INTERACTIVE TCP/IP Guide

INTERACTIVE

product family

INTERACTIVE

A Kodak Company

First printing (September 1991)

INTERACTIVE TCP/IP Guide

CONTENTS

Introduction to INTERACTIVE TCP/IP

Welcome to the *INTERACTIVE TCP/IP Guide*. This guide contains the information you need to install, maintain, and use INTERACTIVE TCP/IP, Version 1.3. Whether you are an experienced INTERACTIVE TCP/IP user or brand new to INTERACTIVE TCP/IP, be sure to read the next few pages of this document. They will tell you what is contained in this guide and how to use the guide to your best advantage.

WHAT'S INCLUDED

The *INTERACTIVE TCP/IP Guide* includes:

- **INTERACTIVE TCP/IP Release Notes**
  Provides a description of the current release of INTERACTIVE TCP/IP.
- **INTERACTIVE TCP/IP Networking Primer**
  Provides basic instructions on how to use a number of user-level networking commands and applications.
- **INTERACTIVE TCP/IP System Administrator's Manual**
  Provides step-by-step instructions on how to install and use the INTERACTIVE TCP/IP software, how to configure a network, how to access the standard Berkeley utilities, and how to use the `sysadm` utility to access remote printers.
- **Setting Up a Gateway on an INTERACTIVE UNIX System With TCP/IP**
  Describes how to create a gateway on INTERACTIVE UNIX* System V/386 Release 3.2 using TCP/IP and two dumb Ethernet* cards.
- **INTERACTIVE TCP/IP Programmer's Supplement**
  Provides supplemental information on how to program the USL Transport Layer Interface and the Berkeley Software Distribution socket interface.

- **INTERACTIVE TCP/IP Reference Manual**
  INTERACTIVE's proprietary manual entries, which supplement the *INTERACTIVE UNIX System V/386 Release 3.2 User's/System Administrator's Reference Manual*.

- **Reader's Comment Form**
  Provides you with a way to tell us what you like or dislike about this guide and to send us your ideas for making it even better.

Supplemental documentation in this guide includes:

- **Name Server Operations Guide for BIND**
  This article, reprinted from the University of California, Berkeley document of the same name, provides information about name servers and about how to set up and manage a domain.

- **Domain Administrator's Operations Guide**
  This article, reprinted from the DDN Network Information Center document of the same name, provides guidelines for domain administrators on how to operate a domain server and maintain its portion of the hierarchical database. This document is intended for experienced users who are familiar with the domain system.

- **Line Printer Spooler Manual**
  This document describes the structure and installation procedure for the line printer spooling system, which supports multiple printers and spooling queues, local and remote printers, and printers attached via serial lines that require line initialization, such as the baud rate.

- **SNMP Overview and Installation Instructions**
  Provides a brief overview of INTERACTIVE's implementation of SNMP, and explains how to install and configure the software.

- **SNMP Reference Manual**
  Provides manual entries describing programs and commands that are available with SNMP.

WHERE TO BEGIN

The *INTERACTIVE TCP/IP Guide* includes a variety of documents for users at varying levels of experience. Depending on your level of experience, you may want to use this guide in a number of different ways. The outline below provides some suggested ways to use this guide:

- **If you are a beginner . . .**
  Work through the "INTERACTIVE TCP/IP Networking Primer" to learn how to use some basic networking commands and applications.
- **If you are an experienced TCP/IP user . . .**
  Read the "INTERACTIVE TCP/IP Programmer's Supplement" for information about how to program the USL Transport Layer Interface and the Berkeley Software Distribution socket interface. For more detailed information on name servers and domains, read the "Name Server Operations Guide for BIND" and the "Domain Administrator's Operations Guide."
- **If you are installing the software . . .**
  First read the "INTERACTIVE TCP/IP Release Notes." Then read and follow the steps outlined in the "INTERACTIVE TCP/IP System Administrator's Manual."
- **If you want supplemental documentation . . .**
  Refer to the *INTERACTIVE Network Drivers Overview and Installation Instructions* for information about how to configure a particular network driver. Refer to the *Network Programmer's Guide* for more information about the Transport Layer Interface. Refer to the 4.3BSD *UNIX Programmer's Reference Manual* for more information about the socket interface. Refer to the *INTERACTIVE Software Development System Guide and Programmer's Reference Manual* for more information about networking support functions. These documents are intended for experienced users. Refer to the "Documentation Roadmap" included in the *INTERACTIVE UNIX System V/386 Release 3.2 Operating System Guide* for a complete listing of all INTERACTIVE UNIX System V/386 Release 3.2 documentation.

OVERVIEW OF INTERACTIVE TCP/IP

The INTERACTIVE Transmission Control Protocol/Internet Protocol (TCP/IP) facilities provide the standard TCP/IP data transfer services, such as internetwork routing between network interfaces, security facilities, retransmission features, and three classes of internetwork addresses. TCP/IP supports two programming interfaces: the USL Transport Layer Interface (TLI) and the Berkeley Software Distribution (BSD) socket interface, which provide access to the data transfer services of the underlying network protocols. Additional applications, such as `telnet` and `ftp`, are also provided, along with a substantial number of networking commands. The software includes device driver components for each supported Ethernet board.

An Ethernet board may be either "dumb" or "intelligent." A dumb board has no processing capability; it is simply used as an input/output device to read and write data to and from the Ethernet cable. On a system using a dumb board, all of the communication data processing is performed by the main CPU. In order to use a dumb board, your operating system must provide independent (host-based) support for communication programs. An intelligent board includes a processing element and some on-board memory. On a system using an intelligent board, most of the networking programs are downloaded to the board at initialization; thereafter, the board handles most of the communication processing.

The INTERACTIVE TCP/IP software is host-based and is compatible with dumb Ethernet boards produced by various manufacturers, including Western Digital, 3COM*, and Ungermann-Bass. See the *INTERACTIVE Network Drivers Overview and Installation Instructions* for a list of boards currently supported.

INTERACTIVE TCP/IP FEATURES

INTERACTIVE TCP/IP provides a networking solution for TCP/IP protocol capabilities in STREAMS and Transport Layer environments. INTERACTIVE TCP/IP includes the Berkeley socket interface and extensive user-level applications. Features of INTERACTIVE TCP/IP are described below.

FTP (File Transfer Protocol)

The `ftp` program allows users to copy and send files to other hosts on the network, including non-UNIX System hosts, as long as they are connected to the network and use TCP/IP protocols.

Telnet Protocol

The `telnet` utility allows users to log in to remote systems, including non-UNIX System hosts, as long as they are connected to the network and use TCP/IP protocols.

BSD Applications

A number of BSD networking commands and applications are available to access and utilize resources on remote hosts using TCP/IP protocols. These include `lpc`, `lpq`, `lpr` `lprm`, `rcp`, `rlogin`, `rsh`, `ruptime`, and `rwho`.

Jacobson Enhancements

The Karels/Jacobson slow start algorithm was added to the transport. The Jacobson Enhancements are based on work done in the BSD Tahoe release.

Other Applications

An extensive number of user-level applications are also available with INTERACTIVE TCP/IP. These include applications such as `arp`, `finger`, `gated`, `ifconfig`, `named`, `sendmail`, `trpt`, `snmpd`, `getid`, `getmany`, `getnext`, `getone`, `getroute`, `setany`, `snmpd`, and `snmpstat`.

DOCUMENTATION REFERENCES

Throughout this guide, the following full documentation titles will be referenced in shortened versions as follows:

Full Title	*Shortened Version*
INTERACTIVE UNIX System V/386 Release 3.2 Operating System Guide	INTERACTIVE UNIX Operating System Guide
INTERACTIVE UNIX System V/386 Release 3.2 User's/System Administrator's Reference Manual	INTERACTIVE UNIX System User's/System Administrator's Reference Manual
INTERACTIVE Software Development System Guide and Programmer's Reference Manual	INTERACTIVE SDS Guide and Programmer's Reference Manual
UNIX System V/386 DOCUMENTER'S WORKBENCH Technical Discussion and Reference Manual	DOCUMENTER'S WORKBENCH Technical Discussion and Reference Manual

CONVENTIONS USED

References of the form *title* [n] are listed in the reference section at the end of the "INTERACTIVE TCP/IP Programmer's Supplement."

In the "INTERACTIVE TCP/IP Networking Primer," the prompt for each host is indicated by the host name followed by the `$` symbol. For example, the prompt for the `marketing` host is indicated in this way: `marketing$`.

FOR MORE INFORMATION

The documentation included in this guide provides information about how to install and use INTERACTIVE TCP/IP and supplements the information found in the *INTERACTIVE Network Drivers Overview and Installation Instructions*, *Network Programmer's Guide*, the *UNIX Programmer's Reference Manual*, and the *INTERACTIVE SDS Guide and Programmer's Reference Manual*. For a complete listing of all INTERACTIVE UNIX System related documentation, refer to the "Documentation Roadmap" included in the *INTERACTIVE UNIX Operating System Guide*.

INTERACTIVE TCP/IP
Version 1.3
Release Notes

CONTENTS

INTERACTIVE TCP/IP
Version 1.3
Release Notes
September 1991

1. INTRODUCTION

INTERACTIVE TCP/IP Version 1.3 is intended for installation on an AT* 386* machine or on a machine with Micro Channel* architecture (such as a PS/2*) running the INTERACTIVE UNIX* Operating System Version 2.2 or later or the 386/ix* Operating System Release 2.0.2 or later. INTERACTIVE TCP/IP Version 1.3 includes several new features and a number of bug fixes. There are several new features in this release of TCP/IP that depend on the presence of INTERACTIVE UNIX Operating System Version 3.0, and they are not supported on earlier versions of the operating system. These features are noted in section 2.

This document includes information about the TCP/IP software and outlines important installation procedures.

☛ Read these release notes completely before attempting to install the TCP/IP software.

2. NEW FEATURES

- With this release, Simple Network Management Protocol (SNMP) support is included. The SNMP subset consists of an agent (daemon) and a few simple utilities for querying the agent. The SNMP agent only has support for the INTERACTIVE TCP/IP and INTERACTIVE Network Drivers packages.
- The TCP/IP package no longer includes the device drivers. Instead, the separate INTERACTIVE Network Drivers package is provided. This makes the addition of new driver support simpler and also allows better use of the drivers with other protocol implementations.
- Token Ring is better supported by the software.
- The `libinet` library has additional Berkeley software (BSD) emulation support:

  - The `adjtime` system call is provided (supported under the INTERACTIVE UNIX System Version 3.0 only).
  - `gettimeofday` will return more accurate time information. Under the INTERACTIVE UNIX System Version 3.0, the information is now accurate to the microsecond. On earlier versions of the operating system, the time is only be an approximation in the microsecond value, but it should no longer result in a second call returning a value which appears earlier than the first call. Refer to section 6, "BUGS," for additional information.
  - `settimeofday` has been added to the library.
  - `rename` now works properly.
- Network Time Protocol (NTP) support:
  The `ntpd` facility developed by the University of Maryland has been added as a standard component of INTERACTIVE TCP/IP. To use it, however, you must have the INTERACTIVE UNIX System Version 3.0 installed.
- Berkeley Time Daemon support is now included. The 4.3BSD `timed` server has been added as a standard part of INTERACTIVE TCP/IP. Like NTP, its operation is dependent upon the presence of the INTERACTIVE UNIX System Version 3.0.
- `named` now does asynchronous zone transfers. This eliminates the delays caused when trying to do a zone transfer with a remote system that is down.
- The Berkeley Line Printer system is now included as an alternative to the System V printing system. It provides a different type of functionality, allowing the use of printers located on remote systems via the BSD standard `lpr/lpd` protocol. This alternate capability will be primarily of use on networks using BSD, ULTRIX*, and SunOS*-based systems, and is not necessarily intended to *replace* the System V `lp` utilities. Note that it is **not** possible to directly share a single printer between the two printing systems, although it *should* be possible for sites to write interface scripts/output filters to pass jobs from one system to the other.

3. BUGS FIXED

Many bugs have been fixed since the previous release of INTERACTIVE TCP/IP (Version 1.2).

Several bugs have been fixed in the TCP and socket modules. The socket module is part of the STREAMS subset. However, a new socket module is included with this release that will automatically be loaded on INTERACTIVE UNIX Operating Systems Releases 2.2 or earlier.

