons can be started and killed using the arun and akill *C_INIT* (1M) commands.

The *chorusNSsite* searches sequentially from the beginning of the local CHORUS site data base until a matching site name or site address is found, or until end-of-file is reached. If the optional *user-sites-file* parameter is omitted, the */etc/sites* data base is used.

The *chorusNSinet* uses the *gethostbyaddr* (3CSAFE) and *gethostbyname* (3CSAFE) functions to retrieve information from an Internet Name Server. It is assumed that a mapping exists between the IP address set and the CHORUS site address set. In this case, the CHORUS symbolic site name is equal to the IP symbolic name. An inetNS daemon must be running somewhere on the network.

The *CHORUSNS* environment parameter may be used to reference the site where a *chorusNS* server is located if it is not local.

No more than one name server can be run on a given site.

FILES

/etc/sites

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

inetNS(1M) , **sites**(4CC)

NAME

chorusNS, chorusNSsite, chorusNSinet – ChorusOS name servers

SYNOPSIS

chorusNSsite [user-sites-file] chorusNSinet

DESCRIPTION

The *chorusNS* utility is a family of ChorusOS Name Server daemons which answer local and remote site naming requests issued using *getsitebyname* (3CSAFE) or *getsitebyaddr* (3CSAFE). It inserts a CHORUS port into a static user group (see *grpAllocate* (2K)), continuously waits for messages on that port, and then replies to clients. The daemons can be started and killed using the arun and akill *C_INIT* (1M) commands.

The *chorusNSsite* searches sequentially from the beginning of the local CHORUS site data base until a matching site name or site address is found, or until end-of-file is reached. If the optional *user-sites-file* parameter is omitted, the */etc/sites* data base is used.

The *chorusNSinet* uses the *gethostbyaddr* (3CSAFE) and *gethostbyname* (3CSAFE) functions to retrieve information from an Internet Name Server. It is assumed that a mapping exists between the IP address set and the CHORUS site address set. In this case, the CHORUS symbolic site name is equal to the IP symbolic name. An inetNS daemon must be running somewhere on the network.

The *CHORUSNS* environment parameter may be used to reference the site where a *chorusNS* server is located if it is not local.

No more than one name server can be run on a given site.

FILES

/etc/sites

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

inetNS(1M) , **sites**(4CC)

NAME

C_INIT – initial ChorusOS actor and command interpreter

SYNOPSIS

`C_INIT` is not invoked directly by a user, but by the ChorusOS

DESCRIPTION

`C_INIT` is a supervisor `c_actor` loaded at system startup. It provides a command interpreter, enabling administration of the ChorusOS target system.

The command interpreter may be accessed in two ways, either from the host system using the **`rsh`**(1) protocol, or through a local console. `C_INIT` may be configured to support one or both of these methods. The `RSH` feature enables support for the `rshd` protocol within `C_INIT`. The `LOCAL_CONSOLE` feature enables support for the local console. Both can be set at the same time.

The `C_INIT` actor is also responsible for authentification of users who issue `C_INIT` commands. The ChorusOS system can be configured in secured mode such that remote host access is checked against the administration file, **`security`**(4CC). User credentials specified in the security administration file override default configuration values.

SYSTEM INITIALIZATION

Before the ChorusOS system loads `C_INIT`, it automatically mounts a pseudo root file system and creates a `/dev` special files directory, in which it creates a `console` node and into which it mounts the contents of the system image as a FAT 12 file system, in `/dev/archive`. This provides the `C_INIT` actor with access to the console and to the contents of the system image.

Next, `C_INIT` executes the commands in the file `/dev/archive/sysadm.ini`. The commands in this file generally pertain to special device file creation and network initialization. See **`sysadm.ini`**(4CC) for details. However, usually one of the commands in `sysadm.ini` mounts a new root file system for the target. For example:

`mount` *`host`*`:/export/work/ChorusOS/root /`

The new root file system supercedes the existing pseudo root file system. However, the contents of `/dev` and `/dev/archive` remain accessible.

`C_INIT` next attempts to execute `/etc/rc.chorus.`*`target.system.IP.address`* (**`rc.chorus`**(4CC)), or alternatively, `/etc/rc.chorus` if `/etc/rc.chorus.`*`target.system.IP.address`* is not available. Note therefore that `C_INIT` only finds the `rc.chorus` file if one of the commands in `sysadm.ini` mounts a complete root file system for the target.

Finally, `C_INIT` attempts to read `/etc/security` and activate secured mode.

SECURITY

`C_INIT` relies on the information in the **`security`**(4CC) file to perform user authentification after it activates secure mode. As explained above, `C_INIT` attemps to activate secured mode as the last step of system initialization. As a result, `C_INIT` normally performs authentification authentification for all commands issued after it executes the `sysadm.ini` and `rc.chorus` files.

In secured mode, authentification fails when a user issues a command through `rsh` from a remote host if:

- The user name of the remote users is not listed in the `security` file
- The remote host is not listed in the list of remote hosts in the `security` file entry for the user
- The user is not `TRUSTED`, yet the command may only be executed by a `TRUSTED` user.

When authentification fails, a `permission denied` message is sent back to the host and the command is aborted. If authentification succeeds, the user's privilege (`TRUSTED` or not) and credentials (`uid`, `gid` and additional groups) are read from the security file. `TRUSTED` users have access to all `C_INIT` commands, which includes many system administration commands.

The local console user is `TRUSTED` and has default credentials.

If `C_INIT` cannot read `/etc/security`, it does not activate secured mode. All users are `TRUSTED` and have default credentials.

C_INIT COMMANDS

Remote hosts communicate with `C_INIT` through the `rsh` command on the host. Commands therefore take the form:

`rsh` *target_name C_INIT_command*

`C_INIT` processes commands synchronously, in the order it receives them. When using `C_INIT` to run actors that should not terminate (daemons), an ampersand, `&`, must be added to the end of the command.

The following operators allow redirection in `C_INIT` commands:

`<` Redirects standard output

`>` Redirects standard output to a file. If the file exists, its contents are overwritten. If the file does not exist, it is created.

`>&` Redirects standard output and standard error to a file. If the file exists, its contents are overwritten. If the file does not exist, it is created.

`>>` Appends standard output. Similar to `>`, except that it places output at the end of the file rather than overwriting the content of the file.

`>>&` Appends standard output and standard error. Similar to `>&`, except that it places output at the end of the file rather than overwriting the content of the file.

Note that you must escape these operators to prevent the host shell from attempting to interpret them.

The following C_INIT commands are available. Many are also available as dynamically loadable, standalone commands.

AKILL **akill** [–c] [–g *gid*] *aid*

Kills the c_actor whose actor ID is *aid*.

akill supports the following options:

c Used to unblock an actor started in debug mode, using arun -d, that is not attached to a debugger.

g Used to flush a group of restartable actors with group ID *gid*. All persistent memory blocks used by actors in the group are deleted, and all the actors are then killed. If an attempt is made to kill a restartable actor without using the -g option, the actor will be restarted (as if it had crashed) and not killed.

This command cannot be used to kill C_INIT. Only TRUSTED users can kill c_actors with different uids than the current uid.

APS **aps**

Displays the list of all c_actors running on the target system, including the following information for each c_actor:

UID The user ID

AID The actor ID

NAME The name (argv[0]) of the c_actor

DBG 1 if the c_actor is being debugged.

ARUN **arun** [–D] [–d] [–g *gid*] [–k] [–q] [–S|–U] [–T] [–xip] [–Z] *actor_name* [*arguments*]

Runs *actor_name* on the target system and reports the actor ID (*aid*) of the new c_actor. If secured mode is activated, *actor_name* runs with the credentials specified in /etc/security. If not, *actor_name* runs with the *uid* and *gid* specified during configuration of the ChorusOS system image. The current working directory is the root directory of the target system where *actor_name* started. The search PATH for C_INIT is /bin.

arun accepts the following arguments:

–D Starts *actor_name* in DEBUGMODE mode, in which system retains extra information about thread execution that is useful for some source-level debuggers.

–d	Starts *actor_name* in a stopped state, usually to allow a debugger to attach itself to the `c_actor` before the actor starts.
–g *gid*	Starts *actor_name* as part of a restartable actor group whose identifier is specified as *gid*. The *gid* parameter must be between 0 and `cinit.hrCtrl.maxgroups-1`. If no *gid* is specified, the default value is 0.
–k	Allows the debugger to access the symbol table of *actor_name*. This option is ignored for user `c_actor`s.
–q	Do not print the string `Started aid` *aid*.
–S	Starts *actor_name* as a supervisor actor. Mutually exclusive with –U. This option may only be used by `TRUSTED` users. If neither –S nor –U is set, privilege is deduced from the binary at load time.
–xip	Loads the binary text segment of the actor code in place, without copying it. The data segment is copied, however. By default, both new actor code and data segments are loaded in newly allocated memory regions. The –xip option allows execution of absolute binaries built into the system image such as network administration daemons located in the `/dev/archive` directory. In order to optimize memory use even further, you can make the actor a boot actor. For boot actors, both text and data segments are loaded in place. Note, however, that boot actors are limited in that they cannot be started dynamically, using the `arun` command, and, futhermore, only one instance of a boot actor may be run.
–T	Starts *actor_name* as a `TRUSTED` actor. This option may only be used by `TRUSTED` users.
–U	Starts *actor_name* as a user actor. Mutually exclusive with –S. If neither –S nor –U is set, privilege is deduced from the binary at load time.

–Z	Starts *actor_name* in a stopped state, usually to allow a debugger to attach itself to the `c_actor` before the actor starts.
actor_name	Full path name to the actor on the target system.
arguments	Arguments passed to the actor without being interpreted by `C_INIT`.

CHORUSSTAT **chorusStat**

If the `ADMIN_CHORUSSTAT` feature is set to `true`, `chorusStat` displays information about currently allocated resources such as actors, threads and memory regions. See **`chorusStat`**(1CC) for details.

CONSOLE **console**

If the `LOCAL_CONSOLE` feature is set to `true`, `console` starts a thread within the `C_INIT` actor that runs a command interpreter in an infinite loop to read input from the console of the target system.

The `console` command may be run only once during the lifetime of the system. Subsequent attempts to run the command cause a `console already running` error message to be displayed on the console.

DTREE **dtree**

Displays all connected devices in the target device tree. See the *ChorusOS 4.0 Device Driver Framework Guide*, the *ChorusOS 4.0 Porting Guide* and specific device man pages for details.

ECHO **echo** [*arguments*]

Echos *arguments* to standard output.

ENV **env**

Displays the current environment.

HELP **help**

Displays a brief message summarizing available commands.

IFCONFIG **ifconfig** *interface* [*addr_family*] *addr* [*dest_addr*] [*parameters*]

ifconfig *interface* [*protocol_family*]

Configures a network interface or network interface parameters. See **`ifconfig`**(1M) for details.

IFWAIT **ifwait** *ifname* [*timeout*]

Waits for an interface to be set up. `ifwait` is typically used if the network is configured with DHCP or PPP, both of which take an undefined amount of time to configure the interface. `ifwait` takes the following parameters:

ifname Interface name, such as `ppp0`.

timeout Maximum time to wait in seconds. If *timeout* is not specified, `ifwait` returns only after *ifname* is up.

MEMSTAT **memstat**

Displays information about current memory usage, including total memory size, current free memory and current locked memory, in bytes.

MKDEV **mkdev** bpf *unit*

mkdev ifeth *unit* [*dtree_path*]

mkdev lo *unit*

mkdev ppp *unit*

mkdev sl *unit*

mkdev tty *unit* [*dtree_path*]

Creates in interface in the `IOM` actor. `mkdev` behaves differently, depending on the type of device to create. It takes the following arguments:

bpf Creates a BSD packet filter structure in the `IOM` actor. *bpf* requires a node device such as `/dev/bpf0`.

dtree_path Binds the interface to the hardware device corresponding to the path in the device driver tree. If *dtree_path* is not provided, the interface is bound to the first available device in the device driver tree.

ifeth Creates an ifnet structure of type Ethernet in the `IOM` actor.

lo Creates an ifnet structure of type loopback in the `IOM` actor. The loopback interface is purely logical.

ppp Creates an ifnet structure of type PPP in the IOM actor. The interface does not rely directly on a serial hardware driver, but instead is bound at runtime to a tty interface.

sl Creates an ifnet structure of type SLIP in the IOM actor. The interface does not rely directly on a serial hardware driver, but instead is bound at runtime to a tty interface.

tty Creates a tty structure in the IOM actor. In contrast to Ethernet, tty requires a node device, such as /dev/tty01.

unit Represents the unit number of the interface. For bpf and tty interfaces, the unit number corresponds to the node minor number. See the MKNOD section for details.

sysadm.ini(4CC) contains numerous examples of interface creation using mkdev.

MKNOD

mknod *name* [–b|–c] *major minor*

Creates a device in the /dev directory. mknod takes the following arguments:

name Creates the device file /dev/*name*.

–b Creates a buffered (block) device. Mutually exclusive with –c.

–c Creates a character (raw) device. Mutually exclusive with –b.

major Represents the device major number, different for each type of device.

minor Represents the device minor number, different for each node corresponding to a specific type of device.

sysadm.ini(4CC) contains numerous examples of device creation using mknod.

MOUNT

mount [–v]

mount [*hostaddr*:*filesystem mount_point*]

mount –t ufs *special_file mount_point*

mount –t msdosfs *special_file mount_point*

mount –t swap *special_file* /swap

If the ADMIN_MOUNT feature is set to `true`, `mount` adds a file system to the existing directory hierarchy. Such use of the command is restricted to `TRUSTED` users. The command may also be used without arguments or with the `–v` option only, in which case it displays a list of currently mounted filesystems.

`mount` accepts the following arguments:

filesystem Indicates the file system exported through NFS by the remote host to be mounted at *mount_point*.

hostaddr Indicates the IP address of the remote host exporting *filesystem* through NFS.

mount_point Indicates the point in the existing file system hierarchy under which the file system is mounted. If a UFS or MS-DOS file system is already mounted at *mount_point*, the new mount hides the old mount. This does not work if the file system mounted at *mount_point* is of type NFS.

special_file Indicates the device on which the file system or swap partition resides.

`–t` Indicates that the next argument specifies a local file system type (`ufs`, or `msdosfs`) or `swap` to indicate that the partition is to be used for paging and swapping.

`–v` Display information about currently mounted file systems in verbose format.

For a complete description of this command, see **`mount`**(1M).

NETSTAT

netstat [`–i`|`–r`]

Displays the state of network interfaces. `netstat` accepts the following options:

`–i` Displays the state of interfaces that have been auto-configured.

`–r` Displays the routing table.

PING

ping *host*

Request an ICMP ECHO_RESPONSE from the specified *host*. If *host* responds, `ping` prints *`host`* `is alive`, and then exits. Otherwise, after 20 seconds have elapsed, `ping` prints `no answer from` *`host`*.

PPPCLOSE

pppclose *device*

Requests that the `pppstart` daemon close a PPP line previously opened using `pppd` on *device*. `pppclose` may take time to finish, because closing a PPP line usually involves sending termination messages to the peer.

PPPD

pppd *device...*

Requests that the `pppstart` daemon start a thread to open a PPP line on *device*.

Before you can open a PPP line, you must enable PPP services by creating serial lines and interfaces for PPP, and then running `pppstart`.

PPPSTOP

pppstop

Disables PPP services on the target system by killing the `pppstart` daemon. All active lines are closed and `pppstart` terminates. Subsequent attempts to open PPP lines using `pppd` result in the following error message:

```
C_INIT: pppd: service unavailable
```

RARP

rarp *ifname*

Sets the IP address of the Ethernet interface *ifname*.

`rarp` does this by broadcasting a RARP request, waiting for a reply from the RARP server, and then executing `ifconfig` to set the interface IP address. If no reply is received from the RARP server, two more RARP requests are broadcast at intervals of three seconds. If no replies are received after the third RARP request, `rarp` returns an error. The following message is displayed on the target console:

```
no answer, retrying...
no answer, retrying...
no answer, retrying...
no answer, giving up.
C_INIT: rarp failed
```

REBOOT

reboot

Kills all `c_actors` on the target system, including `C_INIT`, as if `akill` had been used for each `c_actor`, synchonizes and unmounts all mounted file systems, and then reboots the target system. `reboot` is restricted to `TRUSTED` users.

RESTART

restart

Equivalent to `shutdown -i 1` command. `restart` is restricted to `TRUSTED` users.

ROUTE

route [–nqv] *command* [[*modifiers*] *args*]

If the ADMIN_ROUTE feature is set to `true`, `route` can be used as a `C_INIT` command that makes it possible to operate directly on the target system routing table.