The bugs most visible to users that have been fixed include:

- Multiple, simultaneous incoming connections no longer "hang" a server.
- The incidence of connections not closing completely has been significantly reduced, if not eliminated entirely.
- The `ping` command is as reliable as the underlying network. Also, the problem in which the use of `ping` could hang a TCP connection has been eliminated.
- `gated` will reliably support the EGP and HELLO routing protocols. In previous releases, use of these routing protocols could interfere with TCP connections.
- The `X-DISPLAY-LOCATION` option negotiation in the TELNET server has been fixed. It now works correctly with other implementations and has been tested against the 4.4BSD and NCD implementations.
- `rshd` now sets up the environment variables correctly.
- The `rsh` command now both correctly detects end-of-file and implements the `-n` option.
- Token Ring users will find that `/etc/netd` now waits for the board to finish initializing before allowing the rest of the network software to start.
- Using `t_accept` to accept a connection onto the same file descriptor the connection indication was received on no longer causes a kernel panic.
- All known kernel panics related to the TCP/IP package have been fixed.

- `rcp` will no longer attempt to copy *special* files.
- The `accept` library call now supports the `O_NDELAY` mode of operation correctly.

4. SUPPORTED BOARDS

With this release, drivers have been moved to the INTERACTIVE Network Drivers package. Refer to the *INTERACTIVE Network Drivers Overview and Installation Instructions* for details.

5. INSTALLATION

The INTERACTIVE TCP/IP software should *not* be run on a system that is already running a non-INTERACTIVE TCP/IP networking software package. You can overlay INTERACTIVE TCP/IP Version 1.3 on a previous version of the INTERACTIVE TCP/IP extension.

The installation must be run at `init 2` level or at `single user` level.

5.1 Configuration Requirements

The following software subsets or extensions must be installed on the INTERACTIVE UNIX Operating System (386/ix Operating System Release 2.0 or later) before installing INTERACTIVE TCP/IP:

- Core: Version 2.0 or later (if SLIP is used, you must be running Version 2.2 or later)
- Kernel Configuration: Version 2.0 or later
- File Management: Version 2.0 or later

If you are running the INTERACTIVE UNIX System Version 2.2 or later, pseudo-tty drivers should be installed before installing the INTERACTIVE TCP/IP package.

For installation of a network card driver, follow the installation instructions in the *INTERACTIVE Network Drivers Overview and Installation Instructions*.

5.2 Summary: Installing and Activating INTERACTIVE TCP/IP

Here is a summary of the steps that would be required to activate TCP/IP for the first time:

1. Install the INTERACTIVE Network Drivers extension.
2. Install the INTERACTIVE TCP/IP extension.
3. Configure the driver you require using the `netdrvrmgmt` option of `sysadm`.
4. Configure TCP/IP, making sure `/etc/hosts` and `/etc/host.equiv` are valid, using the `tcpipmgmt` option of `sysadm`.
5. Configure an interface into TCP/IP by using the `sysadm tcpipmgmt` option `hbtcpmgmt`.
6. Set the run level. TCP/IP is set to run at "init level 3." The run level can be examined and changed if needed by using the `sysadm tcpipmgmt` option `hbtcpmgmt`.
7. Rebuild the kernel and shut down the system. If the LAN board has not yet been installed in the machine, then power down the machine and install it, and then power up the machine. Reboot the system. It will come back up with TCP/IP running.

6. BUGS

A problem with the `gettimeofday` emulation in the TCP/IP software has been noted. This bug will disappear once the INTERACTIVE UNIX System Version 3.0 kernel is available. `gettimeofday` uses the value of the `times( )` call to get clock ticks, pulls out the fraction of a second value, and scales this value to microseconds. Since the `time( )` call and `times( )` calls are separate, it is possible for the fraction of a second to wrap around independently of the second value, making it appear to be a negative time. The `gettimeofday` emulation will detect this condition and compensate for it, but the actual microsecond time will only be an approximation.

INTERACTIVE TCP/IP
Networking Primer

CONTENTS

INTERACTIVE TCP/IP
Networking Primer

INTRODUCTION

What Will I Learn From This Primer?

This primer will introduce you to some basic, user-level networking commands and applications. In this primer you will learn how to:

- Copy files and directories from one host on the network to another.
- Log in to a remote host.
- Execute a shell command on a remote host.
- List users on a remote host.
- Find out the status of a remote host.

Command Syntax

The UNIX* System is *case sensitive*, which means that the system always distinguishes between uppercase and lowercase letters. Most UNIX System commands, options, and arguments are typed in lowercase letters. Options typically begin with a dash (−). Each command, option, or argument consists of one *word*, which is interpreted as a group, or string, of characters surrounded by spaces.

If you make an error when typing a command, use the BACKSPACE key to correct the error. You may *not* use the cursor-positioning keys.

Unless otherwise specified, always type the command name first, followed by a space; the desired option or options, each followed by a space; then any arguments, separated by spaces.

This is the command format:

COMMAND NAME	`command name`
FORMAT	`command` [option(s)] *argument(s)*
DESCRIPTION	A brief description of what the command does.
OPTIONS	A list of the most useful options and a brief description of each.
ARGUMENTS	Mandatory or optional arguments.

If an argument is not required, it is shown in square brackets [ ]. Options are always "optional," so they are always shown in square brackets [ ]. Only the most common options and arguments are discussed in this primer. If there are additional options or arguments available for a particular command that are not presented in this primer, this is indicated by the phrase "Not presented in this document." For a complete listing of the available options and arguments for a command, refer to the reference manuals for your system.

USER NETWORKING COMMANDS

INTERACTIVE TCP/IP is delivered with user-level commands that allow you to copy, execute, or delete files on a remote machine. In some cases, you must supply a *host* (or *rhost*) name as an argument to the command. A table containing host names on your local network is kept in the file `/etc/hosts`. This is a system file that contains information, including the host name, for each host on the network. A sample `/etc/hosts` file might look similar to this:

```
127.1        local        localhost
193.16.14.1 marketing    mktg   m   # Marketing VAX running Ultrix
193.16.14.2 training     trng   t   # Training 386 INTERACTIVE UNIX
193.16.14.3 accounting   acct   a   # Accounting 3B2 running UNIX v.3
193.16.14.4 engineering  eng    e   # Engineering VAX running BSD 4.3
```

Each entry includes the internet address, the host name, and aliases for the host name. For example, the host name in the first entry is `marketing`, and the aliases are `mktg` and `m`. You may use either the full host name or one of its aliases wherever command syntax calls for a host name. Refer to the "INTERACTIVE TCP/IP System Administrator's Manual" in this guide for information about internet addresses.

To indicate a file or directory on a particular machine, use a colon (`:`) to separate the host name from the directory or file name. For example, to indicate the `accounts` directory on the host `training`, use the following format:

```
training:/accounts
```

To indicate the `jan.89` file in the `accounts` directory on the `training` machine, separate each part of the path name with a slash (`/`), as you would in a normal UNIX System path name:

```
training:/accounts/jan.89
```

Remote Login (`rlogin`)

The `rlogin` command is used to log in to a remote host so that you may work as though you were directly connected to that host.

COMMAND NAME	`rlogin`
FORMAT	`rlogin` *rhost* [–e*c*] [–l *username*]
DESCRIPTION	Log in to the named remote host.
OPTIONS	`-e` *c* Change default escape character, ~, to the named character. Do not leave a space between the option and the new escape character. `-l` *username* Log in to a remote host using a different remote user name. A `.rhosts` file, containing your name and the local host name, must exist on the remote host in the login directory of the user whose name you specify. Other options not presented in this document. Refer to *rlogin*(1).
ARGUMENTS	The name of a remote host.

A file called `/etc/hosts.equiv` is stored on every host machine. This file is maintained by your system administrator and contains a list of remote host names. If the `/etc/hosts.equiv` file on the remote host contains the name of your local host and you have a login account on that machine, you can use `rlogin` to log in automatically, i.e., without entering a password. (If you have a login account on a remote host but the name of your local host is not specified in the `/etc/hosts.equiv` file on the remote host, you can still log in to the remote host by providing your password.)

For security reasons, your system administrator may not have added your local host's name to the `/etc/hosts.equiv` file on the remote host. You can still log in automatically to the remote host by creating your own private equivalent file on that host. This file is called `.rhosts` and should be kept in your login directory.

Like the `/etc/hosts.equiv` file, the `.rhosts` file should contain the host names of remote machines. This allows you to log in automatically from any host listed in your `.rhosts` file. You

can also use the `.rhosts` file to allow users on other hosts to log in automatically to your account. For example, user `janet`'s `.rhosts` file on the `marketing` host might look like this:

```
sales
pubs
sales          susan
training       mary
training       john
```

The first two entries indicate that user `janet` can automatically log in to her account on the `marketing` host from both the `sales` and the `pubs` hosts. The next three entries indicate that user `susan` on the host `sales` and users `mary` and `john` on the host `training` can log in to `janet`'s account on `marketing`.

To log in automatically to another user's account, specify the `-l` option followed by the login name of the user whose account you wish to access. For example, if user `janet`'s `.rhosts` file on the `marketing` host contains an entry for user `john` on the `training` host, `john` can log in to `janet`'s account on `marketing` in this way:

```
training$  rlogin marketing -ljanet
marketing$
```

Use the CTRL d sequence to terminate an `rlogin` session.

The `rcp` and `rsh` commands also make use of the `/etc/hosts.equiv` and `.rhosts` files. Refer to "Remote Copy (`rcp`)" and "Execute a Shell Command on a Remote Host (`rsh`)" in this primer for more information on these commands.

Remote Copy (`rcp`)

The `rcp` command allows you to copy files and directories from one host on the network to another.

COMMAND NAME	`rcp`
FORMAT	`rcp` [`-p`] [*host1*:] *file1* [*host2*:] *file2* [`-r`] *file directory*
DESCRIPTION	Copy the named files and directories from one host to another.
OPTIONS	`-p` Preserve the modification times and permissions of the original files and directories. `-r` Copy any subtree (subdirectories and files within a directory) in the named directory. The destination must be a directory.
ARGUMENTS	A source file name and a destination file or directory name.

If you provide only one host name, the local host is used as the default for the unspecified host. For example, user `janet` might copy `file.new` in her current directory on host `marketing` to the `accounts` directory on the host `training` in this way:

```
marketing$ rcp file.new training:/accounts
```

Since a source host is not specified, the local host is assumed.

The `rcp` command may also be used to copy files from one remote host to another. User `janet` on `marketing` could copy the file `/meetings/summary` on the `sales` host to the `accounts` directory on the `training` host by specifying the source host and the destination host:

```
marketing$ rcp sales:/meetings/summary training:/accounts
```

In order to use the `rcp` command, the `/etc/hosts` file on the remote host must contain an entry for the local host. The `rcp` command also makes use of the `/etc/hosts.equiv` and `.rhosts` files. Refer to "Remote Login (`rlogin`)" in this primer for more information about these files.

Execute a Shell Command on a Remote Host (`rsh`)

The `rsh` command allows you to connect to the named remote host and execute a shell command.

COMMAND NAME	`rsh`
FORMAT	`rsh` *rhost* [*command*] [*arguments*]
DESCRIPTION	Log into a remote host and execute a shell command.
OPTIONS	`-l` Log into a remote host using a different user name. A `.rhosts` file containing your name and the local host name must exist on the remote host in the login directory of the user whose name you specify.
OPTIONS	Other options not presented in this document. Refer to *rsh*(1).
ARGUMENTS	A host name and a command name.

`rsh` logs you in to a remote host and executes the named command. If you specify a command, `rsh` terminates and returns you to your local machine as soon as the command has finished processing. For example, user `jack` on host `marketing` might use the `rsh` command to get a listing of the files in his login directory on the `sales` host:

```
marketing$  rsh sales ls
forecast.89
prospects
plans.new
projects
sample.lit
marketing$
```

If you do *not* specify a command name, the system logs you into the remote host and waits for you to issue a command. You will remain logged in to the remote host until you log out from it. To log out of a remote host, use the CTRL d sequence:

```
marketing$  rsh sales
sales$  ls
forecast.89
prospects
plans.new
projects
sample.lit
sales$  CTRL d
marketing$
```

Screen-oriented applications, such as `vi`, will not run properly with the `rsh` command. To run `vi` (or a similar, screen-oriented application), `rlogin` to the remote host, then run the application.

The `rsh` command also makes use of the `/etc/hosts.equiv` and `.rhosts` files. Refer to "Remote Login (`rlogin`)" in this primer for more information about these files. If you are unable to run the `rsh` command on a remote host, make sure that the `/etc/hosts.equiv` file on the remote host contains an entry for the local host. See your system administrator if the problem persists. For more information on `rsh`, refer to *rsh*(1).