The `C_INIT route` command differs from the standalone `route` utility only in that it does not support the `monitor` *command*, which continuously reports changes to the routing table. See **`route`**(1M) for details.

RSHD

rshd

If the RSH feature is set to `true`, `rshd` starts a thread within the `C_INIT` actor that runs a command interpreter in an infinite loop to read input from remote systems using the remote shell protocol.

The `rshd` command may be run only once during the lifetime of the system. Subsequent attempts to run the command cause a `rshd already running` error message to be displayed on the console.

SETENV

setenv *envar value*

Sets the environment variable *envar* to *value* and adds the variable to the environment. Variables added to the environment are inherited by `c_actor`s created by `C_INIT` using **`afexec`**(2K).

SHUTDOWN

shutdown [–i 0|1|2|3]

Brings the system to the state specified (`0`, `1`, `2` or `3`) as an argument to the –i. If no arguments are used, `shutdown` brings the system down to a state in which all actors have stopped running, and then reboots the system.

`shutdown` is restricted to TRUSTED and supervisor users.

See **`shutdown`**(1M) for details.

SLEEP

sleep [*seconds*]

Suspends execution of the current thread for *seconds*, or one second if the number of *seconds* is not specified.

SOURCE

source *filename*

Reads and executes commands in *filename*. This command may not be nested; do not include `source` *filename* from a file that is read using the `source` command.

Note - An ampersand, &, must be added to the end of `arun` commands running actors that should not terminate, such as daemons.

SWAPON **swapon** /swap

If the `FS_MAPPER`, `ON_DEMAND_PAGING` and `VIRTUAL_ADDRESS_SPACE` features are set to `true`, `swapon` can be used to specify an additional local device on which paging and swapping are to take place. See **swapon**(1M) for details.

ULIMIT **ulimit** [–afHnS] [*limit*]

Sets or displays resource limits. If *limit* is specified, the resource is set; otherwise, the current resource value is displayed. `ulimit` accepts the following arguments:

–a	Displays all resources. Mutually incompatible with *limit*.
–f	Displays or sets the maximum file size, in increments of 512-byte blocks, specified by *limit*.
–H	Displays or sets the hard limit specified by *limit*. All users can reduce hard limits. Only the superuser can increase them. Mutually incompatible with –S.
–n	Displays or sets the maximum number of descriptors that can be opened by a process, as specified by *limit*.
–S	Displays or sets the soft limit specified by *limit*. Mutually incompatible with –H.
limit	Specifies the value to which the limit is set. If neither the –H nor –S option is set, both hard and soft limits are set.

See **getrlimit**(2POSIX) and **setrlimit**(2POSIX) for details.

UMASK **umask** [mode]

Displays or sets the file creation mask for `C_INIT`. The mask is inherited by all `c_actors` spawned using the `arun` command.

If no *mode* is specified, the current value of the mask is displayed. The initial default value is `22`.

The *mode* is expressed in octal form: *ooo*. The three octal digits refer to the read, write and execute permissions for the owner, group and other users, respectively. See **chmod**(2POSIX) and **umask**(2POSIX) for details. The value

of each specified digit is subtracted from the corresponding "digit" specified by the system for file creation. For example, the mask `022` removes write permission for group and other users; files normally created using mode `777` take mode `755`; files created using mode `666` take mode `644`.

UMOUNT

umount *mount_point* | *special_file*

Unmounts the currently mounted file system specified either by *mount_point* or *special_file*, regardless of the type of file system that is currently mounted. `umount` is restricted to `TRUSTED` users.

UNSETENV

unsetenv *envar*

Sets the environment variable *envar* from the environment.

EXAMPLES

The following example creates several device interfaces and device nodes required for PPP:

```
mkdev tty 0 /pci/pci-isa/ns16550-2
mkdev tty 1 /pci/pci-isa/ns16550-3
mknod /dev/tty01 c 6 0
mknod /dev/tty01 c 6 1
mkdev bpf 0
mkdev bpf 1
mknod /dev/bpf0 c 20 0
mknod /dev/bpf1 c 20 1
```

First, two tty interfaces are bound to serial ports and their nodes created. PPP binds to these nodes at runtime. Next, two BSD packet filter interfaces are created with their corresponding nodes. Note that the `bpf` interfaces are not bound to hardware devices.

The following example uses `ifwait` while initializing a PPP line, and may be used in the system initialization file, `sysadm.ini`:

```
arun /dev/archive/pppstart &
pppd /dev/tty01
ifwait ppp0
```

First, the `pppstart` actor, which has been built in to the system image, is started to enable PPP. Next, the `pppd` command opens a PPP line on `tty01`. Finally, the `ifwait` causes the system to wait until `ppp0` is set up on `tty01`.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO **rc.chorus**(4CC), **security**(4CC), **sysadm.ini**(4CC), **mount**(1M)

NAME

dhclient – Dynamic Host Configuration Protocol client

SYNOPSIS

dhclient [–p *port*] [*if0...*]

FEATURES

BPF

DESCRIPTION

The Internet Software Consortium DHCP client, `dhclient`, provides a means for configuring one or more network interfaces using the Dynamic Host Configuration Protocol, BOOTP protocol, or, if these protocols fail, by statically assigning an address.

EXTENDED DESCRIPTION

The DHCP protocol allows a host to contact a central server which maintains a list of IP addresses which may be assigned on one or more subnets. A DHCP client may requiest an address from this pool, and then use it on a temporary basis for communication on the network. The DHCP protocol also provides a mechanism whereby a client can learn important details about the network to which it is attached, such as the location of a default router, the location of a name server and so on.

On startup, dhclient reads the `/dev/archive/dhclient.cf` for configuration instructions. It then gets a list of all the network interfaces that are configured in the current system. For each interface, it attempts to configure the interface using the DHCP protocol. `dhclient` does not keep track of leases across system reboot and server restart. As a result, `dhclient` always negotiates its IP address from scratch.

COMMAND LINE

The names of the network interfaces that `dhclient` should attempt to configure may be specified on the command line. If no interface names are specified on the command line `dhclient` will identify all network interfaces, elimininating non-broadcast interfaces if possible, and attempt to configure each interface.

If dhclient should listen and transmit on a port other than the standard (port 68), the –p flag may used. It should be followed by the udp port number that `dhclient` should use. This is mostly useful for debugging purposes.

FILES

`/dev/archive/dhclient.cf`

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`bpf`(7S), **`dhclient.cf`**(4CC), **`dhcp-options`**(4CC)

AUTHOR

`dhclient` was written by Ted Lemon <mellon@vix.com> under a contract with Vixie Labs. Funding for this project was provided by the Internet Software Corporation. Information about the Internet Software Consortium can be found at `http://www.isc.org/isc`.

This client was substantially modified and enhanced by Elliot Poger for use on Linux while he was working on the MosquitoNet project at Stanford.

The current version owes much to Elliot's Linux enhancements, but was substantially reorganized and partially rewritten by Ted Lemon so as to use the same networking framework that the Internet Software Consortium DHCP server uses.

LIMITATIONS

Many options are defined for DHCP. The implementation of `dhclient` in this release takes only the subnet-mask option into account. Other options are silently ignored.

NAME

disklabel – read and write disk pack label

SYNOPSIS

disklabel [–r] disk

disklabel –w [–r] disk disktype [packid]

DESCRIPTION

The *disklabel* command can be used to install, examine or modify the label on a disk drive or pack. When writing the label, it can be used to change the drive identification, the disk partitions on the drive, or to replace a damaged label.

There are two forms of the command that read (display), install, or rewrite the label on a disk. Each form has an additional option, `-r` , which causes the label to be read from or written to the disk directly, rather than going through the system's in-core copy of the label. This option may allow a label to be installed on a disk without kernel support for a label, such as when labels are first installed on a system. It must be used when first installing a label on a disk. The specific effect of `-r` is described under each command.

The first form of the command (read) is used to examine the label on the named disk drive (for example, sd0 or /dev/rsd0c). It will display all of the parameters associated with the drive and its partition layout. Unless the `-r` flag is set, the kernel's in-core copy of the label is displayed; if the disk has no label, or the partition types on the disk are incorrect, the kernel may have constructed or modified the label. If the `-r`. flag is set, the label from the raw disk will be displayed rather than the in-core label.

The second form of the command, with the `-w` flag set, is used to write a standard label on the designated drive. The required arguments to disklabel are the drive to be labelled (for example, sd0), and the drive type as described in the *disktab (4CC)* file. The drive parameters and partitions are taken from that file. If different disks of the same physical type are to have different partitions, it will be necessary to have separate disktab entries describing each, or to edit the label after installation as described below. The optional argument is a pack identification string, up to 16 characters long. The pack id must be quoted if it contains blanks.

If the `-r` flag is set, the disk sectors containing the label and bootstrap will be written directly. A side-effect of this is that any existing bootstrap code will be overwritten and the disk rendered unbootable for any kind of operating system. If `-r` is not specified, the existing label will be updated via the in-core copy and any bootstrap code will be unaffected. If the disk does not already have a label, the `-r` flag must be used. In either case, the kernel's in-core label is replaced.

When updating the label, *disklabel* first tries to detect any existing valid DOS partitions on the disk. If any are found, *disklabel* notifies the user and asks for confirmation. This confirmation can be suppressed by using the `-y` argument

(assuming a `yes` response) or `-n` argument (assuming a `no` response) to all questions. These arguments allow *disklabel* to be run in an *unattended* mode.

FILES

`/etc/disktab`

EXAMPLES

`disklabel sd0`

Display the in-core label for sd0 as obtained via `/dev/rsd0c`.

`disklabel -w -r /dev/rsd0c sd2212 foo`

Create a label for sd0 based on information for ‘‘sd2212’’ found in `/etc/disktab`. Any existing bootstrap code will be clobbered.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`disktab`(4CC)

DIAGNOSTICS

The kernel device drivers will not allow the size of a disk partition to be decreased, or the offset of a partition to be changed, while it is open. Some device drivers create a label containing a single large partition if a disk is unlabeled; the label must therefore be written to the ‘‘a’’ partition of the disk while it is open. This sometimes requires the desired label to be set in two steps, and the second setting the label on the new partition while shrinking the ‘‘a’’ partition.

The *newfs(1M)* utility will disallow creation of filesystems on FS_BOOT partitions.

BUGS

When a disk name is given without a full pathname, the constructed device name uses the ‘‘c’’ partition.

RESTRICTIONS FOR ChorusOS

Originally, this utility was able to install bootstrap code. Options related to bootstrap installation (–B, –b –s) are not supported in this version.

Restoring labels is not supported (–e option).

The –N option which prevents a new label being written on the disk is not supported. The reverse option –W which allows a new label to be written on the disk when a previous label was in read-only mode is not supported.

NAME

flashdefrag – defragment a flash memory

SYNOPSIS

flashdefrag [value]

DESCRIPTION

The `flashdefrag` command is used to defragment the flash memory referred to by the */dev/rflash* device name. This makes more physical sectors immediately available to be written to in the flash device, depnding on what is entered for the *value* argument.

When omitted, the value is assumed to be zero and no defragmentation is performed. In this case, the count of immediately available writable sectors is printed to the standard output.

If *value* is non null, the defragmentation will stop as soon as *value* sectors are available for writing, or if no further defragmentation is possible. When completed, the number of immediately available writable sectors is printed to the standard output.

DIAGNOSTICS

`flashdefrag` exits with 0 if successful, or > 0 in case of failure.

LIMITATIONS

It is not currently possible to pass the pathname of the flash to the command as an argument.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

format – format a Flash memory device

SYNOPSIS

format raw_device

FEATURES

FLASH

DESCRIPTION

`format` is used to prepare a Flash memory device, *raw_device*, so that it may be labelled. By convention, *raw_device* is created during system initialization as `/dev/rflash` using the **C_INIT**(1M) built-in command, `mknod`.

`format` must be used before **disklabel**(1M).

FILES

`/dev/rflash`

SEE ALSO

C_INIT(1M)

NAME

fsck – filesystem consistency check and interactive repair

SYNOPSIS

fsck [–bblock#] [–clevel] [–lmaxparallel] [–y] [–n] filesystem

DESCRIPTION

The `fsck` command audits and interactively repairs inconsistent conditions for filesystems. If the filesystem is inconsistent, the operator is prompted for concurrence before each correction is attempted.

The following flags are interpreted by `fsck`:

`–b` Use the block specified immediately after the flag as the super block for the filesystem. Block 32 is usually an alternate super block.

`–y` Assume a yes response to all questions asked by `fsck`. This should be used with great caution, as itis a license to continue after major problems have been encountered.

`–n` Assume a no response to all questions asked by `fsck` except for `CONTINUE?`, which is assumed to be affirmative; do not open the filesystem for writing.

Inconsistencies checked are as follows:

– Blocks claimed by more than one inode or the free map

– Blocks claimed by an inode outside the range of the filesystem

– Incorrect link counts

Size checks:

– Directory size not a multiple of DIRBLKSIZ

– Partially truncated file

– Bad inode format

– Blocks not accounted for anywhere

Directory checks:

– File pointing to unallocated inode

– Inode number out of range

– Dot or dot-dot are not the first two entries of a directory or have the wrong inode number

Super Block checks:

– More blocks for inodes than there are in the filesystem

– Bad free block map format

– Total free block and / or free inode count incorrect

DIAGNOSTICS

The diagnostics produced by `fsck` are fully enumerated and explained in Appendix A of *Fsck – The UNIX File System Check Program*

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`newfs`(1M)

NAME

fsck_dos – create an MS-DOS (FAT) file system

SYNOPSIS

fsck_dos [–p]

fsck_dos [–n|–y] [–v]

DESCRIPTION

This utility checks and repairs FAT filesystems (more commonly known as *DOS* filesystems).

The first form of *fsck_dos* preens the specified filesystems. It is normally started at boot time (see `rc.classix`) during automatic reboot.

When preening file systems, *fsck_dos* will repair common inconsistencies non-interactively. If more serious problems are found, *fsck_dos* does not try to repair them, indicates that it was unsuccessful, and exits.

The second form of *fsck_dos* checks the specified file systems and tries to repair all detected inconsistencies, requesting confirmation before making any changes.

The options are as follows:

–n Causes *fsck_dos* to use `no` as the answer to all operator questions, except `CONTINUE?`.

–p Preen the specified filesystems.

–y Causes *fsck_dos* to use `yes` as the answer to all operator questions.

–v Causes *fsck_dos* to display information about the boot sector. (Mainly used for debugging purposes.)

DIAGNOSTICS

An exit status of 0 is returned upon successful completion, and file system characteristics are printed. In the case of errors, the following exit statuses are returned:

1. Boot block was modified.
2. One or more directories were modified.
3. The FAT was modified.
4. Some unrecovered error remains.
5. Some unrecoverable error occurred.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO **C_INIT**(1M), **fsck**(1M), **newfs_dos**(1M)

NAME

ftpd – Internet File Transfer Protocol server

SYNOPSIS

/etc/ftpd_s [–dl] [–Tmaxtimeout] [–ttimeout]

DESCRIPTION

The *Ftpd* daemon is the Internet File Transfer Protocol server process. The server uses the *TCP* protocol and listens at the port specified in the `ftp` service specification (see *services(5)*).

Available options are:

`–d` Debugging information is written to the syslog using LOG_FTP.

`–l` Each successful and failed *ftp(1)* session is logged using syslog with a facility of LOG_FTP. If this option is specified twice, the retrieve (get), store (put), append, delete, make directory, remove directory and rename operations and their filename arguments are also logged.

`–T` A client may also request a different timeout period; the maximum period allowed may be set to *maxtimeout* seconds using the `–T` option. The default limit is 2 hours.

`–t` The inactivity timeout period is set to *timeout* seconds (the default is 15 minutes).

The file */etc/nologin* can be used to disable ftp access. If the file exists, `ftpd` displays it and exits. If the file */etc/ftpwelcome* exists, `ftpd` prints it before issuing the *ready* message.