Report Status of Remote Hosts (`ruptime`)

The `ruptime` command provides a status line for each host on the local network.

COMMAND NAME	`ruptime`
FORMAT	`ruptime [ –alrtu ]`
DESCRIPTION	Report the status of all hosts on the local network. The output can include host uptime, load averages, and the number of interactive users.
OPTIONS	`-a` Include all users, even those idle an hour or more (users idle an hour or more are not counted unless this option is specified). `-l` List hosts in order of load averages. `-r` List hosts in reverse order. `-t` List hosts by system uptime. `-u` List hosts by number of users.
ARGUMENTS	None.

The `ruptime` command typically lists hosts in alphabetical order by host name. For example, an `ruptime` listing might look similar to this:

```
eng          up       1+02:41    2 users    load  0.15, 0.22, 0.19
marketing    up          9:17    5 users    load  1.50, 2.11, 1.35
pubs         down        3:15
sales        up      12+03:32   15 users    load  2.10, 2.00, 1.75
tech         up       5+23:51    3 users    load  0.10, 0.21, 0.50
```

You may use the options described above to sort the status listing by load average, uptime, or number of users. For example, the command `ruptime -t` will sort the listing according to the amount of time each machine has been "up":

```
sales        up      12+03:32   15 users    load  2.10, 2.00, 1.75
tech         up       5+23:51    3 users    load  0.10, 0.21, 0.50
eng          up       1+02:41    2 users    load  0.15, 0.22, 0.19
marketing    up          9:17    5 users    load  1.50, 2.11, 1.35
pubs         down        3:15
```

`uptime` refers to the amount of time the host has been transmitting data gathered by the `rwhod` process, which collects information about the number of users and the average load on each machine in the network. This information is sent to each host approximately every minute and stored in a system file. When the `ruptime` command is run, the information from this file is displayed on your screen. If no information has been received from the `rwhod` process on a host for 11 minutes, `ruptime` lists that host as `down` and does not provide further information on that host. `uptime` is listed in days, hours, and minutes.

The `load average` indicates the number of jobs waiting during the last 1, 5, and 15 minutes. To sort the status listing according to `load average`, use the `-l` option.

The `number of users` indicates all users logged into the host *except* for those users whose terminals have been idle for an hour or more. To sort the status listing according to the number of users who are actively using their terminals (i.e., have typed on the terminal within the last hour), use the `-u` option. Users whose terminals have been idle for an hour or more are not listed. To include logged in users whose terminals have been idle for an hour or more, use the `-a` option. For example, to view a status listing sorted according to the number of users on each host, including users who have not typed on their terminals for an hour or more, use this command:

```
ruptime -u -a
```

List Users on Remote Hosts (`rwho`)

The `rwho` command lists users logged in to hosts on the local network.

COMMAND NAME	`rwho`
FORMAT	`rwho [ -a ]`
DESCRIPTION	List users on all hosts on the local network. Output includes the host name of the system the user is logged into, the name associated with the user's terminal, the user's login time, and the idle time if greater than 1 minute. Time is specified in the form *hours:minutes:seconds*.
OPTIONS	`-a` Include users who have not typed on their terminals for an hour or more. These users are not listed unless this option is specified.
ARGUMENTS	None.

At regular intervals, each machine on a local network broadcasts information about users who are logged in. This information is stored in system files. The `rwho` command displays information from these system files on the screen. A typical `rwho` listing might look similar to this:

```
alan     marketing:tty10        Jul 25  11:08:01
bob      sales:tty22            Jul 25  09:23:22
diane    training:tty29         Jul 25  08:15
jane     sales:tty25            Jul 25  08:22:34
john     training:tty31         Jul 25  10:02
ken      engineering:tty12      Jul 25  15:31:33
mary     training:tty33         Jul 25  13:20:02
susan    sales:tty16            Jul 25  08:45:12
tom      engineering:tty14      Jul 25  10:30:14
```

If no broadcast information has been received from a remote host for 11 minutes, `rwho` assumes that the host is down and does not include users on that host in its output.

File Transfer Program (`ftp`)

The `ftp` command allows you to transfer files between your local host and a remote host, including a non-UNIX System host, as long as that host is connected to the network and uses TCP/IP protocols.

COMMAND NAME	`ftp`
FORMAT	`ftp` [–vni] [*rhost*]
DESCRIPTION	Transfer files between a local host and a remote site.
OPTIONS	`-v` Display all responses from the remote server and report on data transfer statistics. `-n` Turn off auto-login feature. `-i` Turn off interactive prompting during multiple file transfers. Other options not presented in this document. Refer to *ftp*(1).
ARGUMENTS	A host name.

You may invoke the `ftp` command with or without a remote host name. If you do not specify a remote host when you invoke `ftp`, you must use the `open` subcommand in order to connect to a remote host. To determine which remote hosts are available to you and what the correct host names are, refer to the `/etc/hosts` file on your system or see your system administrator.

When you type the `ftp` command, `ftp` enters its command mode and displays the `ftp>` prompt. In this mode, you can issue any of the `ftp` subcommands (see the partial list of subcommands below). These commands behave like standard UNIX System commands and use standard UNIX System syntax. At the `ftp>` prompt, type `?` to view a complete listing of the `ftp` subcommands. To view a one-line description of a particular subcommand, type `help` followed by the subcommand name.

Once you make a connection to a remote host and log in, you can use any of the `ftp` subcommands to exchange files or obtain directory or status information. You can end an `ftp` session with the `quit` or `bye` subcommands, either of which will end the session and exit `ftp`.

Some commonly used `ftp` subcommands include:

`cd` *remote-directory*
Change the working directory on the remote host to *remote-directory*.

`delete` *remote-file*
Delete the *remote-file* on the remote host.

`get`
Retrieve the *remote-file* and store it on the local machine.

`ls` [*remote-directory*] [*local-file*]
Print an abbreviated directory listing of *remote-directory*. If *local-file* is specified, place the listing in that file. If *local-file* is not specified, display the listing on the terminal.

`mkdir` *directory-name*
Make a directory on the remote host.

`put` *local-file* [*remote-file*]
Store a local file on the remote machine.

`quit`
Terminate the `ftp` session and exit `ftp`.

`remotehelp` [*command-name*]
Request help from the remote `ftp` server.

A simple `ftp` session to list the files on the remote host `training` from the local host `marketing` might look similar to this:

```
marketing$  ftp training
ftp>
ftp>  ls
course.new
hands.on
literature
products
sales

ftp>  delete hands.on

ftp>  ls
course.new
literature
products
sales

ftp>  quit
221 Goodbye.
marketing$
```

Refer to *ftp*(1) for a complete list of `ftp` subcommands.

Remote Login Using Telnet Protocol (`telnet`)

The `telnet` utility is an interactive program that allows you to log in to a remote host as though you were directly connected to that host. Unlike `rlogin`, `telnet` allows you to connect to any remote host, including a non-UNIX System host, as long as that host is connected to the network and uses TCP/IP protocols. This can be very useful, for example, if you want to log in to a host that is not running the Berkeley UNIX System, from a host that is running the UNIX System.

COMMAND NAME	`telnet`
FORMAT	`telnet [` *rhost* `] [` *port* `]`
DESCRIPTION	Perform remote login using the Telnet protocol.
OPTIONS	None.
ARGUMENTS	A remote host name and a port number.

To use `telnet`, you must know the name of the remote host you wish to access. To determine which remote hosts are available to you and what the correct host names are, refer to the `/etc/hosts` file on your system or see your system administrator. In addition to specifying a host name, you may specify a *port* number for the remote host. A port is a particular physical connection to a host and is identified by a port number.

If you specify a remote host name without specifying a port number, `telnet` uses a default port number. Once you are connected to the remote host, you enter `telnet`'s input mode. In this mode you may type text and commands as you normally would if you were logged directly in to the remote system. In input mode, you may also execute the `telnet` subcommands (see the partial listing of `telnet` subcommands below). To execute a subcommand from input mode, you must first type CTRL]:

```
marketing$  telnet sales
Login: janet
Password:

sales$  ls
forecast.89
prospects
plans.new
projects
sample.lit

sales$  CTRL ]

telnet>  status
Connected to marketing
Escape character is CTRL ].

sales$
```

Alternatively, you may use the `telnet` command without specifying an rhost or port number. If you run `telnet` without an argument, you will enter `telnet`'s command mode, indicated by the `telnet>` prompt. In this mode, `telnet` accepts and executes any of the standard `telnet` subcommands (see the partial listing of `telnet` subcommands below).

Some commonly used `telnet` subcommands are listed below:

`?` [*command*]
`help` [*command*]
: List a summary of help information for `telnet` subcommands. If a command is specified, help information for just that command will be listed.

`escape` [*new character*]
: Set the escape character to the named *new character*. Prompts for escape sequence if not specified.

`open` *host* [*port*]
: Open a connection to the named host.

`quit`
: Close an open `telnet` session and exit `telnet`.

`status`
: Show connect/disconnect status.

To use a subcommand from command mode, simply type the subcommand at the `telnet>` prompt. For example, to find out whether you are connected to the remote host, use the `status` subcommand:

```
marketing$  telnet
telnet> status
No connection.
Escape character is '^]'.
telnet>
```

Note that in this example, no connection has been made because the `telnet` command was used without a remote host name. A connection is only made when a remote host name is specified as an argument to `telnet` (`telnet` *rhost*) or as an argument to the `open` subcommand (`telnet> open` *rhost*).

The following example illustrates how to use the `open` subcommand to connect to a remote host from the `telnet` command mode:

```
marketing$  telnet
telnet> open sales
Login: janet
Password:
sales$
```

To quit `telnet` from input mode, use the CTRL d sequence. To quit `telnet` from command mode, use the `quit` subcommand. For more information on `telnet` and a complete listing of `telnet` subcommands, refer to *telnet*(1).

INDEX

INTERACTIVE TCP/IP
System Administrator's Manual

CONTENTS

INTERACTIVE TCP/IP
System Administrator's Manual

1. INTRODUCTION

The "INTERACTIVE TCP/IP System Administrator's Manual" describes the basic requirements that are necessary to install and use the TCP/IP software. For a description of how a network interface board should be configured for use with the INTERACTIVE TCP/IP software, refer to the *INTERACTIVE Network Drivers Overview and Installation Instructions.*

1.1 Overview of This Document

This document is divided into five major sections:

1. INTRODUCTION
   This section provides a general overview of this document.
2. INSTALLING THE TCP/IP SOFTWARE
   This section provides detailed information about how to load the TCP/IP software and configure the kernel.
3. CONFIGURING A NETWORK
   This section provides information about host naming conventions and about how to use the `sysadm` utility to modify configuration files. It also discusses more advanced configuration topics such as how to set a subnet mask and how to set a broadcast address.
4. ACTIVATING TCP/IP AND USING NETWORK APPLICATIONS
   This section provides information about how to activate the TCP/IP software and how to access the standard Berkeley utilities.
5. CONFIGURING LPR
   This section provides information about how to use the `sysadm` utility to access and manage remote printers over a network.

1.2 What Will I Learn From This Document?

This document provides the instructions required to install and use the INTERACTIVE TCP/IP software. This document describes:

- How to install the TCP/IP software.
- How to configure a network.
- How to activate TCP/IP.
- How to use `lpr` to access remote printers through the `sysadm` utility.

A set of networking commands is included as part of the TCP/IP software. Refer to the "INTERACTIVE TCP/IP Networking Primer" included in this guide for user-level information about these commands. For more technical information, refer to the appropriate manual entry in the "INTERACTIVE TCP/IP Reference Manual," also included in this guide.

2. INSTALLING THE TCP/IP SOFTWARE

2.1 Loading the Software

The following procedures are used to install the INTERACTIVE TCP/IP software.

Note that if you are installing this extension over a previous version, your screen displays will look slightly different than the ones shown here; for example, some files shown as being created will already exist, so they will be listed as existing files. If you currently have a version of the TCP/IP extension installed, a WARNING message will be displayed and you will be asked if you want to overwrite your current system. Respond `y` to install INTERACTIVE TCP/IP Version 1.3.

To begin the installation:

1. Use the `sysadm` command or log in as `sysadm` and access the Main menu. Your screen will look similar to this:

```
                        SYSTEM ADMINISTRATION

   1 diskmgmt        disk management menu
   2 filemgmt        file management menu
   3 machinemgmt     machine management menu
   4 packagemgmt     package management menu
   5 softwaremgmt    software management menu
   6 syssetup        system setup menu
   7 ttymgmt         tty management menu
   8 usermgmt        user management menu

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  q to QUIT:
```

2. Select option `5` to access the Software Management menu. Your screen will look similar to this:

```
                        SOFTWARE MANAGEMENT

   1 installpkg      install new software package onto built-in disk
   2 listpkg         list packages already installed
   3 removepkg       remove previously installed package from built-in disk
   4 runpkg          run software package without installing it

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT:
```

3. Select option 1, `installpkg`. On systems with two diskette drives, the system then prompts you to enter the drive number of the drive from which you want to install. Your screen will look similar to this:

```
        This system has two diskette drives.

Enter the drive number you wish to install
from ((default) 0, 1):
```

4. Type `0` if you plan to install from the `A:` drive, `1` if you plan to install from the `B:` drive. The system then prompts you for the density of the diskettes you are using for the installation. Your screen will look similar to this:

```
Enter density of the diskettes you are installing from:

1) 1.2MB    (5 1/4" high density)
2) 1.44MB   (3 1/2" high density)
3) 360KB    (5 1/4" low density)
4) 720KB    (3 1/2" low density)

Please enter #(1-4), default 1:
```

5. Type in the number corresponding to the type of diskettes you are using for the installation. The system prompts you to insert the first diskette into the diskette drive. The screen will look similar to this:

```
Confirm

Please insert the floppy disk.

If the program installation requires more than one floppy
disk, be sure to insert the disks in the proper order,
starting with disk number 1.
After the first floppy disk, instructions will be provided
for inserting the remaining floppy disks.

Strike ENTER when ready
or ESC to stop.
```

6. If you want to stop the installation process and return to the system prompt, press ESC. To proceed with the installation, insert the first diskette into the drive and press ENTER. The following message is displayed:

```
Installation in progress -- do not remove the floppy disk.
Install TCP Ethernet Support-Version 1.3? (y):
```

7. Type `y` to proceed with the installation of the INTERACTIVE TCP/IP package. (Type `n` to stop the installation.) If you type `y`, your screen will look similar to this:

```
Installing the TCP Ethernet Support-Version 1.3.
Copyright (c) 1986, 1988, 1991 INTERACTIVE Systems
Corporation.
All Rights Reserved
The following files are being installed:
```

8. A listing of the files being copied onto your system from the first diskette will accompany the above message. When all files from the first diskette have been copied, the installation will prompt you to insert the second diskette:

```
Floppy diskette number 1 is complete
Remove floppy and insert floppy number 2
Type <enter> when ready:
```

9. Insert the second diskette in the diskette drive and press **ENTER**. A list of files being copied onto your system from the second diskette will be displayed on your screen.

10. When the second diskette is copied, the following message displays:

```
Installing drivers
Creating Necessary System Files.
If these files currently exist, they will NOT be
over-written.
The file /etc/hosts has been created.
You should edit this file to your site specifications

<Press any key to continue>

Creating Necessary System Files.
If these files currently exist, they will NOT be
over-written:

 /etc/services
 /etc/networks
 /etc/protocols
 /etc/inetd.conf
 /etc/resolv.conf
 /etc/syslog.conf
 /etc/gated.conf

The file /etc/hosts has been created, and may need to be
edited for your site.  See the TCP/IP System
Administrator's Manual for details.

The file /etc/hosts.equiv has been created, and may need to
be edited for your site.  See the TCP/IP System
Administrator's Manual for details.

         *** TO COMPLETE THIS INSTALLATION ***
 You must install new drivers from the Network Drivers
 subset and configure these drivers using
 `sysadm netdrvrmgmt` (unless you have already done this.)
 Then you must add an interface to the new drivers using
 the 'addinterface' submenu of the 'sysadm hbtcpmgmt menu',
 and build and install a new kernel using 'kconfig'.

Press RETURN to continue:
```

☛ There are a number of potential problems that can cause the installation to abort. The most common problems are insufficient space on the `root` file system or a `/usr` file system that is unmounted. Other common problems include having the device `/dev/dsk/f0q15dt` already mounted on a local directory or a system failure to recognize the medium type. When one of these situations is detected by the installation procedure, an appropriate error message and a solution to the problem are displayed on the screen.

2.2 Configuring the TCP/IP Interface

You must install and configure the Network Drivers extension with the drivers you want to use, and you must install INTERACTIVE TCP/IP before you can configure the interface. Once the software is loaded, a new kernel must be generated to include the INTERACTIVE TCP/IP STREAMS modules and the Ethernet* board(s) of your choice.

1. Log in as `sysadm` and select the `packagemgmt` menu option from the System Administration menu. (If you are not at the System Administration menu display, use the `sysadm` command or log in as `sysadm` to access this menu, then select `packagemgmt`.) Your screen will look similar to this:

```
                    PACKAGE MANAGEMENT

 1 lpmgmt          print spooler management menu
 2 lprmgmt         Berkeley-style print spooler management menu
 3 netdrvrmgmt     Network Driver management menu
 4 nfsmgmt         NFS management menu
 5 tcpipmgmt       extended networking utilities menu
 6 uucpmgmt        basic networking utilities menu

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT:
```

2. Select the `tcpipmgmt` option from the `packagemgmt` menu. Your screen will look similar to this:

```
                EXTENDED NETWORKING UTILITIES MANAGEMENT

   1 equivmgmt       Manage tcp/ip hosts.equiv entries (list, add, delete)
   2 hbtcpmgmt       Manage INTERACTIVE TCP/IP setup (add, list, remove)
   3 hostmgmt        Manage tcp/ip hosts entries (list, add, delete)

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT:
```

3. Select the `hbtcpmgmt` option. The screen displays options for managing your network interface. Your screen will look similar to this:

```
                    Manage Network Interfaces

   1 addinterface    Add a network interface
   2 lsinterfaces    List currently configured network interfaces
   3 rminterfaces    Remove a configured network interface
   4 setrunlevel     Change default system run level

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT:

This procedure is used to add network interfaces to the TCP/IP package.
```

You can use these options to add, list, or remove a network interface from your system. Note that both the Network Drivers extension and INTERACTIVE TCP/IP must be installed on your system before INTERACTIVE TCP/IP can be used. Refer to the *INTERACTIVE Network Drivers Overview and Installation Instructions* for more information about how to configure a specific driver into the kernel.

2.2.1 Adding a Network Interface

In order to use a driver that is installed on your system, you must add it as a new interface. The `addinterface` menu provides you with five options. You may add a driver from those installed with the INTERACTIVE Network Drivers extension, add a third-party driver, add a slip interface, configure TCP/IP for local operation only, or view a list of the network interfaces currently installed and configured on your system.

1. Select option 1 to see the `addinterface` menu. The system will display a list of options for adding a network interface. Your screen will look similar to this:

```
                    ADD INTERFACE COMMAND MENU
   1 add a driver from the network driver subset as a new
     interface
   2 add a third party Ethernet driver as a new interface
   3 add a slip interface
   4 configure TCP/IP for local operation only (no interface
     card)
   5 list the installed and configured drivers
   "q" to quit
  Enter the operation you want to perform:
```

2.2.1.1 Adding an Installed Driver.

1. To add a driver that is already installed on your system, select option 1 from the `addinterface` menu. The system will display a list of each driver that is configured into the kernel and can be used as a network interface. In this example, `ec0`

is the only driver installed. Your screen will look similar to this:

```
Choose one of these configured drivers
        ec0
Please enter the name
```

2. Type the name of the driver you want to add. The system next prompts you for the host name. In this example, `myhost` is used. Your screen will look similar to this:

```
Please enter the hostname you wish to be associated
with this interface.  This hostname must be in the
/etc/hosts file. Names can be added to this file with
the hostmgmt sysadm menus.
Enter the hostname for this interface :  myhost
```

3. The system prompts you for the netmask. Netmasks are most often needed if you are using more then one network interface or if your LAN has more then one segment (connected by a gateway). If you are configuring only one interface and you have a small local area network, you will probably not need to change the netmask. Type `n` to use the default netmask. Your screen will look similar to this:

```
You will be configured to use the default netmask.
Change the netmask? [y, n, ?] n
```

If you want to change the netmask, see section 3.4.1, "Setting the Subnet Mask," or consult your network administrator.

4. The system prompts you for the broadcast address. Usually, you will not need to change the default broadcast address. Type `n` to use the default broadcast address.

```
You will be configured to use the default broadcast address.
Do you want to change the broadcast address? [y, n, ?] n
```

The `addinterface` screen will again be displayed.

2.2.1.2 Adding a Third-Party Driver. If you have a third-party driver that supports the INTERACTIVE TCP/IP interface, you can attempt to add it with this option. Note that this does not apply to network drivers installed on previous releases of INTERACTIVE TCP/IP. These previously installed drivers must be reinstalled from the Network Drivers extension before they can be used. Refer to the *INTERACTIVE Network Drivers Overview and Installation Instructions* for information about reinstallation of these drivers.

1. To add a third-party driver, select option 2 from the `addinterface` menu.

2. The system will prompt you for the name of the driver you want to add. Type the name of the interface. In this example, `ne0` is used. Your screen will look similar to this:

```
Enter the name of the driver you wish to use: ne0
```

3. If the driver you have selected is not installed, the system will remind you of this and ask you whether you want to (a) continue with the interface configuration and install the driver later or (b) choose another driver. Note that a driver must be installed through the Network Drivers extension, but it can be installed either before or after it has been associated with a host name.

4. If the driver you have selected is installed, you will be prompted for the host name you want associated with it. In this example, `myhost-ne0` is used. Your screen will look similar to this:

```
Please enter the hostname you wish to be associated
with this interface.  This hostname must be in the
/etc/hosts file.  Names can be added to this file
with the hostmgmt sysadm menus.
Enter the hostname for this interface : myhost-ne0
```

5. The system prompts you for the netmask. Your screen will then look similar to this:

```
You will be configured to use the default netmask.
Change the netmask? [y, n, ?]
```

6. Type `n` or press ENTER to accept the default netmask.

7. The system prompts you for the broadcast address. Your screen will look similar to this:

```
You will be configured to use the default broadcast address.
Do you want to change the broadcast address? [y, n, ?]
```

   Type `n` to use the default broadcast address. The system will return you to the `addinterface` screen.

8. To list the currently configured network interfaces, first press ENTER to access the `hbtcpmgmt` menu, then select option 2. Your screen will look similar to this:

```
Currently installed network interfaces:

Driver name      Hostname              IP Address
    ec0          myhost              128.212.16.59
    ne0          myhost-ne0          128.212.16.60

Press the RETURN key to see the hbtcpmgmt menu [?, ^, q]:
```

The screen displays the interfaces you have installed.

2.2.1.3 Adding a SLIP Interface. A Serial Line Internet Protocol (SLIP) interface is a point-to-point conection, for example, between a tty and a modem. A SLIP interface permits users to use any tty driver as a network interface. Performance is dependent upon the speed of the serial line used to connect hosts. A limited dial-up utility that allows users to establish network links with modems over normal phone lines is also supported. You can run the SLIP interface alone or in addition to an Ethernet interface on your system. If you are using SLIP and an Ethernet interface, then you should first install the Ethernet interface driver through the INTERACTIVE Network Drivers extension.

SLIP performance is dependent on the speed of the serial line used to connect hosts. Although it is functional at speeds of 2400 baud, some users may find SLIP performance unacceptable at speeds of less then 9600 baud. To use SLIP, it is required that INTERACTIVE UNIX System Version 2.2 or later be installed on your machine.

1. To add a SLIP interface, select option `3` from the `addinterface` screen. Your screen will look similar to this:

   ```
   SLIP will be configured to use the "sl0" device
   ```

2. The system prompts you for a host name. Type in the host name you want to use. Your screen will look similar to this:

   ```
   Please enter the hostname you wish to be associated
   with this interface.  This hostname must be in the
   /etc/hosts file.  Names can be added to this file
   with the hostmgmt sysadm menus.
   Enter the hostname for this interface : myhost-sl
   ```

   In this example, the host name `myhost-sl` is used.

3. The system prompts you for a target host name. Type in the target host name you want to use. Your screen will look similar to this:

```
Please enter the target hostname you wish to be associated
with this interface. SLIP is a point to point link, so
Interactive TCP/IP needs to know the hostname of the system
that will be on the other side of the SLIP link.
This host name must also be in the /etc/hosts database.
Enter the target hostname for the slip interface : target-sl
```

In this example the target host name `target-sl` has been used.