The ftp server currently supports the following ftp requests, the case of the requests is ignored:

ABOR abort previous command

ACCT specify account (ignored)

ALLO allocate storage (vacuously)

APPE append to a file

CDUP change to parent of current working directory

CWD change working directory

DELE delete a file

HELP give help information

LIST give list files in a directory (ls -lgA)

MKD make a directory

MDTM show last modification time of file

MODE specify data transfer *mode*

NLST give name list of files in directory

NOOP do nothing

PASS specify password

PASV prepare for server-to-server transfer

PORT specify data connection port

PWD print the current working directory

QUIT terminate session

REST restart incomplete transfer

RETR retrieve a file

RMD remove a directory

RNFR specify rename-from file name

RNTO specify rename-to file name

SITE non-standard commands (see next section)

SIZE return size of file

STAT return status of server

STOR store a file

STOU store a file with a unique name

STRU specify data transfer *structure*

SYST show operating system type of server system

TYPE specify data transfer `type`

USER specify user name

XCUP change to parent of current working directory (deprecated)

XCWD change working directory (deprecated)

XMKD make a directory (deprecated)

XPWD print the current working directory (deprecated)

XRMD remove a directory (deprecated)

The remaining ftp requests specified in Internet RFC 959 are recognized, but not implemented.

The ftp server will abort an active file transfer only when the ABOR command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in Internet RFC 959. If a STAT command is received during a data transfer, preceded by a Telnet IP and Synch, the transfer status will be returned.

The *Ftpd* daemon interprets file names according to the *globbing* conventions used by csh. This allows users to use the metacharacters & * ? [ ]{ } ~ .

The *Ftpd* daemon authenticates users according to two rules:

Secure mode	The login name must be in the security data base, */etc/security* , and not have a null password (see *security*(4CC)). In this case a password must be provided by the client before any file operations may be performed.
Non secure mode	In this case the */etc/security* file is not present, no password check is performed and the user has the default credential values (see *conf*(1CC)).

In *secure mode*, if the user name is *anonymous* or `ftp`, an anonymous ftp account must be present in the security file (user `ftp`). In this case the user is allowed to log in with no password check, and `ftpd` takes special measures to restrict the client's access privileges.

FILES

`/etc/ftpwelcome`	Welcome notice.
`/etc/nologin`	Displayed and access refused.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

ftp(1CC), **C_INIT**(1M), **conf**(1CC), **security**(4CC)

RESTRICTIONS FOR ChorusOS

The *Ftpd* daemon must be started once using the following command:

% rsh -n <*target*> **arun /etc/ftpd_s** <*options*> **&**

Each ftp connection will then spawn a new *ftpd_s* actor.

The user's home directory is always the / directory.

If the –d or –l options are used, I/Os will go to the window where the `ftpd` server was started.

NAME

ifconfig – configure network interface parameters

SYNOPSIS

ifconfig interface <address_family> address <dest_address> <parameters>

ifconfig interface <protocol_family>

DESCRIPTION

The `ifconfig` command is used to assign an address to a network interface and/or configure network interface parameters. It must be used at boot time to define the network address of each interface present on a machine; it may also be used later to redefine an interface's address or other operating parameters.

Available operands for `ifconfig` are the following:

`Address` — For the DARPA-Internet family, the address is either a host name present in the host name data base *hosts(4CC)*, or a DARPA Internet address expressed using the Internet standard dot (".") notation. For the Xerox Network Systems family, addresses are *net:a.b.c.d.e.f*, where *net* is the assigned network number (in decimal), and each of the six bytes of the host number, *a* through *f*, are specified in hexadecimal. The host number may be omitted on 10Mb/s Ethernet interfaces, which use the hardware physical address, and on interfaces other than the first. For the ISO family, addresses are specified as a long hexadecimal string, as in the Xerox family. However, two consecutive dots imply a zero byte, and the dots are optional (if the user wishes to count out strings of digits in network byte order).

`address_family` — Specifies the *address family* which affects interpretation of the remaining parameters. As an interface can receive transmissions in differing protocols with different naming schemes, specifying the address family is recommeded. The address or protocol families currently supported are *inet*, *iso*, and *ns*.

`Interface` — The *interface* parameter is a string of the form *name unit*, for example, *en0*.

The following parameters may be set with `ifconfig` :

`alias` — Establish an additional network address for this interface. This is useful when changing network numbers, while still wanting to accept packets addressed to the old interface.

`arp`	Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses.
`arp`	Disable the use of the Address Resolution Protocol.
`broadcast`	(Inet only) Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host containing all 1's.
`debug`	Enable driver—dependent debugging code; this generally turns on extra console error logging.
`-debug`	Disable driver dependent debugging code.
`delete`	Remove the network address specified. This would be used if you specified an alias incorrectly, or if it was no longer needed. If you have set an NS address incorrectly which has the side effect of specifying the host portion, removing all NS addresses will allow you to respecify the host portion.
`dest_address`	Specify the address of the correspondent on the other end of a point—to—point link.
`down`	Mark an interface ''down''. When an interface is marked ''down'', the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well. This action does not automatically disable routes using the interface.
`ipdst`	This is used to specify an Internet host which is willing to receive ip packets encapsulating NS packets bound for a remote network. An apparent point—to—point link is constructed, and the address specified will be taken as the NS address and network of the destination. This cannot be used for IP encapsulation ofCLNP packets.

`metric n`	Set the routing metric of the interface to *n*, the default is 0. Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.
`netmask mask`	(Inet and ISO) . Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation Internet address, or with a pseudo-network name listed in the network table *networks(4CC)*. The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion.
`nsellength n`	(ISO only). This specifies a trailing number of bytes for a received NSAP used for local identification, the remaining leading part of which is taken to be the NET (Network Entity Title). The default value is 1, which conforms to US GOSIP. When an ISO address is set in an `ifconfig` command, it is really the NSAP which is being specified. For example, in US GOSIP, 20 hex digits should be specified in the ISO NSAP to be assigned to the interface. There is some evidence that a number other than 1 may be useful for AFI 37 type addresses.
`trailers`	Request the use of a ''trailer'' link—level encapsulation when sending (default). If a network interface supports *trailers*, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory—to—memory copy operations performed by the receiver. On networks that support the Address Resolution Protocol (see *arp(7P)* ; currently only 10 Mb/s

	Ethernet), this flag indicates that the system should request that other systems use trailers when sending to this host. Similarly, trailer encapsulations will be sent to other hosts that have made such requests. Currently used by Internet protocols only.
`-trailers`	Disable the use of a ‘‘trailer’’ link level encapsulation.
`link[0-2]`	Enable special processing of the link level of the interface. These three options are interface—specific in effect; however, they are generally used to select special modes of operation. An example of this is enabling SLIP compression. It is currently only used by SLIP.
`-link[0-2]`	Disable special processing at the link level with the specified interface.
`up`	Mark an interface ‘‘up’’. This may be used to enable an interface after an ‘‘ifconfig down.’’ It happens automatically when setting the first address on an interface. If the interface was reset when previously marked down, the hardware will be re-initialized.

The `ifconfig` command displays the current configuration for a network interface when no optional parameters are supplied. If a protocol family is specified, Ifconfig will report only the details specific to that protocol family.

Only the super-user may modify the configuration of a network interface.

DIAGNOSTICS

Messages indicating that the specified interface does not exist, the requested address is unknown, or the user is not privileged and tried to alter an interface’s configuration are returned by this command.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`netstat`(1CC)

NAME

inetNS, inetNShost, inetNSien116, inetNSdns, inetNSnis – Internet name servers

SYNOPSIS

inetNShost [user-hosts-file] [user-networks-file]

inetNSien116 server-address

inetNSdns [–env] [server-address-list] [domain]

inetNSnis

DESCRIPTION

The `inetNS` is a family of Internet Name Server daemons which answer local and remote site naming requests issued using **`gehostbyname`**(3STDC) or **`gethostbyaddr`**(3STDC) . It inserts a CHORUS port into a static user group (see **`grpAllocate`**(2k)), continuously waits for messages on that port, and then replies to clients. The daemons can be started and killed using the `arun` and `akill` **`C_INIT`**(1M) commands.

The INETNS environment parameter may be used to give the site number of an `inetNS` server if it is not located locally.

No more than one name server may be run on a given site.

inetNShost

`inetNShost` searches sequentially from the beginning of the local network host data base until a matching host name or host address is found, or until end-of-file is reached. If the optional *user-hosts-file* is not present, the `/etc/hosts` data base is used. The **getnetbyname()** and **getnetbyaddr()** routines are also provided to get networks' names and addresses from the `/etc/networks` file. If the optional *user-networks-file* is omitted, then the `/etc/networks` data base is used. If `/etc/hosts` and `/etc/networks` , `inetNShost` fails.

inetNSien116

`inetNSien116` calls a UDP Name Server as specified in IEN116 whose Internet address is *server-address* (specified using the "." notation). This name server always causes **gethostbyaddr()** to return a NULL value.

inetNSdns

`inetNSdns` calls a DOMAIN Name Server. The default domain and server address are taken from the `/etc/resolv.conf` file unless specified otherwise in the optional arguments. If optional arguments are used, *server-address-list* is a list of dot (".") notation Internet addresses separated by spaces.

When you use the –env option, `inetNSdns` takes the DOMAIN name server addresses from the ChorusOS environment variables DNS1 and DNS2 . These variables can be set using an external tool such as **`pppd`**(1M) . They are looked for each time a site naming request occurs (**`gethostbyaddr`**(3STDC) or **`gehostbyname`**(3STDC)). If these variables are not set when `inetNSdns` is started, the DOMAIN name server addresses are initialized from the

`/etc/resolv.conf` file and the *server-address-list* arguments, as explained above.

inetNSnis

inetNSnis calls an NIS (Network Information Services) Name Server. There is no default domain. The NIS domain must be set using the **domainname**(1CC) command. In order to run the NIS Name Server, two other actors have to be activated: the **portmap**(1M) actor and the **ypbind**(1M) actor. The NIS Name Server allows processes to be run on target NIS Clients. These processes can be used to interrogate NIS Servers located on other machines on the network using the **ypcat**(1CC) and **ypmatch**(1CC) commands. There is also an API which can run **gethostbyaddr**(3STDC) , **gehostbyname**(3STDC) , **getnetbyaddr**(3POSIX) , and **getnetbyname**(3POSIX) .

FILES

/etc/hosts

/etc/networks

/etc/resolv.conf

NOTES

getnetbyname(3CC) and getnetbyaddr(3CC) are only provided for use in the *inetNShost* and *inetNSnis* Name Servers.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

hosts(4CC) , **networks**(4CC) , **resolv.conf**(4CC)

For NIS Name Server:

domainname(1CC) , **ypbind**(1M) , **ypcat**(1CC) , **ypmatch**(1CC) , **ypwhich**(1CC) , **gethostbyaddr**(3STDC) , **gehostbyname**(3STDC) , **getnetbyaddr**(3POSIX) , **getnetbyname**(3POSIX)

NAME

inetNS, inetNShost, inetNSien116, inetNSdns, inetNSnis – Internet name servers

SYNOPSIS

inetNShost [user-hosts-file] [user-networks-file]

inetNSien116 server-address

inetNSdns [–env] [server-address-list] [domain]

inetNSnis

DESCRIPTION

The `inetNS` is a family of Internet Name Server daemons which answer local and remote site naming requests issued using **`gehostbyname`**(3STDC) or **`gethostbyaddr`**(3STDC) . It inserts a CHORUS port into a static user group (see **`grpAllocate`**(2k)), continuously waits for messages on that port, and then replies to clients. The daemons can be started and killed using the `arun` and `akill` **`C_INIT`**(1M) commands.

The INETNS environment parameter may be used to give the site number of an `inetNS` server if it is not located locally.

No more than one name server may be run on a given site.

inetNShost

`inetNShost` searches sequentially from the beginning of the local network host data base until a matching host name or host address is found, or until end-of-file is reached. If the optional *user-hosts-file* is not present, the `/etc/hosts` data base is used. The **getnetbyname()** and **getnetbyaddr()** routines are also provided to get networks' names and addresses from the `/etc/networks` file. If the optional *user-networks-file* is omitted, then the `/etc/networks` data base is used. If `/etc/hosts` and `/etc/networks` , `inetNShost` fails.

inetNSien116

`inetNSien116` calls a UDP Name Server as specified in IEN116 whose Internet address is *server-address* (specified using the "." notation). This name server always causes **gethostbyaddr()** to return a NULL value.

inetNSdns

`inetNSdns` calls a DOMAIN Name Server. The default domain and server address are taken from the `/etc/resolv.conf` file unless specified otherwise in the optional arguments. If optional arguments are used, *server-address-list* is a list of dot (".") notation Internet addresses separated by spaces.

When you use the –env option, `inetNSdns` takes the DOMAIN name server addresses from the ChorusOS environment variables DNS1 and DNS2 . These variables can be set using an external tool such as **`pppd`**(1M) . They are looked for each time a site naming request occurs (**`gethostbyaddr`**(3STDC) or **`gehostbyname`**(3STDC)). If these variables are not set when `inetNSdns` is started, the DOMAIN name server addresses are initialized from the

`/etc/resolv.conf` file and the *server-address-list* arguments, as explained above.

inetNSnis

inetNSnis calls an NIS (Network Information Services) Name Server. There is no default domain. The NIS domain must be set using the **domainname**(1CC) command. In order to run the NIS Name Server, two other actors have to be activated: the **portmap**(1M) actor and the **ypbind**(1M) actor. The NIS Name Server allows processes to be run on target NIS Clients. These processes can be used to interrogate NIS Servers located on other machines on the network using the **ypcat**(1CC) and **ypmatch**(1CC) commands. There is also an API which can run **gethostbyaddr**(3STDC) , **gehostbyname**(3STDC) , **getnetbyaddr**(3POSIX) , and **getnetbyname**(3POSIX) .

FILES

/etc/hosts

/etc/networks

/etc/resolv.conf

NOTES

getnetbyname(3CC) and getnetbyaddr(3CC) are only provided for use in the *inetNShost* and *inetNSnis* Name Servers.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

hosts(4CC) , **networks**(4CC) , **resolv.conf**(4CC)

For NIS Name Server:

domainname(1CC) , **ypbind**(1M) , **ypcat**(1CC) , **ypmatch**(1CC) , **ypwhich**(1CC) , **gethostbyaddr**(3STDC) , **gehostbyname**(3STDC) , **getnetbyaddr**(3POSIX) , **getnetbyname**(3POSIX)

NAME

inetNS, inetNShost, inetNSien116, inetNSdns, inetNSnis – Internet name servers

SYNOPSIS

inetNShost [user-hosts-file] [user-networks-file]

inetNSien116 server-address

inetNSdns [–env] [server-address-list] [domain]

inetNSnis

DESCRIPTION

The `inetNS` is a family of Internet Name Server daemons which answer local and remote site naming requests issued using **`gehostbyname`**(3STDC) or **`gethostbyaddr`**(3STDC) . It inserts a CHORUS port into a static user group (see **`grpAllocate`**(2k)), continuously waits for messages on that port, and then replies to clients. The daemons can be started and killed using the `arun` and `akill` **`C_INIT`**(1M) commands.

The INETNS environment parameter may be used to give the site number of an `inetNS` server if it is not located locally.

No more than one name server may be run on a given site.

inetNShost

`inetNShost` searches sequentially from the beginning of the local network host data base until a matching host name or host address is found, or until end-of-file is reached. If the optional *user-hosts-file* is not present, the `/etc/hosts` data base is used. The **getnetbyname()** and **getnetbyaddr()** routines are also provided to get networks' names and addresses from the `/etc/networks` file. If the optional *user-networks-file* is omitted, then the `/etc/networks` data base is used. If `/etc/hosts` and `/etc/networks` , `inetNShost` fails.

inetNSien116

`inetNSien116` calls a UDP Name Server as specified in IEN116 whose Internet address is *server-address* (specified using the "." notation). This name server always causes **gethostbyaddr()** to return a NULL value.

inetNSdns

`inetNSdns` calls a DOMAIN Name Server. The default domain and server address are taken from the `/etc/resolv.conf` file unless specified otherwise in the optional arguments. If optional arguments are used, *server-address-list* is a list of dot (".") notation Internet addresses separated by spaces.

When you use the –env option, `inetNSdns` takes the DOMAIN name server addresses from the ChorusOS environment variables DNS1 and DNS2 . These variables can be set using an external tool such as **`pppd`**(1M) . They are looked for each time a site naming request occurs (**`gethostbyaddr`**(3STDC) or **`gehostbyname`**(3STDC)). If these variables are not set when `inetNSdns` is started, the DOMAIN name server addresses are initialized from the

`/etc/resolv.conf` file and the *server-address-list* arguments, as explained above.

inetNSnis

inetNSnis calls an NIS (Network Information Services) Name Server. There is no default domain. The NIS domain must be set using the **domainname**(1CC) command. In order to run the NIS Name Server, two other actors have to be activated: the **portmap**(1M) actor and the **ypbind**(1M) actor. The NIS Name Server allows processes to be run on target NIS Clients. These processes can be used to interrogate NIS Servers located on other machines on the network using the **ypcat**(1CC) and **ypmatch**(1CC) commands. There is also an API which can run **gethostbyaddr**(3STDC) , **gehostbyname**(3STDC) , **getnetbyaddr**(3POSIX) , and **getnetbyname**(3POSIX) .