4. The system will now prompt you to use the default netmask and broadcast address. Type **n** at each prompt to use the system defaults.

5. The system now displays a screen similar to this:

```
                    SLIP Hardware Configuration

The sl0 driver is currently configured to use
the following TTY Line settings:

        TTY Line..................... tty00
        Baud Rate.................... 9600

Do you want to use the current TTY Line [ Yes ]? [y, n]
```

Type **y** or press **ENTER** to use the current line. If you do not wish to use the current line, type **n**. The system will then prompt you for the line you want to use. Your screen will look similar to this:

```
Do you want to use the current baud rate [ Yes ]? [y, n]
```

6. Type **y** or press **ENTER** to use the current baud rate. Type **n** if you do not want to use the current baud rate. The system will then prompt you to supply a baud rate. Your screen will then look similar to this:

```
Is this SLIP connection going to be directly connected
to another system running SLIP (without a modem) [ No ] ?
[y, n]
```

7. If your SLIP connection will be with a modem, press **ENTER** or type **n**. Type **y** if it will be a direct connection.

8. If your connection is with a modem, you will need to add an account for dialup SLIP access. Your screen will look similar to this:

```
Would you like to add an account for dialup Slip access
[ No ] [y, n]
```

Type `y` to add an account.

9. The system next prompts you for information about the new account. You must provide a login ID. You may either provide user and group ID numbers or select the system defaults by pressing ENTER at the appropriate prompts. Your screen will look similar to this:

```
Enter SLIP login ID [?, q]: target1

Enter user ID number (default 836) [?, q]:

Enter group ID number or group name
(default 1) [?, q]:

This is the information for the new login:
        User's name:    Dialup SLIP account
        login ID:       target1
        user ID:        836
        group ID:       1       (other)
Do you want to install, edit, or skip this entry
[i, e, s, q]?

        SLIP configuration completed
```

10. If you want to list the currently configured network interfaces, return to the `hbtcpmgmt` menu and select option 2. Your screen will look similar to this:

```
Currently installed network interfaces:

Driver name     Hostname          IP Address
    pc0         myhost            128.212.16.59
    ne0         myhost-ne0        128.212.16.60
    sl          myhost-sl         128.212.16.61

Press the RETURN key to see the hbtcpmgmt menu [?, ^, q]:
```

2.2.1.4 Configuring TCP/IP for Local Operation. To use TCP/IP without a network, select option `4` from the `addinterface` menu and follow the menu instructions that appear on your screen.

2.2.2 Removing a Network Interface

You can remove a network interface through the `hbtcpmgmt` menu.

1. To remove a network interface, return to the `hbtcpmgmt` menu and select option 3.

2. The system will prompt you for the name of the interface to be removed. In this example, the `sl` interface will be removed. Your screen will look similar to this:

```
Currently installed network interfaces:

Driver name      Hostname          IP Address
    pc0          myhost            128.212.16.59
    ne0          myhost-ne0        128.212.16.60
    sl           myhost-sl         128.212.16.61

Enter the name of the interface you wish to remove:
```

After entering the name of the interface you want to remove, you will see the message:

```
The /etc/netd.cf file has been updated.

Press the RETURN key to see the hbtcpmgmt menu [?, ^, q]:
```

2.2.3 Listing Network Interfaces

To view the list of currently installed interfaces, use option 2 of the `hbtcpmgmt` menu. Your screen will look similar to this:

```
Currently installed network interfaces:

Driver name      Hostname          IP Address
    pc0          myhost            128.212.16.59
    ne0          myhost-ne0        128.212.16.60

Press the RETURN key to see the hbtcpmgmt menu [?, ^, q]:
```

3. CONFIGURING A NETWORK

A host machine is configured into the network through special configuration files. To link a *host* to the rest of the network, the host name and the internet address of the host must be added to the configuration files `/etc/hosts` and `/etc/hosts.equiv`. (For more information about the internet address, refer to section 3.1.2.) The following default configuration files are provided with the software:

- `/etc/hosts`
- `/etc/hosts.equiv`
- `/etc/networks`
- `/etc/protocols`
- `/etc/services`

Refer to section 3.2 and section 3.3 for more information about these files.

3.1 Host Naming Conventions

Each host in the network must be identified according to TCP/IP naming conventions. These conventions include two ways of identifying a network host: host name and internet address. The host name and the internet address *must* be set in the `/etc/hosts` file and may be set in the `/etc/hosts.equiv` file, using the `sysadm` menu.

3.1.1 Host Name

The host name (also called the node name) is initially set using the `setup` command during the installation of the INTERACTIVE UNIX Operating System. It can also be set with the `hostname` command or with the `syssetup` menu of the `sysadm` utility. A host name may be chosen arbitrarily. It is used primarily as a way for users to refer to a particular machine. For example, `ism780`, `marketing`, and `engineering` are all possible host names. To display a host name, use the `uname -n` command:

```
$ uname -n
marketing
$
```

3.1.2 Internet Address (Software "Logical" Address)

The internet address is a 32-bit (4-byte) number that includes a network number and a local address and uniquely identifies a host machine. Each internet address consists of two, three, or four parts. Each part is separated by a dot, for example, 128.212.32.1. The network number identifies your network to other networks and may consist of up to three parts of the internet address. In the above example, "128.212" is the network number; all hosts on a local network must have the same network number. The local address consists of the rest of the internet address and includes a node number, which identifies the host within the network and possibly a subnetwork number as described below. The node number is an arbitrary number assigned by the system administrator (the "32.1" in the above example).

☛ If your network will not be communicating with outside networks or already has an assigned network number, it is *not* necessary to obtain a network number from NIC (see below); the network number and local address may be selected arbitrarily. The network number should be the same for all machines in the network; the node number should be unique for each machine.

If your network is to communicate with other large networks, such as ARPANET, you must obtain a unique network number from the following institute:

Hostmaster
DDN Network Information Center (NIC)
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
1-800-235-3155

HOSTMASTER@SRI-NIC.ARPA

The network will be classified as A, B, or C and a network number assigned.

- Class A
  A class A internet address begins with a number in the range of 0-127 and uses only the first byte for the network number; the other three bytes are available for the local address, which is assigned by the system administrator. Class A network numbers

are used for very large networks, such as ARPANET. A host on a class A network might have any of the following internet addresses:

```
57.0.1.1
57.0.1
57.1
```

In each case, the network number (the first byte) identifies the network as a class A network. All subsequent numbers are interpreted as part of the local address. The last three bytes are interpreted as the node number and should uniquely identify the machine.

- Class B
  A class B internet address begins with a number in the range of 128.1 to 191.254 and uses the first two bytes for the network number; the other two bytes are available for the local address, which are assigned by the system administrator. Class B addresses are typically assigned to most large organizations. The following is an example of a class B internet address:

  ```
  150.8.8.16
  ```

  The first two bytes identify the network as a class B network. All subsequent numbers are interpreted as part of the local address.

- Class C
  A class C internet address begins with a number in the range of 192.1.1 to 223.255.254 and uses the first three bytes for the network number; the last byte is used for the local address, which is assigned by the system administrator. The following is an example of a class C internet address:

  ```
  192.20.0.1
  ```

  The first three bytes identify the network as a class C network. The last byte is interpreted as the local address.

A subnetwork is a local interpretation of the internet address that allows bits from the local address to be used to extend the network number some additional bits, providing a subnetwork identifier.

Subnetworking provides a way for a large network to communicate with smaller networks through a *gateway*, rather than individually. A gateway serves as a way of routing information between smaller and larger networks. On large networks, such as ARPANET, there are a limited number of gateway routing table entries available;

therefore, subnetworking can be used to subdivide an internet address into network, subnetwork, and local address parts with the subnetwork identifier interpreted locally. Instead of communicating with each subnetwork individually, the larger network routes all information for hosts within the subnetwork to the gateway, which then redistributes the data according to the subnetwork identifier and the local address.

For example, a class B network might have the following internet address:

```
128.212.0.0
```

Subnetworks within that address might be:

```
128.212.16.1
128.212.32.2
128.212.46.3
128.212.10.4
```

The first two bytes are the network number, the third byte is the subnetwork identifier, and the fourth byte is the local address (node number).

For a typical example of a network configuration, refer to Figure 1.

For more information about internet addresses and subnetworks, refer to the following Request For Comment (RFC) documentation available from NIC:

- RFC997 *Internet Numbers*
- RFC950 *Internet Standard Subnetting Procedure*

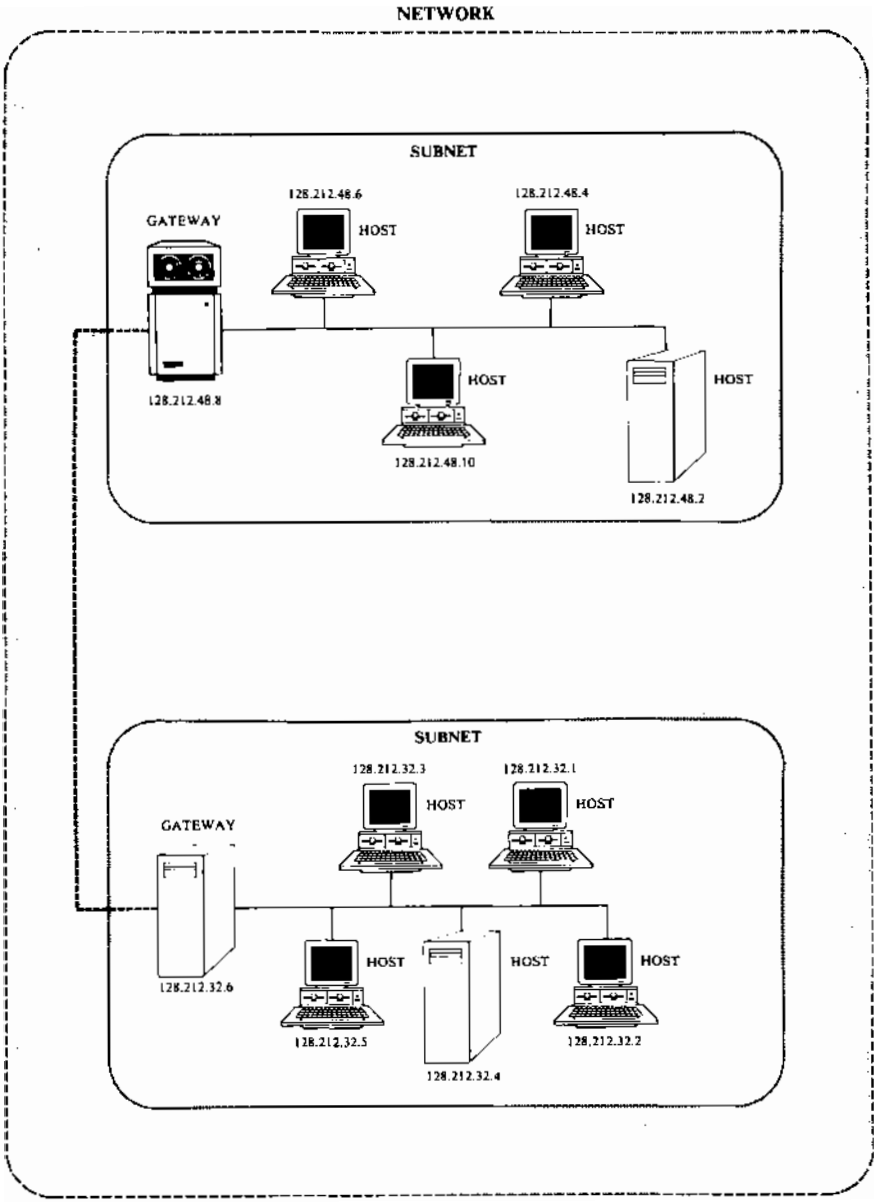


Figure 1. Diagram Showing a Typical Network Configuration

3.2 Configuring the Network

In order to make a machine known to the rest of the network, the `/etc/hosts` and the `/etc/hosts.equiv` files must be modified to include an entry for that host. This is done using the `sysadm` facility. This section describes the `/etc/hosts` and `/etc/hosts.equiv` files and explains how to use the `sysadm` utility to make changes to these files.

3.2.1 `/etc/hosts`

The `/etc/hosts` file lists all hosts in the network. An entry is required for each host (including the local machine). Entries must have the following format:

```
internet_address  host_name  aliases  [comments]
```

`internet address`
: The internet address is the logical address of the host, which consists of the network number (including subnetwork number and sub-subnetwork number, if any) and the local address (including the node number). Refer to section 3.1.2 for information about how to obtain an internet address.

`host name`
: The host name is an arbitrarily selected name set by the system administrator. For example, the host name for a machine that is used by the marketing staff of a corporation might be `marketing`.

`aliases`
: Aliases are nicknames for the host, separated by blanks. They are used to reduce the amount of typing required when referring to the host.

`comments`
: Comments must begin with the # character. They are used to provide descriptive information about a host.

The following table shows a sample `/etc/hosts` file:

```
127.1          local          localhost
193.16.14.1    marketing      mktg   m    # Marketing VAX running Ultrix
193.16.14.2    training       trng   t    # Training 386 INTERACTIVE UNIX
193.16.14.3    accounting     acct   a    # Accounting 3B2 running UNIX v.3
193.16.14.4    engineering    eng    e    # Engineering VAX running BSD 4.3
```

An `/etc/hosts` file must exist on each host in the network.

3.2.2 `/etc/hosts.equiv`

The `/etc/hosts.equiv` file establishes "equivalent" accounts among hosts on the network. It enables users other than `root` on other hosts to access the local host without passwords. If a remote machine is listed in the `/etc/hosts.equiv` file located on the user's local host, the user may use `rlogin` (refer to *rlogin*(1C)) to log in to the remote machine. The user will not have to provide a login ID if the user has been assigned the same login ID on both the local and remote hosts. The `/etc/hosts.equiv` file located on the local host `engineering` might look like this:

```
accounting
marketing
```

The entries in this file specify that users on hosts `accounting` and `marketing` who maintain a login ID on host `engineering` can use the `rlogin` command to easily access `engineering`.

Other commands that use the `hosts.equiv` file are `rcp` and `rsh`.

☛ Note that using `hosts.equiv` and `.rhosts` files is considered to be a security problem; they should not be used on an open network.

3.2.3 *Using* `sysadm` *to Modify Configuration Files*

The `sysadm` utility can be used to modify the configuration files `/etc/hosts` and `/etc/hosts.equiv`.

1. Use the `sysadm` command to display the System Administration menu, then select the `packagemgmt` menu. Your screen should look similar to this:

```
                    PACKAGE MANAGEMENT

 1 lpmgmt          print spooler management menu
 2 lprmgmt         Berkeley-style print spooler management menu
 3 netdrvrmgmt     Network Driver management menu
 4 nfsmgmt         NFS management menu
 5 tcpipmgmt       extended networking utilities menu
 6 uucpmgmt        basic networking utilities menu

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT:
```

2. From the `packagemgmt` menu, select the `tcpipmgmt` submenu:

```
                EXTENDED NETWORKING UTILITIES MANAGEMENT

  1 equivmgmt       manage tcp/ip hosts.equiv entries (list, add, delete)
  2 hbtcpmgmt       INTERACTIVE TCP/IP setup
  3 hostmgmt        manage tcp/ip hosts entries (list, add, delete)

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT:
```

3. The first option, `equivmgmt`, allows you to list, add, or delete entries from the `/etc/hosts.equiv` file:

```
This procedure is used to list, add, and delete entries in
the Extended Networking Utilities '/etc/hosts.equiv' file.

Type 'q' at any time to quit the current operation.
If a '?' appears as a choice, type '?' for help.

If a default appears in the question, type <ENTER> for
the default.


Enter the operation you want to perform:

        1 list
        2 add
        3 delete

(default list)[q]:

Press the ENTER key to see the tcpipmgmt menu [?, ^, q]:
```

Whenever a host is added to or deleted from the network, this file should be modified on each host in the network.

4. The third option, `hostmgmt`, allows you to list, add, or delete entries from the `/etc/hosts` file:

```
This procedure is used to list, add, and delete entries in
the Extended Networking Utilities '/etc/hosts' file.

Type 'q' at any time to quit the current operation.
If a '?' appears as a choice, type '?' for help.

If a default appears in the question, type <ENTER> for
the default.


Enter the operation you want to perform:

        1 list
        2 add
        3 delete

(default list)[q]:

Press the ENTER key to see the tcpipmgmt menu [?, ^, q]:
```

Whenever a host is added to or deleted from the network, this file should be modified on each host in the network.

3.3 Other Default Files

Three other default files are included in the INTERACTIVE TCP/IP software package: `/etc/networks`, `/etc/protocols`, and `etc/services`. These files are described below.

3.3.1 `/etc/networks`

The `/etc/networks` file contains the network names and numbers of those networks your network communicates with. The file should contain one entry per network, including the local network. Entries have the following format:

```
network_name    network_number  [network aliases] [# comments]
```

The following is an example of an `/etc/networks` file:

```
#
# Internet networks
#
loop    127     loopback
iconet  128.212.32
```

This file is used by programs concerned with internetworking, such as `route`, `ifconfig`, `gated`, `named`, and `netstat`.

A list of network and host names may be obtained from the NIC for users on the Internet.

3.3.2 `/etc/protocols`

The `/etc/protocols` file contains a list of protocols used on the network. Each protocol is associated with a unique, identifying

number. The most commonly used protocols are `TCP`, `IP`, and `UDP`. The default `/etc/protocols` file should look similar to this:

```
#
#  Internet (IP) protocols
#
ip      0       IP      #  internet protocol, pseudo protocol number
icmp    1       ICMP    #  internet control message protocol
ggp     3       GGP     #  gateway-gateway protocol
tcp     6       TCP     #  transmission control protocol
egp     8       EGP     #  exterior gateway protocol
pup     12      PUP     #  PARC universal packet protocol
udp     17      UDP     #  user datagram protocol
hmp     20      HMP     #  host monitoring protocol
```

The `/etc/protocols` file is provided as part of the software package and should not be modified.

3.3.3 /etc/services

The `/etc/services` file contains a list of services available on the network. Each service is associated with a unique, identifying number, and each service invoked uses one of two protocols to do data transfer. The two protocols used are `TCP` and `UDP`. The `/etc/services` file contains entries similar to these:

```
#  Network services, Internet style
#
echo            7/udp
ftp             21/tcp
telnet          23/tcp
name            42/tcp      nameserver
whois           43/tcp      nicname
hostnames       101/tcp     hostname      # usually from sri-nic
#
#  Host-specific functions
#
tftp            69/udp
finger          79/tcp
link            87/tcp      ttylink
ingreslock      1524/tcp
#
#  UNIX-specific services
#
exec            512/tcp
login           513/tcp
shell           514/tcp     cmd           # no passwords used
who             513/udp     whod
ntalk           518/udp
```

The `/etc/services` file is provided as part of the software package. Care should be taken when modifying this file.

To invoke certain utilities, such as `echo`, enter `telnet` followed by the host name and its corresponding port number, as listed above. In the above example, the port number for `echo` is 7.

After typing `telnet` *hostname* 7, the user will be in the `echo` program and may enter text to be echoed.

3.4 Advanced Configuration Topics

The topics covered in this section address issues that arise in a fairly complex network environment involving multiple networks, dissimilar TCP/IP implementations, or subnetworking. A simple Local Area Network made up of similar host machines running the same version of INTERACTIVE TCP/IP will not need these features.

3.4.1 Setting the Subnet Mask

If the host is to be used in a network with subnets, the software must know which parts of the internet address contain the network and subnet numbers. This is done by setting the subnet mask. One of the last lines in the `/etc/netd.cf` file is the `ifconfig` line for the network interface that you are using (refer to *ifconfig*(1M)). The correct `ifconfig` line to change is the one whose first parameter to `ifconfig` is the abbreviation for the network driver that you are using (such as `wd0` for a Western Digital board).

The netmask parameter and the network mask should be added to this `ifconfig` line. For example, if you are using network number 128.212, a Class B network, you have two bytes of local address and could use the first byte for a subnet number. In that case, you would set a mask using dot notation containing three full bytes (network and subnet numbers) and add the two parameters `netmask 255.255.255.0` to the `ifconfig` line in `/etc/netd.cf`.

3.4.2 Setting a Broadcast Address

The broadcast address used by default in INTERACTIVE TCP/IP is the internet address of the host, with the node number consisting entirely of ones. If the subnet number 32 is added to our previous example of network number 128.212, the default broadcast address would be 128.212.32.255. If you must change this value to be compatible with older TCP/IP hosts on your network, you can set the broadcast address in the `ifconfig` line of the `/etc/netd.cf` file, as described in section 3.4.1. For example, to use a broadcast address of all ones, add the parameters `broadcast 255.255.255.255` to the `ifconfig` line.

3.4.3 Switching Network Boards

If you must change the network board on your host, the software may need reconfiguration to access the new board. If you use the same kind of board with the same jumper settings, no changes should be required. However, other machines may not recognize your new board until their `arp` table entry for your old board times out (refer to *arp*(1M) for instructions on how to delete the old `arp` table entries). If you use the same kind of board with different jumper settings, you should reconfigure the board through the Network Drivers extension.

If you replace your board with a different type of board, there are several steps that are required:

1. Reconfigure the board through the Network Drivers extension.
2. Use the `rminterface` option of the `hbtcpmgmt` menu to remove the interface for the old board from your system.
3. Use the `addinterface` option of the `hbtcpmgmt` menu to configure your new board. Refer to section 2.2, "Configuring the TCP/IP Interface," for more information about adding and removing interfaces.

3.4.4 Using and Running a Nameserver

If the host is to be run as part of a network that changes configuration often, it may be easier to use a nameserver on one machine instead of changing the `/etc/hosts` file on each machine for each configuration change. The INTERACTIVE TCP/IP software is configured to call a nameserver if you change the file `/etc/resolv.conf` (refer to *resolver*(5)). To use a remote nameserver, replace the `nonameserver` line with the `nameserver` option followed by the internet address of the host running the nameserver.

To use a local `nameserver`, configure `/etc/named`, automatically start `named` in `/etc/rc3.d/S05hbtcp`, and delete the `/etc/resolv.conf` file. Refer to *named*(1M), "Name Server Operations Guide for BIND," and "Domain Administrator's and Operations Guide."

4. ACTIVATING TCP/IP AND USING NETWORK APPLICATIONS

The system is normally configured to run at level 2, which is specified by an entry in the file `/etc/inittab`. The TCP/IP software, however, is configured to run at level 3, which activates networking programs. After the INTERACTIVE TCP/IP software is installed, the entry in the `/etc/inittab` file is updated to specify level 3:

```
is:3:initdefault:
```

If you installed the software properly and initiated a system shutdown and reboot, the TCP/IP software should be running properly. However, if you made any changes to the `/etc/inittab` file or are experiencing problems with the TCP/IP software, make sure that the entry in the `/etc/inittab` file specifies level 3, not level 2.

4.1 `/usr/ucb`

The standard Berkeley utilities are installed in the directory `/usr/ucb`. In order to use the utilities, you must either change your search path to include the `/usr/ucb` directory or type the full path name explicitly. Because the `/usr/ucb` path name has been compiled into some of the utilities, moving them is not recommended. However, copies of the utilities may be put elsewhere.

4.2 Initiating SLIP Over Modem and Serial Lines

4.2.1 Dialing Out

To make a connection via `sldialup` (refer to *sldialup*(1M)):

1. As user `root`, type: `sldialup <port> <baud>` where, `<port>` is the ttyname parameter and `<baud>` is the speed of the modem conected to the terminal line. For example, to use a 2400 baud modem connected to `/dev/tty00` type:

   ```
   sldialup tty00 2400
   ```

2. You are now talking directly to the modem. Give the commands to dial the host. For example, if you are using a Hayes*-type modem, and the phone number is 555-1212, type:

   ```
   ATDT 5551212
   ```

3. When connected, press [ENTER] a few times to get a `login:` prompt. You may have to press [CTRL] [b] (to send a break signal) to change the baud rate.

4. Log in with the account set up on the target system. The account on the target system should have `sllogin` as its login shell. (Refer to *sllogin*(1M) for further information.)

5. When logged in, press [ESC] to return to the # prompt on the local system. You should now be connected.

6. If you are using one system to act as a gateway between a SLIP and a non-SLIP network, then you need to start up the gateway routing daemon. As user `root`, type:

```
/etc/gated
```

7. It takes about 45 seconds to set up the routing tables for the network on the target system. You can see when they are there by typing:

```
netstat -r
```

   Refer to *netstat*(1) for further information.