FILES

/etc/hosts

/etc/networks

/etc/resolv.conf

NOTES

getnetbyname(3CC) and getnetbyaddr(3CC) are only provided for use in the *inetNShost* and *inetNSnis* Name Servers.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

hosts(4CC) , **networks**(4CC) , **resolv.conf**(4CC)

For NIS Name Server:

domainname(1CC) , **ypbind**(1M) , **ypcat**(1CC) , **ypmatch**(1CC) , **ypwhich**(1CC) , **gethostbyaddr**(3STDC) , **gehostbyname**(3STDC) , **getnetbyaddr**(3POSIX) , **getnetbyname**(3POSIX)

NAME

inetNS, inetNShost, inetNSien116, inetNSdns, inetNSnis – Internet name servers

SYNOPSIS

inetNShost [user-hosts-file] [user-networks-file]

inetNSien116 server-address

inetNSdns [–env] [server-address-list] [domain]

inetNSnis

DESCRIPTION

The `inetNS` is a family of Internet Name Server daemons which answer local and remote site naming requests issued using **`gehostbyname`**(3STDC) or **`gethostbyaddr`**(3STDC) . It inserts a CHORUS port into a static user group (see **`grpAllocate`**(2k)), continuously waits for messages on that port, and then replies to clients. The daemons can be started and killed using the `arun` and `akill` **`C_INIT`**(1M) commands.

The INETNS environment parameter may be used to give the site number of an `inetNS` server if it is not located locally.

No more than one name server may be run on a given site.

inetNShost

`inetNShost` searches sequentially from the beginning of the local network host data base until a matching host name or host address is found, or until end-of-file is reached. If the optional *user-hosts-file* is not present, the `/etc/hosts` data base is used. The **getnetbyname()** and **getnetbyaddr()** routines are also provided to get networks' names and addresses from the `/etc/networks` file. If the optional *user-networks-file* is omitted, then the `/etc/networks` data base is used. If `/etc/hosts` and `/etc/networks` , `inetNShost` fails.

inetNSien116

`inetNSien116` calls a UDP Name Server as specified in IEN116 whose Internet address is *server-address* (specified using the "." notation). This name server always causes **gethostbyaddr()** to return a NULL value.

inetNSdns

`inetNSdns` calls a DOMAIN Name Server. The default domain and server address are taken from the `/etc/resolv.conf` file unless specified otherwise in the optional arguments. If optional arguments are used, *server-address-list* is a list of dot (".") notation Internet addresses separated by spaces.

When you use the –`env` option, `inetNSdns` takes the DOMAIN name server addresses from the ChorusOS environment variables DNS1 and DNS2 . These variables can be set using an external tool such as **`pppd`**(1M) . They are looked for each time a site naming request occurs (**`gethostbyaddr`**(3STDC) or **`gehostbyname`**(3STDC)). If these variables are not set when `inetNSdns` is started, the DOMAIN name server addresses are initialized from the

`/etc/resolv.conf` file and the *server-address-list* arguments, as explained above.

inetNSnis

inetNSnis calls an NIS (Network Information Services) Name Server. There is no default domain. The NIS domain must be set using the **domainname**(1CC) command. In order to run the NIS Name Server, two other actors have to be activated: the **portmap**(1M) actor and the **ypbind**(1M) actor. The NIS Name Server allows processes to be run on target NIS Clients. These processes can be used to interrogate NIS Servers located on other machines on the network using the **ypcat**(1CC) and **ypmatch**(1CC) commands. There is also an API which can run **gethostbyaddr**(3STDC) , **gehostbyname**(3STDC) , **getnetbyaddr**(3POSIX) , and **getnetbyname**(3POSIX) .

FILES

/etc/hosts

/etc/networks

/etc/resolv.conf

NOTES

getnetbyname(3CC) and getnetbyaddr(3CC) are only provided for use in the *inetNShost* and *inetNSnis* Name Servers.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

hosts(4CC) , **networks**(4CC) , **resolv.conf**(4CC)

For NIS Name Server:

domainname(1CC) , **ypbind**(1M) , **ypcat**(1CC) , **ypmatch**(1CC) , **ypwhich**(1CC) , **gethostbyaddr**(3STDC) , **gehostbyname**(3STDC) , **getnetbyaddr**(3POSIX) , **getnetbyname**(3POSIX)

NAME

inetNS, inetNShost, inetNSien116, inetNSdns, inetNSnis – Internet name servers

SYNOPSIS

inetNShost [user-hosts-file] [user-networks-file]

inetNSien116 server-address

inetNSdns [–env] [server-address-list] [domain]

inetNSnis

DESCRIPTION

The `inetNS` is a family of Internet Name Server daemons which answer local and remote site naming requests issued using **`gehostbyname`**(3STDC) or **`gethostbyaddr`**(3STDC) . It inserts a CHORUS port into a static user group (see **`grpAllocate`**(2k)), continuously waits for messages on that port, and then replies to clients. The daemons can be started and killed using the `arun` and `akill` **`C_INIT`**(1M) commands.

The INETNS environment parameter may be used to give the site number of an `inetNS` server if it is not located locally.

No more than one name server may be run on a given site.

inetNShost

`inetNShost` searches sequentially from the beginning of the local network host data base until a matching host name or host address is found, or until end-of-file is reached. If the optional *user-hosts-file* is not present, the `/etc/hosts` data base is used. The **getnetbyname()** and **getnetbyaddr()** routines are also provided to get networks' names and addresses from the `/etc/networks` file. If the optional *user-networks-file* is omitted, then the `/etc/networks` data base is used. If `/etc/hosts` and `/etc/networks` , `inetNShost` fails.

inetNSien116

`inetNSien116` calls a UDP Name Server as specified in IEN116 whose Internet address is *server-address* (specified using the "." notation). This name server always causes **gethostbyaddr()** to return a NULL value.

inetNSdns

`inetNSdns` calls a DOMAIN Name Server. The default domain and server address are taken from the `/etc/resolv.conf` file unless specified otherwise in the optional arguments. If optional arguments are used, *server-address-list* is a list of dot (".") notation Internet addresses separated by spaces.

When you use the –`env` option, `inetNSdns` takes the DOMAIN name server addresses from the ChorusOS environment variables DNS1 and DNS2 . These variables can be set using an external tool such as **`pppd`**(1M) . They are looked for each time a site naming request occurs (**`gethostbyaddr`**(3STDC) or **`gehostbyname`**(3STDC)). If these variables are not set when `inetNSdns` is started, the DOMAIN name server addresses are initialized from the

`/etc/resolv.conf` file and the *server-address-list* arguments, as explained above.

inetNSnis

inetNSnis calls an NIS (Network Information Services) Name Server. There is no default domain. The NIS domain must be set using the **domainname**(1CC) command. In order to run the NIS Name Server, two other actors have to be activated: the **portmap**(1M) actor and the **ypbind**(1M) actor. The NIS Name Server allows processes to be run on target NIS Clients. These processes can be used to interrogate NIS Servers located on other machines on the network using the **ypcat**(1CC) and **ypmatch**(1CC) commands. There is also an API which can run **gethostbyaddr**(3STDC) , **gehostbyname**(3STDC) , **getnetbyaddr**(3POSIX) , and **getnetbyname**(3POSIX) .

FILES

/etc/hosts

/etc/networks

/etc/resolv.conf

NOTES

getnetbyname(3CC) and getnetbyaddr(3CC) are only provided for use in the *inetNShost* and *inetNSnis* Name Servers.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

hosts(4CC) , **networks**(4CC) , **resolv.conf**(4CC)

For NIS Name Server:

domainname(1CC) , **ypbind**(1M) , **ypcat**(1CC) , **ypmatch**(1CC) , **ypwhich**(1CC) , **gethostbyaddr**(3STDC) , **gehostbyname**(3STDC) , **getnetbyaddr**(3POSIX) , **getnetbyname**(3POSIX)

NAME

mkfd – create a bootable floppy disk from a CHORUS boot image

SYNOPSIS

mkfd bootimage device

DESCRIPTION

The `mkfd` command creates a bootable floppy disk from a CHORUS boot image. A floppy disk is first formatted using a Unix low-level format; *mkfd* adds a minimal Unix filestytem (SVR4 boot loader and s5 filesystem) to it, then copies the CHORUS boot image onto the disk.

The path of the CHORUS boot image on the floppy disk should be called*bootimage*.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

mkfs – replaced by newfs(1M).

SEE

newfs(1M)

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

mknod – build special file

SYNOPSIS

mknod c | b major minor

DESCRIPTION

The *mknod* command creates special device files.

On top of ChorusOS nodes are built manually, the four arguments required are:

`device_name` Device name, for example *sd* for a SCSI disk, *hd* for an IDE disk, or *ppp* for a PPP device.

`b | c` Type of device. If the device is a block type device such as a tape or disk drive which needs both cooked and raw special files, the type is *b*. All other devices are character type devices, such as terminal and pseudo devices, and are type `c`.

`major` The major device number is an integer number which tells the kernel which device driver entry point to use.

`minor` The minor device number tells the kernel which subunit the node corresponds to on the device; for example, a subunit may be a filesystem partition or a tty line.

EXAMPLES

```
mknod /dev/hd0a b 0 0
```

This creates the first partition of an IDE disk in bloc mode.

You can also read the *UFS User's Guide* to know which supported disk devices you can create On top of ChorusOS.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`mknod` *(2PSX)*

NAME

mount – mount file systems

SYNOPSIS

mount [–v]

mount [–t nfs *options*] *hostaddr:file_system mount_point*

mount –t msdosfs [*options*] *special_device mount_point*

mount –t ufs [*options*] *special_device mount_point*

mount –t pdevfs *mount_point*

mount –t swap *mount_point* /swap

DESCRIPTION

mount calls the **mount**(2POSIX) system call to prepare and graft a remote node (*hostaddr:file_system*) or *special_device* onto the file system tree at the *mount_point*.

The system maintains a list of currently mounted file systems. If no arguments are given to mount, the list of mounted file systems is printed.

OPTIONS

mount supports the following options:

`–o` Options are specified with `–o` followed by a comma-separated list of options from the following list:

async All I/O to the file system should be done asynchronously. This is a dangerous option to use, and should not be used unless you are prepared to recreate the file system should your system crash.

force Forces the revocation of write access when trying to downgrade a file system mount status from read-write to read-only. Also forces the read–write mount of an unclean file system (dangerous; use with caution).

noatime Do not update the file access time when reading from a file. This option is useful on file systems where there are large numbers of files and performance is more critical than updating the file access time (which is rarely ever important). This option is currently only supported on local filesystems.

nodev Do not interpret character or block special devices on the file system. This option is useful for a server that has file systems containing special devices for architectures other than its own.

noexec Do not allow execution of any binaries on the mounted file system. This option is useful for a server that has file systems containing binaries for architectures other than its own.

nosuid Do not allow set-user-identifier or set-group-identifier bits to take effect.

Note - This option is worthless if a publicly available suid or sgid wrapper like `suidperl` is installed on your system.

---

rdonly Mount the file system read-only (even the super-user may not write to it).

	sync	All I/O to the file system should be done synchronously.
	update	Indicate that the status of an already mounted file system should be changed.
	union	Causes the namespace at the mount point to appear as the union of the mounted file system root and the existing directory. Lookups are done in the mounted file system first. If those operations fail due to a non-existent file the underlying directory is then accessed. All creates are done in the mounted file system.
–t *type*		The argument following the –t is used to indicate the file system type. nfs is the default.
–u		The –u flag indicates that the status of an already mounted file system should be changed. Any of the options discussed above (the –o option) may be changed; also a file system can be changed from read-only to read-write or vice versa. An attempt to change from read-write to read-only will fail if any files on the file system are currently open for writing unless the –f flag is also specified. The set of options is determined by first extracting the options for the file system from the **fstab**(4CC) table, then applying any options specified by the –o argument, and finally applying the –r or –w option.
–v		Verbose mode. This is the default mode.

DIAGNOSTICS

Various, most of them are self-explanatory.

filesystem not available

filesystem type is not supported.

FILES

/etc/fstab file system table

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

mount(2POSIX), **fstab**(4CC), **mount_msdos**(1M), **mount_nfs**(1M), **swapon**(1M), **umount**(1M)

NAME

mountd – NFS daemon providing remote mount services

SYNOPSIS

mountd [–n | –d] [export_file]

DESCRIPTION

The `mountd` daemon is the server for *NFS* mount requests from other client machines. It listens for service requests at the port indicated in the *NFS* server specification; see "Network File System Protocol Specification" RFC1094.

The following options and operands are available for `mountd`:

–n Allows non-root mount requests to be served. This should only be specified if there are clients such as PC's, that require it.

–d Allows `mountd` to be set in trace mode, displaying messages for each request. This flag is specific to ChorusOS.

The export file name alternate location can also be specified explicitly when `mountd` is started. This argument must be in the last position of the argument list. By default, the `/etc/exports` file is analyzed.

When `mountd` is started, it loads the export host addresses and options into the kernel using the *mount(2)* system call. After changing the exports file, a hangup signal should be sent to the `mountd` daemon so that it reloads the export information. All error messages are displayed to the screen, not in a log file. Upon each successful remote mount operation the `/etc/mtab` is updated containing the following information:

- The name of the machine
- The mounted file system

The `/etc/exports` file contains the list of exported file systems.

FILES

`/etc/exports`

`/etc/mtab`

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`exports`(4CC), **`nfsd`**(1M), **`portmap`**(1M), **`showmount`**(1M)

HISTORY

The `mountd` utility first appeared in 4.4BSD.

NAME

mount_msdos – mount an MSDOS file system

SYNOPSIS

mount_msdos [–9] [–g *gid*] [–L *locale*] [–l] [–m *mask*] [–o *options*] [–s] [–u *uid*] [–W *table*] *special_device node*

DESCRIPTION

`mount_msdos` attaches the MSDOS file system residing on the device *special_device* to the global file system namespace at the location indicated by *node*. This command is normally executed by **mount**(1M), but can be used by any user to mount an MSDOS file system on any directory that they own (provided, of course, that they have appropriate access to the device that contains the file system).

OPTIONS

`mount_msdos` supports the following options:

–9	Ignore the special Windows 95 directory entries even if deleting or renaming a file. This forces –s.
–g *gid*	Set the group of the files in the file system to *gid*. The default group is the group of the directory on which the file system is being mounted.
–L *locale*	Specify locale name used for internal uppercase and lowercase conversions for DOS and Windows 95 names. By default ISO 8859-1 assumed as local character set.
–l	Force listing and generation of Win'95 long filenames and sepa rate creation / modification / access dates. If neither –s nor –l are given, `mount_msdos` searches the root directory of the file system to be mounted for any existing Windows 95 long filenames. If no such entries are found, –s is the default. Otherwise –l is assumed.
–m *mask*	Specify the maximum file permissions for files in the file system. (For example, a *mask* of 755 specifies that, by default, the owner should have read, write, and execute permissions for files, but others should only have read and execute permissions. See **chmod**(2POSIX) for more information about octal file modes.) Only the nine low-order bits of mask are used. The default mask is taken from the directory on which the file system is being mounted.
–o *options*	Use the specified mount options, as described in **mount**(1M).
–s	Force behavior to ignore and not generate Windows 95 long filenames.

–u *uid* Set the owner of the files in the file system to *uid*. The default owner is the owner of the directory on which the file system is being mounted.

–W *table* Specify text file with three conversion tables:

1. Local character set to Unicode conversion table (upper half) for Windows 95 long names, 128 Unicode codes. If a code is not present in Unicode, use `0x003F` code (?) as a replacement.
2. DOS to local character set conversion table (upper half) for DOS names, 128 character codes. Code `0x3F` (?) used when translation is not possible.
3. Local character set to DOS conversion table (upper half) for DOS names, 128 character codes. Some codes have special meanings:

   0x00 character disallowed in DOS file name

   0x01 character should be replaced by _ in DOS file name

   0x02 character should be skipped in DOS file name.

By default ISO 8859-1 assumed as local character set. If the table path is not absolute, `/usr/libdata/msdosfs/` is prepended.

FILES

`/usr/libdata/msdosfs` default place for character set conversion tables

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

mount(2POSIX), **fstab**(4CC), **mount**(1M), **mount_nfs**(1M), **umount**(1M)

NOTES

The use of the –9 flag could result in damaged file systems, albeit the damage is in part taken care of by procedures similar to the ones used in Windows 95.

The default handling for –s and –l results in empty file systems being populated with short filenames only. To generate long filenames on empty DOS filesystems use –l.

NAME

mount_nfs – mount an NFS file system

SYNOPSIS

mount [–23bcdiKlPqsTU] [–a *maxreadahead*] [–D *deadthresh*] [–g *maxgroups*] [–I *readdirsize*] [–L *leaseterm*] [–m *realm*] [–o *options*] [–R *retrycnt*] [–r *readsize*] [–t *timeout*] [–w *writesize*] [–x *retrans*] *rhost*:*path node*

DESCRIPTION

mount_nfs calls the **mount**(2POSIX) system call to prepare and graft a remote NFS file system (*hostaddr*:*file_system*) onto the file system tree at the *mount_point*. The command is normally executed by **mount**(1M). It implements the mount protocol as described in RFC 1094, Appendix A and , *NFS: Network File System Version 3 Protocol Specification*, Appendix I.

OPTIONS

mount_nfs supports the following options:

–2 Use the NFS Version 2 protocol (the default is to try version 3 first then version 2).

–3 Use the NFS Version 3 protocol.

–a Set the read-ahead count to the specified value. This may be in the range of 0 to 4, and determines how many blocks will be read ahead when a large file is being read sequentially. Trying a value greater than 1 for this is suggested for mounts with a large bandwidth * delay product.

–b If an initial attempt to contact the server fails, fork off a child to keep trying the mount in the background. Useful when the filesystem mount is not critical to multiuser operation.

–c For UDP mount points, do not do a **connect**(2POSIX). This must be used for servers that do not reply to requests from the standard NFS port number 2049.

–D Used with NQNFS to set the *dead server threshold* to the specified number of round trip timeout intervals. After a dead server threshold of retransmit timeouts, cached data for the unresponsive server is assumed to still be valid. Values may be set in the range of 1 to 9, with 9 referring to an *infinite dead threshold* (never assume cached data is still valid). This option is not generally recommended and is really an experimental feature.

–d Turn off the dynamic retransmit timeout estimator. This may be useful for UDP mounts that exhibit high retry rates, since it is possible that the dynamically estimated timeout interval is too short.

`–g` Set the maximum size of the group list for the credentials to the specified value. This should be used for mounts on old servers that cannot handle a group list size of `16`, as specified in *RFC 1057*. Try `8`, if users in many groups cannot get a response from the mount point.

`–I` Set the *readdir* read size to the specified value. The value should normally be a multiple of `DIRBLKSIZ` that is less than or equal to the read size for the mount.

`–i` Make the mount interruptible, which implies that file system calls that are delayed due to an unresponsive server will fail with EINTR when a termination signal is posted for the process.

`–K` Pass Kerberos authenticators to the server for client-to-server user-credential mapping. This requires that the kernel be built with the `NFSKERB` option. (Refer to the INTERNET-DRAFT titled *Authentication Mechanisms for ONC RPC*, for more information.)

`–L` Used with `NQNFS` to set the lease term to the specified number of seconds. Only use this argument for mounts with a large round trip delay. Values are normally in the 10-30 second range.

`–l` Used with NQNFS and NFSV3 to specify that the ReaddirPlus RPC should be used. This option reduces RPC traffic for cases such as **`ls -l`**, but tends to flood the attribute and name caches with prefetched entries. Try this option and see whether performance improves or degrades. Probably most useful for client to server network interconnects with a large bandwidth * delay product.

`–m` Set the Kerberos realm to the string argument. Used with the `–K` option for mounts to other realms.

`–o` Options are specified with a `–o` flag followed by a comma separated string of options. See **`mount`**(1M) for possible op tions and their meanings. The following NFS specific option is also available:

port=*port_number* Use specified port number for NFS requests. The default is to

query the portmapper for the NFS port.

Historic options:

bg Same as –b.

conn Same as not specifying –c.

dumbtimer Same as –d.

intr Same as –i.

kerb Same as –K.

nfsv2 Same as –2.

nfsv3 Same as –3.

rdirplus Same as –l.

nqnfs Same as –q.

soft Same as –s.

–q Use the leasing extensions to the NFS Version 3 protocol to maintain cache consistency. This protocol Version 2, referred to as Not Quite Nfs (NQNFS), is only supported by this updated release of NFS code. (It is not backwards compatible with the release of NQNFS that went out on 4.4BSD-Lite. To interoperate with a 4.4BSD-Lite NFS system you will have to avoid this option until you have had an opportunity to upgrade the NFS code on all your 4.4BSD-Lite based systems.)

–R Set the retry count for doing the mount to the specified value.

–r Set the read data size to the specified value. It should normally be a power of two greater than or equal to 1024. This should be used for UDP mounts when the *fragments dropped due to timeout* value is getting large while actively using a mount point. (Use netstat(1) with the -s option to see what the fragments dropped due to timeout value is.) See the –w option as well.

–s A soft mount, which implies that file system calls will fail after Retry round trip timeout intervals.

`–t` Set the initial retransmit timeout to the specified value. May be useful for fine tuning UDP mounts over internetworks with high packet loss rates or an overloaded server. Try increasing the interval if **`nfsstat`**(1CC) shows high retransmit rates while the file system is active or reducing the value if there is a low retransmit rate but long response delay observed. (Normally, the `–d` option should be specified when using this option to manually tune the timeout interval.)

`–w` Set the write data size to the specified value. See the comments for the `–r` option, but considering the fragments dropped due to timeout value on the server instead of the client. Note that both the `–r` and `–w` options should only be used as a last ditch effort at improving performance when mounting servers that do not support TCP mounts.

`–x` Set the retransmit timeout count for soft mounts to the specified value.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`mount`(2POSIX), **`fstab`**(4CC), **`mount`**(1M), **`mount_msdos`**(1M), **`umount`**(1M)

NAME

newfs – construct a new file system

SYNOPSIS

newfs [–N] [–S *sector-size*] [–a–maxcontig] [–b *block-size*] [–c *cylinders*] [–d *rotdelay*] [–e *maxbpg*] [–f *frag-size*] [–i *bytes*] [–k *skew*] [–l *interleave*] [–m *freespace*] [–o *optimization*] [–p *sectors*] [–r *revolutions*] [–s *size*] [–t *tracks*] [–u *sectors*] [–x *sectors*]

DESCRIPTION

The newfs command replaces the now obsolete *mkfs(1M)* program. Before running newfs the disk must be labeled using *disklabel (1M)* .

The newfs command builds a file system on the special device specified, taking the defaults from the information on the disk label. Typically, the defaults are reasonable, however newfs has numerous options to allow the defaults to be selectively overridden.

The special file is only used to read the disk label, which provides a set of configuration parameters for the memory based file system. The special file is typically that of the primary swap area, as that is where the file system will be backed up when free memory gets low and the memory supporting the file system needs to be paged.

The following options define the general layout policies:

–N	Causes the file system parameters to be printed out (without creating the file system)
–a *maxcontig*	This specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay (see the –d option). The default value is one
–b *block-size*	The block size of the file system, in bytes
–c *#cylinders/group*	The number of cylinders per cylinder group in a file system. The default value is 16
–d *rotdelay*	This specifies the expected time (in milliseconds) needed to service a transfer completion interrupt and initiate a new transfer on the same disk. The default is 4 milliseconds
–e *maxbpg*	This indicates the maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. The default is approximately one quarter of the total blocks in a cylinder group

–f *free space %*	The percentage of space which users do not have access to; the minimum free space threshold. The default value is 10%
–o	Optimization preference ‘‘space’’ or ‘‘time’’. The file system can be instructed either to try to minimize the time spent allocating blocks, or to try to minimize the space fragmentation on the disk. If the value of minfree (see above) is less than 10%, the default is to optimize for space; if the value of minfree is greater than or equal to 10%, the default is to optimize for time
–s *size*	The size of the file system, in sectors

The following options override the standard sizes for the disk geometry. Their default values are taken from the disk label.

Changing these defaults is useful only when using `newfs` to build a file system whose raw image will eventually be used on a different type of disk than the one on which it is initially created (for example on a write-once disk) Note that changing any of these values from their defaults will make it impossible for `fsck` to find the alternate superblocks if the standard superblock is lost.

–S *sector-size*	The size of a sector in bytes (rarely anything other than 512)
–k	sector 0 skew , per track Used to describe fluctuations in the media format to compensate for a slow controller. Track skew is the offset of sector 0 on track N relative to sector 0 on track N-1 on the same cylinder
–l *hardware sector interleave*	Used to describe fluctuations in the media format to compensate for a slow controller. Interleave is the physical sector interleave on each track, specified as the denominator of the ratio: sectors read/sectors passed over. Thus an interleave of 1/1 means contiguous layout, while 1/2 means logical sector 0 is separated by one sector from logical sector 1
–p *spare sectors per track*	Spare sectors (bad sector replacements) are physical sectors that occupy space at the end of each track. They are not counted as part of the

sectors/track as they are not available to the file system for data allocation

–r *revolutions/minute* The speed of the disk in revolutions per minute

–t *#tracks/cylinder* The number of tracks/cylinder available for data allocation by the file system

–u *sectors/track* The number of sectors per track available for data allocation by the file system. This does not include sectors reserved at the end of each track for bad block replacement (see the –p option)

–x *spare sectors per cylinder* Spare sectors (bad sector replacements) are physical sectors that occupy space at the end of the last track in the cylinder. They are deducted from the sectors/track of the last track of each cylinder as they are not available to the file system for data allocation

FILES disktab(4CC), disklabel(1M)

ATTRIBUTES See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO **fsck**(1M), **C_INIT**(1M)

NAME

newfs_dos – create an MS-DOS (FAT) file system

SYNOPSIS

newfs_dos [–y | –n] [–N] [–B] [–c *clustersize*] [–F] [–l *volumelabel*]
[–n *serialnumber*] [–r *rootdirectorysize*] [–s *filesystemsize*] [–t *heads*]
[–u *sectors/track*]

DESCRIPTION

The *newfs_dos* command builds a file system structure on the `special device` specified that can be read by an`MSDOS` file system. The disk must have been labelled (see *disklabel(1M)*) before running *newfs_dos*. If the disk has already been partitioned using the MSDOS `fdisk` comand, and the partitions formatted using theMSDOS `format` command, it is not necessary to run *newfs_dos*. The ChorusOS MSDOS file system can use the MSDOS *partition table*, if detected, instead of BSD *disklabel*.

Typically, `Device` will be the character device node for a hard disk drive partition, (for example, /dev/rhd0a, /dev/rsd0a), or a ram disk drive (for example, /dev/rrd0a).

WARNING: All existing data will be unavailable after running *newfs_dos*.

The following options can be used to customize the file system:

–N	Causes the file system parameters to be printed out (without creating the file system)
–B	By default, *newfs_dos* will write an entirely new boot sector, erasing the existing one. If this option is set, only the BPB (*BIOS Parameter Block*) part of the boot sector will be overwritten
–c *cluster size*	Normally, *cluster size* is computed automatically from the file system size. This option will override this calculation; *cluster size* must be given in terms of sectors (512 bytes)
–F	Creates a FAT32 file system. Normally, *FAT* length is automatically computed from the file system size. This option forces the system to format the partition as a FAT32 FAT
–l *volume label*	Allows youto specify a *volume label* other than the default `ChorusOS`. The*volume label*character string must be surrounded by quotes if it contains spaces. The maximum length is twelve characters..

–n *serial number*	Allows you to specify a *serial number* instead of the default, which is randomly assigned. The*serial number* is a decimal number (no dot).
–r *root directory size*	Overrides the compiled *root directory size* parameter of the file system (112 files). This option is only meaningful with FAT12 or FAT16 file system types, because in this case the *root directory* has a `fixed` size which cannot be extended. When using *FAT32* file system types, the *root directory* is no longer a fixed size, and can be extended (like other directories). Use this option to define the maximum number of files you want to create under the root directory of the MSDOS file system.
–s *file system size*	The size of the file system in sectors. Usually, the file system size is obtained directly by reading the MSDOS *partition table* (if one exists) or the BSD *disklabel*.
–t *heads*	The number of physical *heads* of the disk. The default is to read the value from the MSDOS *partition table* (if one exists) or the BSD *disklabel*.
–u *sectors/track*	The number of sectors per track available for data allocation by the file system. The default is to read the value from the MSDOS *partition table* (if one exists) or the BSD *disklabel*.
–y	Assume a `yes` response to all questions asked by *newfs_dos*. Typically, *newfs_dos* tries to detect any existing valid DOS partitions on the disk. If any are found, *newfs_dos* notifies the user and asks for confirmation. This command line argument can be used to suppress this confirmation, assuming a 'yes' response.
–n	Assume a `no` response to all questions asked by *newfs_dos* (see above). These arguments allow *newfs_dos* to be run in an *unattended* mode.

DIAGNOSTICS

An exit status of 0 is returned upon successful completion, and the file system characteristics are printed. Exit status 1 is returned if any errors are detected during file system creation.

FILES

disktab(4CC)

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

disklabel(1M), **fsck_dos**(1M), **C_INIT**(1M)

NAME

nfsd – NFS daemon providing remote NFS services

SYNOPSIS

nfsd [–r|-u|-t] [–nnumber_of_servers]

DESCRIPTION

The `nfsd` daemon runs on a server machine to service *NFS* requests from client machines. At least one `nfsd` must be running for a machine to operate as a server. Unless otherwise specified, four servers for *UDP* transport are started.

The following options are available:

`–r` Register the *NFS* service using *portmap(8)* without creating any servers. This option can be used along with the *-u* or *-t* options to re-register *NFS* if the portmap server is restarted.

`–n` Specifies how many servers to create.

`–t` Serve *TCP NFS* clients.

`–u` Serve *UDP NFS* clients.

For example: `nfsd -ut -n 6 &` serves *UDP* and *TCP* transports using six daemons.

A server should run enough daemons to handle the maximum level of concurrent requests from its clients, typically four to six.

The `nfsd` daemon listens for service requests at the port indicated in the *NFS* server specification; see "Network File System Protocol Specification" , RFC1094.

DIAGNOSTICS

The `nfsd` daemon exits 0 if successful, and >0 if an error occurs.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`nfssvc`(2POSIX), **`mountd`**(1M), **`portmap`**(1M)

HISTORY

The `nfsd` utility first appeared in 4.4BSD.

NAME

portmap – DARPA port to RPC program number mapper

SYNOPSIS

portmap [–d]

DESCRIPTION

The `portmap` daemon is a server that converts *RPC* program numbers into *DARPA* protocol port numbers. It must be running in order to make *RPC* calls.

When an *RPC* server is started, it will tell `portmap` what port number it is listening to, and what *RPC* program numbers it is prepared to serve. When a client wishes to make an *RPC* call to a given program number, it will first contact `portmap` on the server machine to determine the port number to which *RPC* packets should be sent.

The `portmap` daemon must be started before any *RPC* servers are invoked.

Normally, `portmap` forks and dissociates itself from the terminal like any other daemon. On top of ChorusOS an *afexec* is invoked instead of a fork, and all error messages are displayed to a local screen instead of being logged to a file.

The following option is available:

–d This debug option prevents `portmap` from running as a daemon, and causes errors and debugging information to be printed to the standard error output. In addition on top of ChorusOS each request is displayed.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`rpcinfo`(1M), **`inetd`**(1M)

BUGS

If `portmap` crashes, all servers must be restarted.

HISTORY

The `portmap` command first appeared in BSD 4.3

NAME

pppd – Point-to-Point Protocol `C_INIT` command

SYNOPSIS

pppd *tty_name* [*speed*] [*options*]

DESCRIPTION

The Point-to-Point Protocol (PPP) provides a method for transmitting datagrams over serial point-to-point links. PPP is composed of three parts: a method for encapsulating datagrams over serial links, an extensible Link Control Protocol (LCP), and a family of Network Control Protocols (NCP) for establishing and configuring different network layer protocols.

The encapsulation scheme is provided by driver code in the kernel. `pppd` provides the basic LCP, authentication support, and an NCP for establishing and configuring the Internet Protocol (IP) called the IP Control Protocol (IPCP).

OPTIONS

`pppd` supports the following frequently used options:

`asyncmap` *map*

Set the async character map to *map*. The map describes which control characters cannot be sucessfully received over the serial line. `pppd` asks the peer to send these characters as a two-byte escape sequence. The argument is a 32–bit hex number with each bit representing a character to escape. Bit `0` (`00000001`) represents the character `0x00`; bit `31` (`80000000`) represents the character `0x1f` or `^_`. If multiple `asyncmap` options are given, the values are `OR`ed together. If no `asyncmap` option is given, no async character map is negotiated for the receive direction; the peer should then escape *all* control characters. To escape transmitted characters, use the `escape` option.