At this point, the network should be connected between the two systems.

4.2.2 Direct Lines

Direct lines are connected using `slattach` (refer to *slattach*(1M)). If you have configured SLIP, then `slattach` should be started automatically when the network is started. However, if for some reason it needs to be started manually, follow these steps:

1. Type: `slattach <port> <baud>`, where `<port>` is the ttyname parameter and `<baud>` is the optional baud rate parameter used to set the speed of the connection. For example, to use a baud rate 2400 on `/dev/tty00` type:

```
slattach /dev/tty00 2400
```

2. If you are using one system to act as a gateway between a SLIP and a non-SLIP network, then you need to start up the gateway routing daemon. As user `root`, type:

```
/etc/gated
```

3. It takes about 45 seconds to set up the routing tables for the network on the target system. You can see when they are there by typing:

```
netstat -r
```

   Refer to *netstat*(1) for further information.

At this point, the network should be connected between the two systems.

5. CONFIGURING LPR

Version 1.3 of INTERACTIVE TCP/IP includes a Berkeley-style print spooler that allows the user to access remote printers over a network. Through the `lprmgmt` menu, you can configure `lpr` to add or remove remote printers and to start up the `lpd` daemon automatically.

For information about how to use `lpr` to drive local printers, refer to the "Line Printer Spooler Manual" in this guide.

1. To configure `lpr`, first select the `packagemgmt` option from the System Administration menu. Your screen will look similar to this:

```
                    PACKAGE MANAGEMENT

 1 lpmgmt          print spooler management menu
 2 lprmgmt         Berkeley-style print spooler management menu
 3 netdrvrmgmt     Network Driver management menu
 4 nfsmgmt         NFS management menu
 5 tcpipmgmt       extended networking utilities menu
 6 uucpmgmt        basic networking utilities menu

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT:
```

2. Select option 3 `lprmgmt`. Your screen will look similar to this:

```
            Berkeley-style Line Printer Administration Menu

            1       setup to run lpd at startup
            2       don't automatically run lpd at startup
            3       add a remote printer
            4       remove a printer

    Enter selection ("q" to quit):
```

3. The `lprmgmt` menu provides you with several options for managing `lpr`. In order to use `lpr`, you must first add the names of the remote printers you want to access. Select option 3. Your screen will look similar to this:

```
Add a remote printer:

This option will allow you to enable access to a remote
printer from this system.  In order to do this, you must
know the name of the remote system which controls the
printer and the name used by the remote system to specify
that printer.  Additionally, the remote system must have
this system's name specified in either the /etc/hosts.equiv
or /etc/hosts.lpd files.
```

4. The system will now display a number of questions about the remote printer you want to access. Enter the name of the remote printer, a description of the remote printer, the name of the remote host, the name of the remote printer on the remote host, and the directory where the printer jobs should be stored. Note that, in some cases, sample answers are provided in the following screen. Press **ENTER** to accept the default values. Your screen will look similar to this:

```
What is the name of the printer to add? Laser
What is the description of the printer
  (e.g. "Joe's printer")? Laser Printer
What is the remote host to send to? host2
 is this printer called on host2 [laser]?
What directory should be used to store printer jobs
  [/usr/spool/lpd/laser]?

Remote printer "laser" has been added.

        Berkeley-style Line Printer Administration Menu

        1       setup to run lpd at startup
        2       don't run automatically lpd at startup
        3       add a remote printer
        4       remove a printer

Enter selection ("q" to quit):
```

5. Once a remote printer has been added through option 3, you can enable the line printer system to start automatically when the system is booted. Select option 1. Your screen will look similar to this:

```
The line printer system will start the next time the system
is booted.

Do you want to start the printer system now [y]?
```

6. Type y or press **ENTER** to enable the line printer system.

7. If you want to disable option 1 (i.e., stop the line printer system from being started when the system is booted), select option 2 from the `lprmgmt` menu. Your screen will look similar to this:

```
The line printer system will no longer be started when the
system is booted

Do you want to stop the printer system now [y]?
```

8. Type `y` or press ENTER to stop the printer system.

9. To remove a printer from the system, select option 4. Your screen will look similar to this:

```
Remove a printer:

This option will allow you to remove a printer from this
system.  In order to do this, you must know the name of the
printer to be removed.  If any jobs are waiting to be printed
on this printer, you will be asked for confirmation before
continuing.  The printer information will be deleted from
/etc/printcap and the spool directory will be removed.

Which printer should be removed? Laser
```

10. Enter the name of the remote printer you wish to remove. Your screen will look similar to this:

```
Removing spool directory /usr/spool/lpd/laser
Removing "laser" from /etc/printcap.
Printer "laser" has been removed.
```

11. Press `q` to quit the `lprmgmt` menu, then press ENTER to return to the `packagemgmt` menu.

INDEX

Setting Up a Gateway on an INTERACTIVE UNIX System With TCP/IP

CONTENTS

Setting Up a Gateway on an INTERACTIVE UNIX* System With TCP/IP

1. INTRODUCTION

This document describes how to create a gateway on an INTERACTIVE UNIX Operating System using INTERACTIVE TCP/IP and two dumb Ethernet* boards.

1.1 Who Should Use This Document

This document assumes that the user has a basic understanding of internet addressing and of how subnets work. A knowledge of kernel configuration will also prove helpful.

1.2 Pre-Installation Information

The gateway should be set up on an INTERACTIVE UNIX System with INTERACTIVE TCP/IP that is connected to a network and running without problems. This document discusses adding a second board to the system, being able to communicate to both networks at once, and various strategies to "gateway" between networks. It is strongly recommended that a gateway be set up in stages. This will prevent many potential problems from occurring later.

If INTERACTIVE TCP/IP has not yet been installed on your system, refer to the "INTERACTIVE TCP/IP System Administrator's Manual" in this guide for information on this procedure. After the installation of TCP/IP, return to this document.

Although other boards are also presumed to work, two Ethernet boards that have been verified in a gateway configuration are the 3COM* 3C503 EtherLink* II Board and the Western Digital Ethernet Board WD8003E. Contact your distributor for a complete list of supported boards.

Each Ethernet board in the system must have its own unique registered internet protocol (IP) number and host name. For more information, refer to section 3, "CONFIGURING A NETWORK," in the "INTERACTIVE TCP/IP System Administrator's Manual." Obtain an IP address for the second network board. This number must be obtained before the installation because the network will need it to bind to the second board.

2. INSTALLING A SECOND ETHERNET BOARD OF A DIFFERENT TYPE

If you plan to use two boards of the same type, skip this section and go to section 3, "Installing a Second Ethernet Board of the Same Type."

This section discusses setting up a gateway with a first board and second board that are not of the same type, for example a WD8003E and a 3C503. To begin installation of the second network board, follow the basic steps that you used to install the first board (refer to the *INTERACTIVE Network Drivers Overview and Installation Instructions*). The board must be installed in the machine and the appropriate driver enabled:

1. Log in as `sysadm` and select the `packagemgmt` menu option from the System Administration menu. (If you are not at the System Administration menu display, use the `sysadm` command or log in as `sysadm` to access this menu, then select `packagemgmt`.) Your screen will look similar to this:

```
                        PACKAGE MANAGEMENT

 1 lpmgmt          add line printer
 2 rfsmgmt         RFS management menu
 3 tcpipmgmt       extended networking utilities management menu
 4 uucpmgmt        basic networking utilities menu
 5 vpixmgmt        install users for the VP/ix Environment

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT:
```

2. Select the `tcpipmgmt` option from the `packagemgmt` menu. Your screen will look similar to this:

```
                EXTENDED NETWORKING UTILITIES MANAGEMENT

  1 equivmgmt      manage tcp/ip hosts.equiv entries (list, add, delete)
  2 hbtcpmgmt      INTERACTIVE TCP/IP setup
  3 hostmgmt       manage tcp/ip hosts entries (list, add, delete)

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT:
```

3. Select the `hbtcpmgmt` option. The system will display the names and a description of network boards supported by the INTERACTIVE UNIX Operating System. Your screen will look similar to this:

```
                    INTERACTIVE TCP/IP SETUP

  1 el              3COM Etherlink II (3C503) Board
  2 no              No Network Board (loopback only)
  3 ub              Ungermann-Bass PC NIC
  4 wd              Western Digital EtherCard PLUS

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT: 1
```

The screen displays options for a number of Ethernet boards that can be configured into the kernel. When you select one of these options, you are guided step-by-step through the process of modifying the corresponding device-dependent configuration files listed in the `/etc/conf/sdevice.d` directory. Your responses to the questions are entered as values in specific fields within these files; therefore an understanding of the fields contained in these files is helpful in successfully reconfiguring the kernel. For more information on the format of these files and an explanation of each field, refer to *sdevice*(7). The configuration process for supported boards is outlined in the *INTERACTIVE Network Drivers Overview and Installation Instructions*. Refer to the appropriate section for more information about how to configure your particular driver.

As you did when you installed the first board, select the board that is to be installed as the second network board and answer all questions concerning board base I/O address, shared memory, and interrupt levels. Be careful to pick an interrupt request level (IRQ) and memory address that do not conflict with the first Ethernet board's values or the values of any other device in the system. Because the first board and second board are not of the same type, you will have to remake your system kernel. Respond `yes` when asked if you wish to rebuild the kernel at this time. This will add in *both* Ethernet board drivers. Add the second Ethernet board to the machine and reboot the system with the new kernel. Skip the next section and continue with section 4, "Testing the Second Board."

3. INSTALLING A SECOND ETHERNET BOARD OF THE SAME TYPE

This section is specific to the INTERACTIVE UNIX System Version 2.2 environment. Familiarity with the kernel configuration setup of the 386/ix (386/ix* Operating System Release 2.0 or later) is required.

Installing a second Ethernet board of the same type is somewhat more complicated than installing a second Ethernet board of a different type. It requires changes in the device driver's configuration files.

After installing the second board in the machine, change directories (`cd`) to `/etc/conf/pack.d`. Make a directory `device_name`*2*, where `device_name` is the name chosen for the second board. This will be a "dummy" entry to satisfy configuration requirements for kernel builds. Create an empty file named `space.c` in the directory `device_name`*2*. To summarize, here are the steps taken so far, using the WD8003E (`wd`) board as an example:

```
$ cd /etc/conf/pack.d
$ mkdir wd2
$ cd wd2
$ vi space.c

:wq!
```

The `space.c` file must be created with an editor such as `vi` because the C compiler requires at least one line in a file. Now compile the file `space.c` into `Driver.o` it will link by typing:

```
$ cc -c space.c
$ mv space.o Driver.o
```

Change directories to `/etc/conf/pack.d/device_name`. The real configuration for both boards is kept in the file `space.c` in this directory; you must configure the file `space.c` with values for both Ethernet boards. Edit the `space.c` file. Find the defined variable line for the number of Ethernet boards. It will look similar to this:

```
#define NWD     1       /* Number of WD 8003 boards */
```

Change the 1 to 2.

Now find the structure declaration that defines the board parameters. It will look similar to this:

```
struct wdparam wdparams[NWD] = {
  {
   0,                        /* board index */
   WDIRQ0,                   /* interrupt level */
   WDBASEPORT0,              /* I/O port for device */
   (caddr_t)WDBASEADDR0,     /* address of board's memory */
   WDBOARDSIZE0,             /* memory size */
   0,                        /* pointer to mapped memory */
   0,                        /* board type */
   0,                        /* board present flag */
   0,                        /* board status */
   0,                        /* number of streams open */
   WDMAJOR0,                 /* major device number */
   NSTR                      /* number of minor devices allowed */
   }
};
```

For each board of this type in the system, a parameter block is required. To create this second board's parameter block, copy the lines above and place the copy directly below it. Remember to add a comma between the curly braces of each set and check for balanced braces. Now change the parameters that differ between boards. For the WD8003E, these are the interrupt level and the I/O port. A new major device number must also be given. Use a device number that is not in use anywhere else in the system. (Scan the `/dev` directory using `ls -l` to see the device numbers.) In most systems, major code `55` is free. Below is a sample block for two WD8003E boards. See the appendix for complete examples of `space.c` configurations for the WD8003E.

☛ It is important to hard-code all values in this structure. Do not use symbolic constants or define labels.