`auth`

Require the peer to authenticate itself before allowing network packets to be sent or received.

`connect` *command*

Run the executable *command* to set up the serial line. *command* typically uses **`chat`**(1M) to dial the modem and start the remote PPP session. This option is privileged if the `noauth` option is used.

`connect-max-attempts` *number*

Attempt dial-out connection to remote system no more than the specified *number* of times (default = `1`).If the connection is not made, `pppd` exits. Requires that `persist` be specified.

`crtscts`

Use hardware flow control, RTS/CTS, to control the flow of data on the serial port. If neither the `crscts` nor the `nocrscts` option is given, the hardware flow control setting for the serial port is left unchanged.

`defaultroute`

Add a default route to the system routing tables, using the peer as the gateway, when IPCP is successfullly completed. This entry is removed when the PPP connection is broken. This option is privileged if the `nodefaultroute` option is also specified.

`disconnect` *command*

Run the executable *command* after `pppd` has terminated the link. *command* must identify a user-supplied actor that is executed by `pppd` through an **afexec**(2K) call. *command* can, for example, issue commands to the modem to cause it to hang up if hardware modem control signals are not available. *command* is not run if the modem has already hung up. This option is privileged if the `noauth` option is used.

`escape` *xx,yy,...*

Specifies that certain characters should be escaped on transmission (regardless of whether the peer requests them to be escaped with its async control character map). The characters to be escaped are specified as a list of hex numbers separated by commas. Note that almost any character may be specified for the `escape` option, unlike the `asyncmap` option which only allows control characters to be specified. The characters that may not be escaped are those with hex values: `0x20`, `0x3f` and `0x5e`.

`file` *name*

Read options from the file *name*, whose format is specified below. The file must be readable by the user who invokes `pppd`.

`lock`

Specifies that `pppd` should create a UUCP–style lock file for the serial device to ensure exclusive access to the device.

`mru` *number*

Set the Maximum Receive Unit (MRU) value to *number*. `pppd` asks the peer to send packets of no more than *number* bytes. The minimum MRU value is

128. The default MRU value is 1500. A value of 296 is recommended for slow links (40 bytes for the TCP/IP header and 40 bytes for the data).

`mtu` *number*

Set the Maximum Transmit Unit (MTU) value to *number*. Unless a peer requests a smaller value via MRU negotiation, `pppd` requests that the kernel networking code send data packets of no more than *number* bytes through the PPP network interface.

`passive`

Enables the `passive` option in LCP. With this option, `pppd` attempts to initiate a connection. If no reply is received from the peer, `pppd` waits passively for a valid LCP packet from the peer, instead of exiting, as it would without the option.

`pppd` also supports the following options:

local_IP_address`:`*remote_IP_address*

Set the local and remote IP addresses. Either one may be omitted. The IP addresses can be specified with a host name, or in decimal dot notation such as `129.157.197.83`. The default local address is the first IP address of the system unless the `noipdefault` option is given. The remote address is obtained from the peer if not specified in any option. Therefore, in simple cases this option is not required. If a local or remote IP address is specified with this option, `pppd` does not accept a different value from the peer in the IPCP negotiation, unless the `ipcp-accept-local` or `ipcp-accept-remote` options are given.

`bsdcomp` *nr*`,`*nt*

Request that the peer compress packets that it sends, using the BSD-Compress scheme, with a maximum code size of *nr* bits, and agree to compress packets sent to the peer with a maximum code size of *nt* bits. If *nt* is not specified, it defaults to the value given for *nr*. Values in the range `9` to `15` may be used for *nr* and *nt*. Larger values give better compression but consume more kernel memory for compression dictionaries. Alternatively, a value of `0` for *nr* or *nt* disables compression in the corresponding direction. Use `nobsdcomp` or `bsdcomp 0` to disable BSD-Compress compression entirely.

`chap-interval` *number*

Challenge the peer every *number* seconds.

`chap-max-challenge` *number*

Set the maximum number of CHAP challenge transmissions to *number* (default = `10`).

`chap-restart` *number*

Set the CHAP restart interval — retransmission tim eout for challenges — to *number* seconds.

`debug`

Enable connection debugging facilities. If this option is given, `pppd` logs the contents of all control packets sent or received in a readable form. Packets are sent to `stdout` and the kernel log.

`default-asyncmap`

Disable asyncmap negotiation, forcing all control characters to be escaped in both the transmit and receive directions.

`default-mru`

Disable Maximum Receive Unit (MRU) negitiation. With this option, `pppd` uses the default MRU value of `1500` bytes in both the transmit and receive directions.

`deflate` *nr*,*nt*

Request that the peer compress packets that it sends, using the Deflate scheme, with a maximum window size of 2^{nr} bytes, and agree to compress packets sent to the peer with a maximum window size of 2^{nt} bytes. If *nt* is not specified, it defaults to the value given for *nr*. Values in the `8` to `15` range may be used for *nr* and *nt*. Larger values give better compression but consume more kernel memory for compression dictionaries. Alternatively, a value of `0` for *nr* or *nt* disables compression in the corresponding direction. Use `nodeflate` or `deflate 0` to disable Deflate compression entirely.

Note - `pppd` requests Deflate compression in preference to BSD-Compress if the peer can do either.

`demand`

Initiate the link only on demand, that is, when data traffic is present. With this option, the remote IP address must be specified by the user on the command line or in an options file. `pppd` initially configures the interface and enables it for IP traffic without connecting to the peer. When traffic is available, `pppd` connects to the peer and performs negotiation, authentication and so forth. When this is completed, `pppd` begins passing data packets, that is IP packets, across the link.

`domain` *d*

Append the domain name *d* to the local host name for authentication purposes. For example, if **`gethostname`**(2POSIX) returns the name `lethe`, but the fully qualified domain name is `lethe.France.Sun.COM`, you could specify `domain France.Sun.COM`. `pppd` would then use the name `lethe.France.Sun.COM` for looking up secrets in the secrets file, and as the default name to send to the peer when authenticating itself to the peer. This option is privileged.

`holdoff` *number*

Specifies the *number* of seconds to wait before re-initiating the link after it terminates. This option only has an effect if the `demand` or `persist` option is used. The holdoff period is not applid if the link terminates because it is idle.

`idle` *number*

Specifies that `pppd` should disconnect if the link is idle for *number* seconds. The link is idle when no data packets, that is IP packets, are being sent or received.

Note - It is not advisable to use this option with the `persist` option without the `demand` option. If the `active-filter` option is given, data packets that are rejected by the specified activity filter also count as the link being idle.

`ipcp-accept-local`

Accept the remote IP address specified by the peer, even if the local IP address is specified by another option.

`ipcp-accept-remote`

Accept the local IP address specified by the peer, even if the remote IP address is specified by another option.

`ipcp-max-configure` *number*

Set the maximum number of IPCP configure-request transmissions to *number* (default = `10`).

`ipcp-max-failure` *number*

Set the maximum number of IPCP configure-NAKs returned before starting to send configure-Rejects instead to *number* (default = `10`).

`ipcp-max-terminate` *number*

Set the maximum number of IPCP terminate-request transmissions to *number* (default = `3`).

`ipcp-restart` *number*

Set the IPCP restart interval to *number* seconds (default = `3`).

`kdebug` *number*

Enable debugging code in the kernel-level PPP driver. *number* is the sum of the following values: `1` to enable general debug messages, `2` to request that the contents of received packets be printed and `4` to request that the contents of transmitted packets be printed. Messages printed by the kernel are logged by **`syslog`**(1CC) to a file as directed in the `/etc/syslog.conf` configuration file.

`lcp-echo-failure` *number*

Presume the peer to be dead if *number* LCP requests are sent without receiving a valid LCP echo-reply. If this happens, `pppd` terminates the connection. Use of this option requires a non-zero value for `lcp-echo-interval`. This option can be used to enable `pppd` to terminate after the physical connection has been broken, such as after the modem hangs up, in situations where no hardware modem control lines are available.

`lcp-echo-interval` *number*

Send an LCP echo-request frame to the peer every *number* seconds. Normally, the peer should respond to the echo-request by sending an echo-reply. This option can be used with `lcp-echo-failure` to detect that the peer is no longer connected.

`lcp-max-configure` *number*

Set the maximum number of LCP configure-request transmissions to *number* (default = `10`).

`lcp-max-failure` *number*

Set the maximum number of LCP configure-NAKs returned before starting to send configure-Rejects instead to *number* (default = `10`).

`lcp-max-terminate` *number*

Set the maximum number of LCP terminate request transmissions to *number* (default = `3`).

`lcp-restart` *number*

Set the LCP restart interval (transmission timeout) to *number* seconds (default = `3`).

`local`

Do not use the modem control lines. With this option, `pppd` ignores the state of the Carrier Detect (CD) signal from the modem and does not change the state of the Data Terminal Ready (DTR) signal.

`maxconnect` *number*

Terminate the connection when it has been available for network traffic for *number* seconds, that is, *number* seconds after the first network protocol comes up.

`modem`

Use the modem control lines. This option is the default. With this option, `pppd` waits for the Carrier Detect (CD) signal from the modem to be asserted when opening the serial device (unless a connect command is specified), and drops the Data Terminal Ready (DTR) signal briefly when the connection is terminated and before executing the connection command.

`ms-dns` *address*

Allow `pppd` to supply one or two Domain Name Server (DNS) addresses to Microsoft Windows clients. The first instance of this option specifies the

primary DNS address. The second instance, if given, specifies the secondary DNS address.

Note - This option was present in some older versions of `pppd` under the name `dns-addr`.

`ms-wins` *address*

Allow `pppd` to supply one or two Windows Internet Name Services(WINS) addresses to Samba or Microsoft Windows clients. The first instance of this option specifies the primary WINS address. The second instance, if given, specifies the secondary WINS address.

`name` *name*

Set the name of the local system for authentication purposes to *name*. This is a privileged option. With this option, `pppd` uses lines in the secrets files having *name* as the second field when looking for a secret to use in authenticating with the peer. In addition, unless overridden by the `user` option, *name* is used as the name to send to the peer when authenticating the local system to the peer.

Note - `pppd` does not append the domain name to *name*.

`netmask` *number*

Set the interface netmask to *number*, a 32–bit netmask in decimal dot notation, such as `255.255.255.0`. If this option is given, the value specified is `OR`ed with the default netmask. The default netmask is chosen based on the negotiated remote IP address. It is the appropriate network mask for the class of the remote IP address, `OR`ed with the netmasks for any non-point-to-point network interfaces in the system that are not on the same network.

`noaccomp`

Disable Address/Control compression in both send and receive directions.

`noauth`

Do not require the peer to authenticate itself. This option is privileged if the `auth` option is specified in the `options` file.

`nobsdcomp`

Disable BSD-Compress compression. pppddoes not request or agree to compress packets using the BSD-Compress scheme.

noccp

Disable Compression Control Protocol (CCP) negotiation. This option should only be required if the peer is buggy and requests from pppd for CCP negotiation cause problems.

nocrtscts

Disable hardware flow control, RTS/CTS, on the serial port. If neither the crtscts nor nocrtscts option is given, the hardware flow control setting for the serial port is left unchanged.

nodefaultroute

Disable the defaultroute option. The system administrator who wishes to prevent users from creating default routes with pppd can do so by placing this option in the options file, which is either /dev/archive/options or /etc/ppp/options.

nodeflate

Disable Deflate compression. pppd does not request or agree to compress packets using the Deflate scheme.

nodetach

Do not detach from the controlling terminal. Without this option, if a serial device other than the terminal on the standard input is specified, pppd forks to become a background process.

noip

Disable IPCP negotiation and IP communication. This option should only be required if the peer is buggy requests from pppd for IPCP negotiation causse problems.

noipdefault

Disables the default behavior when no local IP address is specified., which is to determine, if possible, the local IP address from the hostname. With this option, the peer must supply the local IP address during IPCP negotiation unless it is specified explicitly on the command line or in an options file.

`nomagic`

Disable magic number negotiation. With this option, `pppd` cannot detect a looped-back line. This option should only be required if the peer is buggy.

`nopcomp`

Disable protocol field compression negotiation in both send and receive directions.

`nopersist`

Exit once a connection has been made and terminated. This is the default behavior unless the `demand` or `persist` option has been specified.

`nopredictor1`

Do not accept or agree to Predictor-1 compression.

`noproxyarp`

Disable the `proxyarp` option. The system administrator who wishes to prevent users from creating proxy ARP entries with `pppd` can do so by placing this option in `/etc/ppp/options`.

`novj`

Disable Van Jacobson-style TCP/IP header compression in both send and receive directions.

`novjccomp`

Disable the connection-ID compression option in Van Jacobson-style TCP/IP header compression. With this option, `pppd` does not omit the connection-ID byte from Van Jacobson-style TCP/IP headers, nor does it ask the peer to do so.

`papcrypt`

Indicates that all secrets in the `/etc/ppp/pap-secrets` file used for checking the identity of the peer are encrypted, and therefore `pppd` should not accept a password which, before encryption, is identical to the secret from the `/etc/ppp/pap-secrets` file.

`pap-max-authreq` *number*

Set the maximum number of PAP authenticate-request transmissions to *number* (default = `10`).

`pap-restart` *number*

Set the PAP restart interval to *number* seconds (default = `3`).

`pap-timeout` *number*

Set the maximum time that `pppd` will wait for the peer to authenticate itself with PAP to *number* seconds (`0` indicates no limit).

`persist`

Do not exit after a connection is terminated. Instead, try to open the connection again.

`predictor1`

Request that the peer compress frames sent using Predictor-1 compression, and agree to compress transmitted frames with Predictor-1, if requested. This option has no effect unless the kernel driver supports Predictor-1 compression.

`proxyarp`

Add an entry to the local system Address Resolution Protocol (ARP) table with the IP address of the peer and the Ethernet address of this system. This has the effect of making the peer appear to other systems on the local Ethernet.

`remotename` *name*

Set the assumed name of the remote system for authentication purposes to *name*.

`refuse-chap`

Do not agree to authenticate the local system to the peer using CHAP.

`refuse-pap`

Do not agree to authenticate the local system to the peer using PAP.

`require-chap`

Require the peer to authenticate itself using Challenge Handshake Authentication Protocol (CHAP) authentication.

`require-pap`

Require the peer to authenticate itself using Password Authentication Protocol (PAP) authentication.

`silent`

Do not transmit LCP packets to initiate a connection until a valid LCP packet is received from the peer, similar to the `passive` option for older versions of `pppd`.

`usehostname`

Enforce the use of the hostname with the domain name appended, if available, as the name of the local system for authentication purposes. Overrides the `name` option.

`user` *name*

Sets the name used for authentication of the local system to the peer to *name*.

`vj-max-slots` *number*

Sets the number of connection slots to be used by Van Jacobson-style TCP/IP header compression and decompression code to a *number* between `2` and `16`, inclusive.

`welcome` *script*

Run the executable or shell command specified by *script* before initiating PPP negotiation, after the connect script, if any, has completed. This option is privileged if the `noauth` option is used.

`xonxoff`

Use software flow control, XON/XOFF, to control the flow of data on the serial port.

OPERANDS

`pppd` supports the following operands:

tty_name Communicate over the named device. The string `/dev/` is prepended if necessary. This operand is mandatory.

speed Set the baud rate to *speed*, a decimal number. The default *speed* is `9600`.

OPTIONS FILES

Options can be taken from files as well as the command line. `pppd` reads options from `options` files found in `/dev/archive` or `/etc/ppp`, respectively, before processing the options on the command line.

An `options` file is parsed into a series of words, delimited by whitespace. Whitespace can be included in a word by enclosing the word in double-quotes ("). A backslash (\) quotes the following character. A hash (#) starts a comment, which continues until the end of the line. No restriction exists to prevent the use of `file` or `call` options within an `options` file.

SECURITY

`pppd` provides system administrators with sufficient access control that PPP access to a server machine can be provided to legitimate users without fear of compromising the security of the server or the network it's on. In part this is provided by the `options` file, where the administrator can place options to restrict the ways in which `pppd` can be used, and in part by the PAP and CHAP secrets files, where the administrator can restrict the set of IP addresses which individual users may use.

AUTHENTICATION

Authentication is the process whereby one peer convinces the other of its identity. This involves the first peer sending its name to the other, together with some kind of secret information which could only come from the genuine authorized user of that name. In such an exchange, the first peer is called the "client" and the other the "server". The client has a name by which it identifies itself to the server, and the server also has a name by which it identifies itself to the client. Generally, the genuine client shares some secret (or password) with the server, and authenticates itself by proving that it knows that secret. Very often, the names used for authentication correspond to the internet hostnames of the peers, but this is not essential.

At present, `pppd` supports two authentication protocols: the Password Authentication Protocol (PAP) and the Challenge Handshake Authentication Protocol (CHAP). PAP involves the client sending its name and a cleartext password to the server to authenticate itself. In contrast, the server initiates the CHAP authentication exchange by sending a challenge to the client (the challenge packet includes the name of the server). The client must respond with a response which includes its name plus a hash value derived from the shared secret and the challenge, in order to prove that it knows the secret.

The PPP protocol, being symmetrical, allows both peers to require the other to authenticate itself. In that case, two separate and independent authentication exchanges will occur. The two exchanges could use different authentication protocols, and in principle, different names could be used in the two exchanges.

The default behavior of pppd is to agree to authenticate if requested, and to not require authentication from the peer. However, pppd will not agree to authenticate itself with a particular protocol if it has no secrets which could be used to do so.

pppd stores secrets for use in authentication in secrets files (pap.scr for PAP, chap.scr for CHAP found in either /dev/archive, if included in the system image, or /etc/ppp, respectively). Both secrets files have the same format. The secrets files can contain secrets for pppd to use in authenticating itself to other systems, as well as secrets for pppd to use when authenticating other systems to itself.

Each line in a secrets file contains one secret. A given secret is specific to a particular combination of client and server - it can only be used by that client to authenticate itself to that server. Thus each line in a secrets file has at least three fields: the name of the client, the name of the server, and the secret. These fields may be followed by a list of the IP addresses that the specified client may use when connecting to the specified server.

A secrets file is parsed into words as for an options file, so the client name, server name and secrets fields must each be one word, with any embedded spaces or other special characters quoted or escaped. Any following words on the same line are taken to be a list of acceptable IP addresses for that client, or an override for *local*:*remote* addresses (the same format used on the command line or in the options file) when on a line that contains a specific client name (not a wildcard nor empty). If there are only three words on the line, or if the first word is -, then all IP addresses are disallowed. To allow any address, use *. A word starting with ! indicates that the specified address is not acceptable. An address may be followed by / and a *number*, to indicate a whole subnet, such as all addresses which have the same value in the most significant *number* bits. Note that case is significant in the client and server names and in the secret.

If the secret starts with an @, what follows is assumed to be the name of a file from which to read the secret. A * as the client or server name matches any name. When selecting a secret, pppd takes the best match, that is, the match with the fewest wildcards.

Thus a secrets file contains both secrets for use in authenticating other hosts, plus secrets which we use for authenticating ourselves to others. When pppd is authenticating the peer (checking the peer's identity), it chooses a secret with the peer's name in the first field and the name of the local system in the second field. The name of the local system defaults to the hostname, with the domain name appended if the domain option is used. This default can be overridden with the name option, except when the usehostname option is used.

When pppd is choosing a secret to use in authenticating itself to the peer, it first determines what name it is going to use to identify itself to the peer. This name can be specified by the user with the user option. If this option is not used, the name defaults to the name of the local system, determined as described in the previous paragraph. Then pppd looks for a secret with this name in the first field and the peer's name in the second field. pppd will know the name of the peer if CHAP authentication is being used, because the peer will have sent it in the challenge packet. However, if PAP is being used, pppd will have to determine the peer's name from the options specified by the user. The user can specify the peer's name directly with the remotename option. Otherwise, if the remote IP address was specified by a name (rather than in numeric form), that name will be used as the peer's name. Failing that, pppd will use the null string as the peer's name.

When authenticating the peer with PAP, the password supplied is first compared with the secret from the secrets file. If the password doesn't match the secret, the password is encrypted using **crypt()** and checked against the secret again. Thus secrets for authenticating the peer can be stored in encrypted form if desired. If the papcrypt option is given, the first (unencrypted) comparison is omitted, for better security.

Authentication must be satisfactorily completed before IPCP (or any other Network Control Protocol) can be started. If the peer is required to authenticate itself, and fails to do so, pppd will terminated the link (by closing LCP). If IPCP negotiates an unacceptable IP address for the remote host, IPCP will be closed. IP packets can only be sent or received when IPCP is open.

In some cases it is desirable to allow some hosts which can't authenticate themselves to connect and use one of a restricted set of IP addresses, even when the local host generally requires authentication. If the peer refuses to authenticate itself when requested, pppd treats it as equivalent to authenticating with PAP using the empty string for the username and password. Thus, by adding a line to the pap-secrets file which specifies the empty string for the client and password, it is possible to allow restricted access to hosts which refuse to authenticate themselves.

ROUTING

When IPCP negotiation is completed successfully, pppd will inform the kernel of the local and remote IP addresses for the PPP interface. This is sufficient to create a host route to the remote end of the link, which will enable the peers to exchange IP packets. Communication with other machines generally requires further modification to routing tables or ARP (Address Resolution Protocol) tables. In most cases the defaultroute or proxyarp options are sufficient for this.

Sometimes it is desirable to add a default route through the remote host, as in the case of a machine whose only connection to the Internet is through the

PPP interface. The `defaultroute` option causes `pppd` to create this type of default route when IPCP comes up, and delete it when the link is terminated.

In some cases it is desirable to use a proxy ARP, for example on a server machine connected to a LAN, in order to allow other hosts to communicate with the remote host. The `proxyarp` option causes `pppd` to look for a network interface on the same subnet as the remote host (an interface supporting broadcast and ARP, which is up and not a point-to-point or loopback interface). If found, `pppd` creates a permanent, published ARP entry with the IP address of the remote host and the hardware address of the network interface found.

When the `demand` option is used, the interface IP addresses have already been set at the point when IPCP comes up. If `pppd` has not been able to negotiate the same addresses that it used to configure the interface (for example when the peer is an ISP that uses dynamic IP address assignment), `pppd` has to change the interface IP addresses to the negotiated addresses. This may disrupt existing connections, and the use of demand dialing with peers that do dynamic IP address assignment is not recommended.

EXAMPLES

The following examples assume that the `/dev/archive/options` file contains the `auth` option.

```
pppd /dev/tty01 connect "/dev/archive/chat –v –f /dev/archive/chat.cmd"
```

In this example, chat is used to dial the modem. `/dev/archive/chat.cmd` contains the `chat` commands. It could contain something like this, for example:

```
ABORT        BUSY
ABORT        'NO CARRIER'
TIMEOUT      5
''           ATZ
OK           ATDT10009
TIMEOUT      40
CONNECT
```

See the **`chat`**(1M) man page for details regarding `chat` scripts.

If your serial connection is any more complicated that a piece of wire, you may need to arrange for some control characters to be escaped. In particular, it is often useful to escape XON (^Q) and XOFF (^S), using `asyncmap a0000`. If the path includes a telnet, you should probably escape ^] as well, using `asyncmap 200a0000`.

DIAGNOSTICS

Messages are sent to `stdout` and to the kernel log, which can be viewed using the kernel debugger, `kdb`.

The `debug` option causes the contents of all control packets sent or received to be logged, that is, all LCP, PAP, CHAP or IPCP packets. This can be useful if the PPP negotiation does not succeed or if authentication fails. If debugging is enabled at compile time, the `debug` option also causes other debugging messages to be logged.

FILES

`pppd` looks for files first in `/dev/archive`, and then in `/etc/ppp`. The choice is abbreviated below as *directory*.

directory`/chap.scr`

Names, secrets and IP addresses for CHAP authentication. This file should be owned by root and not readable or writable by any other user. `pppd` logs a warning if this is not the case.

directory`/options`

System default options for `pppd`. Read before command-line options.

directory`/pap.scr`

Usernames, secrets and IP addresses for PAP authentication. This file should be owned by root and not readable or writable by any other user. `pppd` logs a warning if this is not the case.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`chat`(1M)

RFC1144 Jacobson, V. *Compressing TCP/IP headers for low-speed serial links*. February 1990.

RFC1321 Rivest, R. *The MD5 Message-Digest Algorithm*. April 1992.

RFC1332 McGregor, G. *PPP Internet Protocol Control Protocol (IPCP)*. May 1992.

RFC1334 Lloyd, B. Simpson, W. A. *PPP authentication protocols*. October 1992.

RFC1661 Simpson, W. A. *The Point-to-Point Protocol (PPP)*. July 1994.

RFC1662 Simpson, W. A. *PPP in HDLC-like Framing*. July 1994.

NAME

route – manipulate the routing tables manually

SYNOPSIS

route [–nqv] *command* [[*modifiers*] *args*]

DESCRIPTION

`route` is a utility used to manually manipulate the network routing tables.

The `route` utility supports a limited number of general options, but a rich command language, enabling the user to specify any arbitrary request that could be delivered via the programmatic interface discussed in **`route`**(7P).

`route` supports the following options:

`–n` Bypass attempts to print host and network names symbolically when reporting actions. (The process of translating between symbolic names and numerical equivalents can be quite time consuming, and may require correct operation of the network; thus it may be expedient to forgot this, especially when attempting to repair networking operations).

`–q` Suppress all output.

`–v` (verbose) Print additional details.

The route utility provides six *commands*:

add Add a route.

flush Remove all routes.

delete Delete a specific route.

change Change aspects of a route, such as its gateway.

get Lookup and display the route for a destination.

monitor Continuously report any changes to the routing information base, routing lookup misses, or suspected network partitionings.

This *command* is not available in the `C_INIT` built-in `route` utility.

The `monitor` command has the syntax:

route [–n] monitor

The `flush` command has the syntax:

route [–n] flush [*family*]

If the `flush` command is specified, `route` will "flush" the routing tables of all gateway entries. When the address family may is specified by any of the –`osi`, –`xns`, –`atalk`, or –`inet` modifiers, only routes having destinations with addresses in the delineated *family* will be deleted.

The other commands have the following syntax:

route [–`n`] *command* [–`net` | –`host`] *destination gateway*

where *destination* is the destination host or network, *gateway* is the next-hop intermediary via which packets should be routed. Routes to a particular host may be distinguished from those to a network by interpreting the Internet address specified as the destination argument. The optional modifiers –`net` and –`host` force the destination to be interpreted as a network or a host, respectively. Otherwise, if the destination has a "local address part" of `INADDR_ANY` (`0.0.0.0`), or if the destination is the symbolic name of a network, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host.

For example, `128.32` is interpreted as –`host 128.0.0.32`; `128.32.130` is interpreted as –`host 128.32.0.130`; –`net 128.32` is interpreted as `128.32.0.0`; and –`net 128.32.130` is interpreted as `128.32.130.0`.

A destination of `default` is a synonym for –`net 0.0.0.0`, which is the default route.

If the destination is directly reachable via an interface requiring no intermediary system to act as a gateway, the interface modifier should be specified; the gateway given is the address of this host on the common network, indicating the interface to be used for transmission. Alternately, if the interface is point to point the name of the interface itself may be given, in which case the route remains valid even if the local or remote addresses change.

The optional modifiers -xns, -osi, -atalk, and -link specify that all subsequent addresses are in the XNS, OSI, or AppleTalk address families, or are specified as link-level addresses, and the names must be numeric specifications rather than symbolic names.

The optional -netmask modifier is intended to achieve the effect of an OSI ESIS redirect with the netmask option, or to manually add subnet routes with netmasks different from that of the implied network interface (as would otherwise be communicated using the OSPF or ISIS routing protocols). One specifies an additional ensuing address parameter (to be interpreted as a network mask). The implicit network mask generated in the AF_INET case can be overridden by making sure this option follows the destination parameter.

Routes have associated flags which influence operation of the protocols when sending to destinations matched by the routes. These flags may be set (or sometimes cleared) by indicating the following corresponding modifiers:

–cloning	RTF_CLONING; generates a new route on use
–xresolve	RTF_XRESOLVE; emit message on use (for external lookup)
–iface	~RTF_GATEWAY; destination is directly reachable
–static	RTF_STATIC; manually added route
–nostatic	~RTF_STATIC; pretend route added by kernel or daemon
–reject	RTF_REJECT; emit an ICMP unreachable when matched
–blackhole	RTF_BLACKHOLE; silently discard packets (during updates)
–proto1	RTF_PROTO1; set protocol specific routing flag number 1
–proto2	RTF_PROTO2; set protocol specific routing flag number 2
–llinfo	RTF_LLINFO; validly translates proto address to link address

The optional modifiers –rtt, –rttvar, –sendpipe, –recvpipe, –mtu, –hopcount, –expire, and –ssthresh provide initial values to quantities maintained in the routing entry by transport level protocols, such as TCP or TP4. These may be individually locked by preceding each such modifier to be locked by the –lock meta-modifier, or one can specify that all ensuing metrics may be locked by the –lockrest meta-modifier.

In a change or add command where the destination and gateway are not sufficient to specify the route (as in the ISO case where several interfaces may have the same address), the –ifp or –ifa modifiers may be used to determine the interface or interface address.

All symbolic names specified for a destination or gateway are looked up first as a host name using **gethostbyname**(3STDC). If this lookup fails, **getnetbyname**(3STDC) is then used to interpret the name as that of a network.

`route` uses a routing socket and the new message types `RTM_ADD`, `RTM_DELETE`, `RTM_GET`, and `RTM_CHANGE`. As such, only the super-user may modify the routing tables.

DIAGNOSTICS

add [host | network] %s: gateway %s flags %x

The specified route is being added to the tables. The values printed are from the routing table entry supplied in the **`ioctl`**(2POSIX) call. If the gateway address used was not the primary address of the gateway (the first one returned by **`gethostbyname`**(3STDC)), the gateway address is printed numerically as well as symbolically.

delete [host | network] %s: gateway %s flags %x

As above, but when deleting an entry.

%s %s done

When the `flush` command is specified, each routing table entry deleted is indicated with a message of this form.

Network is unreachable

An attempt to add a route failed because the gateway listed was not on a directly-connected network. The next-hop gateway must be given.

not in table

A delete operation was attempted for an entry which was not present in the tables.

routing table overflow

An add operation was attempted, but the system was low on resources and was unable to allocate memory to create the new entry.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`C_INIT`(1M), **`route`**(7P)

NAME

shutdown – shut down and reboot system, change system state

SYNOPSIS

shutdown [–i *state*]

DESCRIPTION

shutdown can be executed by Trusted and Supervisor actors to change the state of the system. In most cases, it is used to bring the system down safely and reboot.

By default, shutdown brings the system down to a state in which all actors have stopped running, and then reboots the system. The default behavior can also be achieved using the following command:

$ **rsh** *target* **shutdown -i 0**

System states are:

0 Shut down the entire system safely and reboot.

1 Shut down the entire system safely and restart the site without rebooting, such that hot restartable actors are automatically restarted. This state is designed for use with hot restart functionality and requires the HOT_RESTART feature.

2 Kill all actors known to the AM and IOM except for invokers, and shut down the AM, the IOM and the invokers.

3 Kill all actors known to the AM except for invokers, and shut down the AM and the invokers.

OPTIONS

–i *state* Specifies the system state to which to change.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

C_INIT(1M)

NOTES

Before changing the state of the system to *state*, shutdown tests the environment variable, OS_CONF, to determine whether the actors to be killed are present.

NAME

slattach – attach serial lines as network interfaces

SYNOPSIS

slattach [–a] [–c] [–h] [–K *keepalive*] [–l] [–n] [–O *outfill*] [–r *redial-command*] [–S *unit*] [–s *baudrate*] [–z] {*ttyname*}

DESCRIPTION

`slattach` is used to assign a tty line to a network interface, and to define the network source and destination addresses.

Only the superuser may attach a network interface.

OPERANDS

`slattach` supports the following operands:

–a	Auto-enable the VJ header compression option. If the other end of the link is capable of VJ header compression, then it is used. Otherwise, normal headers are used.
–c	Enables the VJ header compression option. Note that both ends of the link must be able to use VJ header compression for this to work.
–h	Turns on CTS/RTS–style flow control on the SLIP port. By default, no control is performed.
–K *keepalive*	Set SLIP *keepalive* time-out in seconds. If `FRAME_END` is not received in this amount of time, reconnect occurs. The default value is no time-out.
–l	Disables modem control (`CLOCAL`) and ignores the carrier detected on the SLIP port. By default, the *redial-command* is invoked upon carrier drop and `slattach` aborts if no *redial-command* is specified.
–n	Throw away ICMP packets. The SLIP interface ignores ICMP packets to prevent slow lines from being saturated by ICMP responses.
–O *outfill*	Set SLIP *outfill* time-out in seconds. It forces at least one `FRAME_END` to be sent during this time period, which is necessary for the *keepalive* time-out on the remote side. The default value is no time-out.
–r *redial-command*	Specifies a command to be invoked within a shell `sh –c` *redial-command* whenever the carrier is lost

	on the modem line. An empty *redial-command* (–r "") causes the connection to be reestablished on a leased line without any external command being invoked.
–S *unit*	Set the SLIP unit number directly. Use this option with caution, because no check is made for two interfaces with the same number.
–s *baudrate*	Specifies the speed of the connection. If not specified, the default speed of 9600 is used.
–z	Forces redial using *redial-command* on start-up, irrespective of the carrier.
ttyname	Specifies the name of the tty device. *ttyname* should be a string of the form or /dev/tty*XX*.

EXTENDED DESCRIPTION

To setup slattach to redial the phone when the carrier is lost, use the –r *redial-command* option to specify a script or executable that reconnects the serial line to the SLIP server. For example, the script could redial the server, log in and so forth.

If you use a hard-wired connection rather than a modem, invoke slattach with the –l option in order to ignore the carrier on the SLIP line.

EXAMPLES

The following invokes slattach with the default options on /dev/tty01.

```
slattach /dev/tty01
```

host% **rsh** *target* **slattach /dev/tty01**

The following invoke slattach on tty01 at 4800 bauds.

```
slattach -s 4800 /dev/tty01
```

host% **rsh** *target* **slattach –s 4800 /dev/tty01**

The following invoke slattach on tty01 at 38400 bauds with header compression.

```
slattach -c -s 38400 /dev/tty01
```

host% **rsh** *target* **slattach –c –s 38400 /dev/tty01**

The following invoke `slattach` with a script used to redial when the carrier is lost.

```
slattach -z -r '/bin/chat -f dial.chat' /dev/tty01
```

host% **rsh** *target* **slattach –z –r "'/bin/chat -f dial.chat'" /dev/tty01**

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

chat(1M), **ifconfig**(1M)

NAME

swapon – specify additional device for paging and swapping

SYNOPSIS

swapon /swap

FEATURES

FS_MAPPER, ON_DEMAND_PAGING, VIRTUAL_ADDRESS_SPACE

DESCRIPTION

The built-in **C_INIT**(1M) command, `swapon`, is used to specify an additional local device on which paging and swapping are to take place. Calls to `swapon` occur in the system initialization file `sysadm.ini` making the swap device available.

EXTENDED DESCRIPTION

Before `swapon` is used, the disk must be labelled correctly using **disklabel**(1M), and a swap partition must be mounted using thebuilt-in `C_INIT` command, `mount`.

`disklabel` uses entries in `/etc/disktab`. The `/etc/disktab` entry for a swap partition in must specify the type as `swap`. For example:

```
ST34310A:\
        :dt=ST506:ty=fixed:se#512:nt#16:ns#63:nc#8354:sf \
        :pa#60480:oa#0:ta=ufs: \
        :pb#65536:ob#60480:tb=swap: \
        :pc#8420832:oc#0:tc=unused: \
        :ph#8294816:oh#126016:th=ufs:
```

After the disk has been labelled correctly, the `mount` and `swapon` commands are used to mount a swap partition and activate it, respectively. The following example could be used in the system initialization file, `sysadm.ini`:

```
# Mount a swap partition on the second IDE hard disk partition,
# previously labelled using disklabel, on /swap.
#
mount -t swap /dev/hd0b /swap
#
# Activate the swap partition.
#
swapon /swap
```

Note that the root file system must be mounted before the above commands can work.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

C_INIT(1M), **disklabel**(1M), **mount**(1M), **sysadm.ini**(4CC)

NOTES

Contrary to other implementations that you may be familiar with, the implementation of swapon in this release *does not* support the –a option.

LIMITATIONS

Only one swap partition may be used, and the swap partition must be on a local device.

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

This release does not allow the use of a named file as a swap partition.

NAME

syncd – Update disks periodically

SYNOPSIS

syncd [x | 0]

DESCRIPTION

The *syncd* daemon flushes disks periodically. By default, when executed without any arguments, it flushes disks every 10 seconds.

It is possible to overwrite this default period when *syncd* is called as follows:

```
syncd 40 &
```

In this case disks are updated every 40 seconds.

It is also possible to flush once and exit when *syncd* is called as follows:

```
syncd 0
```

In this case disks are flushed once and the daemon kills itself.

DIAGNOSTICS

This daemon exits and displays an error message if a negative argument is given.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

sysctl – get or set kernel state

SYNOPSIS

sysctl [–bn] *name...*

sysctl [–bn] –w *name=value...*

sysctl [–bn] –AaX

DESCRIPTION

`sysctl` retrieves the kernel state and allows processes with appropriate privilege to set the kernel state. The state to be retrieved or set is described using a Management Information Base (MIB)–style name, described as a dotted set of components.

OPTIONS

`sysctl` takes the following options:

–A
: List all the known MIB names including opaques. Those with string or integer values are printed as with the –a option. For opaque values, information about the format and the length is printed in addition to the first few bytes, which are dumped in hexadecimal form.

–a
: List all the currently available string or integer values.

–b
: Force the value of the variable(s) to be output in raw, binary format. No names are printed and no terminating newlines are output. This is mostly useful with a single variable.

–n
: Specify that the printing of the field name should be suppressed and that only its value should be output. This flag is useful for setting shell variables. For example, to save the page size in variable `psize`, use:

```
set psize=‘sysctl –n hw.pagesize‘
```

–w *name=value*
: Set the MIB name to the new value. If just a MIB style name is given, the corresponding value is retrieved.

–X
: Same as –A except the entire value of opaque variables is dumped in hexadecimal form.

EXTENDED DESCRIPTION

The information available using `sysctl` consists of integers, strings, and opaques. `sysctl` only knows about a couple of opaque types, and will resort to hexdumps for the rest. The opaque information is much more useful if retrieved by special purpose programs such as `ps`, `systat`, and `netstat`.

The string and integer information is summarized below. The changeable column indicates whether a process with appropriate privileges can change the value.

Name	Type	Changeable
kern.ostype	string	no
kern.osrelease	string	no
kern.osrevision	integer	no
kern.version	string	no
kern.maxvnodes	integer	yes
kern.maxproc	integer	yes
kern.maxprocperuid	integer	yes
kern.maxfiles	integer	yes
kern.maxfilesperproc	integer	yes
kern.argmax	integer	no
kern.securelevel	integer	raise only
kern.hostname	string	yes
kern.hostid	integer	yes
kern.clockrate	struct	no
kern.posix1version	integer	no
kern.ngroups	integer	no
kern.job_control	integer	no
kern.saved_ids	integer	no
kern.boottime	struct	no
kern.domainname	string	yes
kern.update	integer	yes
kern.osreldate	string	no
kern.bootfile	string	yes
vm.loadavg	struct	no

Name	Type	Changeable
hw.machine	string	no
hw.model	string	no
hw.ncpu	integer	no
hw.byteorder	integer	no
hw.physmem	integer	no
hw.usermem	integer	no
hw.pagesize	integer	no
hw.floatingpoint	integer	no
hw.machine_arch	string	no
machdep.console_device	dev_t	no
machdep.adjkerntz	integer	yes
machdep.disable_rtc_set	integer	yes
user.cs_path	string	no
user.bc_base_max	integer	no
user.bc_dim_max	integer	no
user.bc_scale_max	integer	no
user.bc_string_max	integer	no
user.coll_weights_max	integer	no
user.expr_nest_max	integer	no
user.line_max	integer	no
user.re_dup_max	integer	no
user.posix2_version	integer	no
user.posix2_c_bind	integer	no
user.posix2_c_dev	integer	no
user.posix2_char_term	integer	no
user.posix2_fort_dev	integer	no
user.posix2_fort_run	integer	no
user.posix2_localedef	integer	no
user.posix2_sw_dev	integer	no
user.posix2_upe	integer	no

Name	Type	Changeable
user.stream_max	integer	no
user.tzname_max	integer	no

EXAMPLES

EXAMPLE 1 Getting the Maximum Number of Processes

The following example gets the maximum number of processes allowed in the system:

```
sysctl kern.maxproc
```

EXAMPLE 2 Setting the Maximum Number of Processes

The following example sets the maximum number of processes allowed in the system to 1000:

```
sysctl -w kern.maxproc=1000
```

EXAMPLE 3 Getting System Clock Rate Information

The following example retrieves the system clock rate:

```
sysctl kern.clockrate
```

EXAMPLE 4 Getting Load Average History

The following example retrieves the load average history for the system:

```
sysctl vm.loadavg
```

More variables than these are available, and the best and likely only place to find their deeper meanings is undoubtedly the source where they are defined.

FILES

For more information see the following files:

`<netinet/icmp_var.h>`	Definitions for fourth level ICMP identifiers
`<netinet/in.h>`	Definitions for third level Internet identifiers and fourth level IP identifiers
`<netinet/udp_var.h>`	Definitions for fourth level UDP identifiers
`<sys/gmon.h>`	Definitions for third level profiling identifiers
`<sys/socket.h>`	Definitions for second level network identifiers
`<sys/sysctl.h>`	Definitions for top level identifiers, second level kernel and hardware identifiers, and user level identifiers

`<vm/vm_param.h>`	Definitions for second level virtual memory identifiers.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`sysctl`(3POSIX)

BUGS

`sysctl` presently exploits an undocumented interface to the kernel **sysctl()** facility to traverse the `sysctl` tree and to retrieve format and name information. The correct interface is being discussed.

NAME

sysenv – ChorusOS system environment

SYNOPSIS

This manual page describes the configurable system environment parameters of a ChorusOS system.

DESCRIPTION

A ChorusOS system is built to allow the user to change certain parameters at system build or boot time. This page describes the system environment variables used in a standard ChorusOS system. These variables can be set or modified using the *configurator*(1CC) utility (or the graphical configurator commands), at system build time, and can be modified at boot time using the netboot configuration files or the boot PROM, depending on the target.

The current system environment can be displayed at any time using *cs*(1CC) with the *-lE* option, and read or modified using the *sysGetEnv*(2K) and *sysSetEnv*(2K) system calls.

Some environment variables are for information only, (they may be set, ususally by the boot procedure) but are not used for system configuration by ChorusOS.

ChorusOS Environment

The ChorusOS contains the following system environment variables which may be modified:

BOOTFILE The boot file name (the path of the system archive that was booted on the target system) for information only.

BOOTSERVER The Internet address (in dotted decimal notation) of the host system where BOOTFILE was booted, if any, for information only.

HOST The hostname of the target system, for information only.

LOCAL_INADDR This parameter is used by the file manager to determine the target's Internet address in dotted decimal notation. By default, the file manager determines the address using the RARP protocol.

GATEWAY The Internet address of the routing gateway, if any.

IP_MASK This parameter is used by the file manager to determine the Internet network mask (8 hexadecimal digits). By default, the file manager determines the network mask according to the target's Internet address. To use the remote CHORUS IPC, the network mask must have a

maximum of 11 null bits (it must be at least fffff800).

ROOT_HOSTADDR The Internet address (in dotted decimal notation) of the host system which provides the default root file system for the target system.

ROOT_HOSTNAME The name of the host system which provides the default root file system for the target system.

ROOT_PATHNAME The location of the target's root file system on the host.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

sysGetEnv(2K), **sysSetEnv**(2K), **configurator**(1CC)

NAME

telnetd – Telnet Protocol server

SYNOPSIS

/etc/teld_s

DESCRIPTION

The `telnetd` command is a server which supports the *DARPA* standard *TELNET* virtual terminal protocol.

When a TELNET session starts up, `telnetd` sends TELNET options to the client side indicating availability to do *remote echo* of characters, and to *suppress go ahead*. The state of the virtual terminal is configured to operate in line mode.

The `telnetd` daemon can perform the following functions:`echo`, *binary*, *suppress go ahead*, and *timing mark*. It allows the remote client to perform: *binary* and *suppress go ahead*.

It must be started once with the following command:

```
% rsh -n <target> arun /etc/teld_s &
```

Subsequently, each telnet connection will spawn a new *teld_s* actor which will act as a *C_INIT*(1M) command shell.

The `telnetd` command authenticates users according to two rules:

Secure mode — The login name must be in the security data base, */etc/security* , and not have a null password (see *security*(4CC)). In this case, a password must be provided by the client.

Non-secure mode — In this case, the */etc/security* file is not present, no password check is performed and the user has the default credential values (see *conf*(1CC)).

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`C_INIT`(1M), **`conf`**(1CC), **`security`**(4CC)

NAME

umount – unmount file systems

SYNOPSIS

umount [–fv] *special_device mount_point*

umount –a [–fv] [–h *host*] [–t *type*]

DESCRIPTION

umount calls the **unmount**(2POSIX) system call to remove a remote node (*hostaddr:file_system*) or *special_device* from the file system tree at the *mount_point*.

OPTIONS

umount supports the following options:

–a — Unmounts all file systems listed in **fstab**(4CC)

–f — Forces unmount of the filesystem. Active special devices continue to work, but all other files return errors if further accesses are attempted.

Note - The root file system (/) cannot be forcibly unmounted.

–h *host* — Unmounts only file systems mounted from the specified *host*. Unmounts only NFS file systems unless specified otherwise with the –t option.

–t *type* — Indicates which is used to indicate the file system type to which the action applies.

For example:

```
umount –a –t nfs,ufs
```

unmounts all NFS and UFS file systems.

–v — Verbose mode. Additional information is displayed as each file system is unmounted.

FILES

/etc/fstab file system table

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO **unmount**(2POSIX), **fstab**(4CC), **mount**(1M)

NAME

ypbind – NIS binder process

SYNOPSIS

ypbind [–broadcast|–ypset|–ypsetme] &

DESCRIPTION

NIS provides a simple network lookup service consisting of databases and processes. The databases are stored on the machine running the NIS server process. The target can only run an *NIS client process*; the NIS server process must be run on a different machine on the network..

The programmatic interface to the NIS client process is described in gethostbyname(3STDC), gethostbyaddr(3STDC), getnetbyname(3POSIX) and getnetbyaddr(3POSIX). Currently, only these 3 routines are provided.

Administrative tools are described in domainname(1CC) and ypwhich(1CC). Tools for viewing the contents of NIS maps are described in ypcat(1CC) and ypmatch(1CC).

The ypbind actor is a *Chorus actor* that must be activated by the user in order to use the NIS services. Before launching the ypbind actor, the NIS domain must have been set using the *domainname(1CC)* command. Otherwise, ypbind will return with an error signaling that domain isn't set. By default, it is invoked as ypbind -broadcast.

The function of ypbind is to store the information that allows all NIS client processes on a node to communicate with the NIS server process. It must be run on every machine which has NIS client processes. The NIS server may or may not be running on the same node, but it must be running somewhere on the network (Note that the NIS server cannot be run on the target; the target can only run NIS client processes).

The information stored by ypbind is called a binding, it is the association of a domain name with an NIS server.

The process of binding is driven by client requests. As a request for an unbound domain comes in (if started with the -broadcast option) the ypbind process broadcasts on the net to find an NIS server. As the binding is established via broadcasting, there must be at least one NIS server on the net.

Once a domain is bound by ypbind, that same binding is given to every client process on the node. The ypbind process on a local or remote node may be queried to find the binding of a particular domain by using the ypwhich(1CC) command.

If ypbind is unable to communicate with the NIS server process to which it is bound, it marks the domain as unbound. It signals to the client process that the domain is unbound, and repeats the attempt to bind the domain. Requests received for an unbound domain will wait until the requested domain is bound. In general, a bound domain is marked as unbound when the node

running the NIS server crashes or gets overloaded. In this type of case, ypbind will attempt to bind to another NIS server using the process described above.

OPTIONS

`–broadcast` Send a broadcast datagram using UDP/IP requesting the information needed to bind to a specific NIS server. This option is analogous to ypbind with no options and is recommended for ease of use.

`–ypset` Not available.

`–ypsetme` Not available.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`domainname`(1CC), **`ypcat`**(1CC), **`ypmatch`**(1CC), **`ypwhich`**(1CC)

NOTES

The *portmap* actor must be activated before the *ypbind* actor can be run.

The *ypbind* actor supports multiple domains, it can maintain bindings to several domains and their servers.

The `–broadcast` option only works on the UDP transport. It is insecure, as it will treat any machine on the network that responds to the broadcast request as an NIS server.

Index