Since `kconfig` (see *kconfig*(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*) has no concept of two boards of the same type, it will not have symbols and/or defines for the second board. You will have to supply these. Because most boards have values set up as hex addresses, constants must include the prefix `0x` in front of addresses.

```
struct wdparam wdparams[NWD] = {
 {
   0,                    /* board index */
   3,                    /* interrupt level */
   0x280,                /* I/O port for device */
   (caddr_t)0xd0000,     /* address of board's memory */
   8192,                 /* memory size */
   0,                    /* pointer to mapped memory */
   0,                    /* board type */
   0,                    /* board present flag */
   0,                    /* board status */
   0,                    /* number of streams open */
   35,                   /* major device number */
   NSTR                  /* number of minor devices allowed */
 },
 {
   1,                    /* board index */
   7,                    /* interrupt level */
   0x2a0,                /* I/O port for device */
   (caddr_t)0xd2000,     /* address of board's memory */
   8192,                 /* memory size */
   0,                    /* pointer to mapped memory */
   0,                    /* board type */
   0,                    /* board present flag */
   0,                    /* board status */
   0,                    /* number of streams open */
   55,                   /* major device number */
   NSTR                  /* number of minor devices allowed */
 }
};
```

Before the kernel can be built, several changes need to be made to the `conf` directory and its files in order to use `kconfig`. Change directories to `/etc/conf/sdevice.d`. `kconfig` needs to find an entry for the second Ethernet device (`wd2`) in it. Create the file `wd2`, for Western Digital boards. Copy the contents of the `wd` file into it. Now the file `wd2` will look similar to:

```
wd   Y   32   5   3   4   320   32f   dc000   ddfff
```

Because all of the configuration values have been hard-coded in the `space.c` file, the values in the file `sdevice` have no direct effect on `kconfig`. However, the values must be changed to reflect the settings on the second board to prevent `kconfig` from reading incorrect values from `sdevice` entries and detecting a false conflict. (Simply copying `wd` to `wd2` does not work because `kconfig` would detect two boards — `wd` and `wd2` — on the same interrupt.) These files must be present if `kconfig` is to build a kernel. Also, because INTERACTIVE TCP/IP does not inform `kconfig` about devices, `kconfig` will see two boards with the same device name (`wd`) from files `wd` (the original) and `wd2`. Change the `wd` to `wd2` in the file `wd2` to prevent this confusion. The `wd2` file should now contain:

```
wd2  Y  32  5  3  4  320  32f  dc000  ddfff
```

Because two `sdevice` files are now present, `kconfig` will expect two `mdevice` entries, and if these are not present it will issue an error message. However, since only one device driver is present, two `mdevice` entries are not really required. To prevent more error messages from `kconfig`, edit the file `/etc/conf/cf.d/mdevice`. Search for a line containing `wd` that looks like:

```
wd  I  iSHOco  wd  0  35  1  32  -1
```

Duplicate this line:

```
wd  I  iSHOco  wd  0  35  1  32  -1
wd  I  iSHOco  wd  0  35  1  32  -1
```

In the second line, change the `wd` in the first field to `wd2` to inform `kconfig` that two boards are in the system. Change the sixth field to refer to the actual major device number for the second board. In the examples below, `55` is used:

```
wd  I  iSHOco  wd  0  35  1  32  -1
wd2 I  iSHOco  wd  0  55  1  32  -1
```

At this point, a device node for the second Ethernet board needs to be made. Change directories to `/etc/conf/node.d`. Create a file named `wd2` that contains the line:

```
wd2  clone  c  55
```

This line requests that a node be made which is a streams device, attached to minor number `55`. The name of the new node is `/dev/wd2`.

At this point, the `kconfig` program may be run to build the kernel. Error messages concerning duplicate defines may be received during the build but this is *not* an error condition. `kconfig` is warning the user that it has detected two boards using the same defines in the `space.c` files. In this case, issuing the messages is the proper response.

4. TESTING THE SECOND BOARD

To test for correct installation and function of the second Ethernet board, bring the system to run level 2 (`init 2`).

Verify your IP address for the second network board. You will need this number during the gateway process. Edit `/etc/hosts` and add an entry for the second network with its IP address. Because there are two network boards involved, the machine must have two host names and two host IP numbers. Both sets of these numbers must appear in `/etc/hosts`. In addition, each network must have a different subnet IP value. For class C networks, it is *not* legal to have a network *A* with a number 192.31.146.10 and a network *B* with the number 192.31.146.11. These are host IP values. Network numbers are three bytes or less in length. For example, a host IP might have the value 128.212.16.14, whereas the class C network number for that would be 128.212.16.

`/etc/netd.cf` is the configuration file used for network setup. You must enter your second host name and IP address in the `/etc/netd.cf` file as well. Using an editor such as `vi`, edit `/etc/netd.cf` and replace the host name and IP address of the first board with the host name and IP address of the second board. For the purposes of this test, you must also change the name of the machine. To do this, use the command:

```
uname -S newname
```

Now bring the system up to run level 3 (`init 3`) and test to see that the machine is communicating with hosts on the second network.

The second board is now installed and tested. You are now ready to set up the netmask and broadcast values for both network boards.

5. BROADCAST ADDRESSES AND NETMASK VALUES

The *broadcast* and *netmask* fields are important. The broadcast field is the address the system will use to communicate with the network at large. If there are less than 255 machines on the network, the network is most likely a class C network. In a class C network address, the first three numbers are assigned by a network coordinator and the network is limited to 255 machines. The network class will be determined by the networking environment and by whether TCP/IP connections to the outside world such as internet are present. The site network administrator should have information concerning the network's broadcast address format.

For class C networks, the broadcast field should be `w.x.y.255` or `w.x.y.0`, where `w.x.y` is the first three bytes of the network address assigned to it. Choosing 0 or 255 depends on whether the network uses "all 0s" or "all 1s" as a broadcast address. The 386/ix will select the proper broadcast address based on the netmask given to it. If the network has broadcast addresses in one of these forms, this information will be provided automatically once the netmask value is set. Broadcasts are "wildcard" addresses to the network.

If the broadcast field is the network wildcard, the netmask value specifies which addresses belong to a particular network. Anything outside of the netmask value belongs outside of the network and should be treated as such, whereas anything within a netmask can be considered local to that network. In class A, B, and C networks, the number of bits actually allocated to a `host-id` IP number are different. A class C address has 12 bits of host information, whereas a class B network has more. The more bits allocated to `host-id` mapping, the more hosts you can represent on that subnet. A netmask value allows the interface to interpret the standard 32-bit number in such a way that it can determine which broadcasts are really meant for its subnet. Any broadcast not falling within the realm set by netmask is ignored.

Set up the netmask and broadcast values for both network boards and bring the system to run level 3. The machine is now capable of communicating with different networks. To allow the exchange of packets between networks, some form of routing must be used. Section 6 will discuss routing alternatives.

5.1 Activating Both Boards at Once

To activate both boards at once, first make a backup copy of `/etc/netd.cf` called `/etc/`*`netd.2nd`*.

Edit `/etc/netd.cf` and search for the following line:

```
arp    = "/dev/arp" lsap 0x800
```

Below it add this line:

```
arp2   = "/dev/arp" lsap 0x801
```

The address change from 0x800 to 0x801 is required. A new IP stream must be bound to the device if IP packets are to reach the second board. In the 386/ix, any `lsap` (logical service access point) value between 0x800 and 0x805 (inclusive) is an IP stream. Each device has its own IP stream `lsap`.

Next, search for every occurrence of the device name `arp` and replace it with the name of the second board, for example `arp2`, throughout the entire file. In addition, any line referencing a device name, such as `wd0` for the WD8003E board, `el0` for the 3C503 board, and `ub0` for the Ungermann-Bass PC-NIC board should be duplicated. For each of these lines, change the second copy to the code for the second Ethernet board. The file should look something like this:

Before editing:

```
# open link level devices
wd_arp = "/dev/wd" lsap 0x806  # stream for ARP messages
wd_ip  = "/dev/wd" lsap 0x800  # stream for IP messages
loop   = "/dev/lo" lsap 0x800  # stream for loopback driver
```

After duplicating and changing occurrences of `wd` to `el` (*never* duplicate loop):

```
# open link level devices
wd_arp = "/dev/wd" lsap 0x806  # stream for ARP messages
wd_ip  = "/dev/wd" lsap 0x800  # stream for IP messages
el_arp = "/dev/el" lsap 0x806  # stream for ARP messages
el_ip  = "/dev/el" lsap 0x800  # stream for IP messages
loop   = "/dev/lo" lsap 0x800  # stream for loopback driver
```

Or:

After duplicating and changing occurrences of `wd` to `wd2` (*never* duplicate loop):

```
# open link level devices
loop    = "/dev/lo" lsap 0x800   # stream for loopback driver
wd_arp  = "/dev/wd" lsap 0x806   # stream for ARP messages
wd_ip   = "/dev/wd" lsap 0x800   # stream for IP messages
wd2_arp = "/dev/wd2" lsap 0x806 # stream for ARP messages
wd2_ip  = "/dev/wd2" lsap 0x800 # stream for IP messages
loop    = "/dev/lo" lsap 0x800   # stream for loopback driver
```

Finally, the last section must be duplicated to reflect the Ethernet name for that device. For each occurrence of a device with the keyword `link` in it, duplicate the line and replace the device name (after the keyword `name`) with the name of your second board.

Before editing:

```
# names must not assume a unit number so use as part of name
link loop under ip_tcp name "lo0"
link arp under ip_tcp name "wd0"
link wd_arp under arp type 0x101
link wd_ip under arp type 0x1
```

After editing:

```
# names must not assume a unit number so use as part of name
link loop under ip_tcp name "lo0"
link arp under ip_tcp name "wd0"
link arp2 under ip_tcp name "el0"
link wd_arp under arp type 0x101
link wd_ip under arp type 0x1
link el_arp under arp2 type 0x101
link el_ip under arp2 type 0x1
```

The `link` lines associate a token such as `arp` (the address resolution protocol, or Arp, device stream) with a particular name and a particular device. A link line could also be written as:

```
link arp under lsap 0x806 device "/dev/wd0"
```

Most users prefer to use symbolic names. Note that it does not matter which device uses `arp2` as long as the name is used in a consistent manner.

The `ifconfig` line must be copied and changed to give each board its proper name. It should look like this:

Before editing:

```
# initial configuration of interfaces
ifconfig "wd0" FIRST-NAME broadcast w1.x1.y1.z1 netmask 255.255.255.0 up
ifconfig "lo0" localhost up
```

After editing:

```
# initial configuration of interfaces
ifconfig "wd0" FIRST-NAME broadcast w1.x1.y1.z1 netmask 255.255.255.0 up
ifconfig "el0" SECOND-NAME broadcast w2.x2.y2.z2 netmask 255.255.255.0 up
ifconfig "lo0" localhost up
```

☛ See the appendix of this document for a sample `/etc/netd.cf` file that is set up for a gateway.

6. USING A GATEWAY BETWEEN NETWORKS

Once the ability to connect machines on either side of the "gateway" has been established, routing "bridges" must be created. Once a bridge is functioning, packets from one network may travel through the bridge to any other connected network. The first and simplest method of bridging networks requires that all network leaf nodes add the following to a startup script that is usually found in `/etc/rc3.d`. In the examples below, *128.212.19.1* has been chosen as the sample IP number of the gateway machine. You would replace *128.212.19.1* with the number for the gateway machine in your network:

```
route add default 128.212.19.1   1
```

This informs leaf nodes that a gateway machine in the network can route to any machine outside of that network. The gateway machines must run `/etc/gated` as well. This last step enables packet routing between networks.

With a larger network, all machines can become aware of the gateway topology by running the `gated` program. This allows each machine to receive routing updates from other machines on the network. Since each machine has a complete routing table, it can determine the best route for a packet using the multiple routes available to it. However, this option does increase the load on the network and, for large networks, it can increase the size of the kernel by as much as 25K.

Find the file `Sxxhbtcp` in the directory `/etc/rc3.d`. (*xx* is a positive number that the system uses to determine which files are activated first. Lower numbers receive priority over higher numbers – thus INTERACTIVE NFS* starts *after* INTERACTIVE TCP/IP.) Edit this file and search for the line:

```
if [ -x /etc/rwho ]
```

This part of the script starts up the `rwho` daemon. Just before this line, add the following, replacing *128.212.19.1* in the example below with the IP number of your gateway machine:

```
if [ -x /etc/gated ]      # For gateway machines
then
        /etc/gated &
fi
```

Or:

```
if [ -x /etc/route ]      # For leaf nodes
then
        /etc/route add default 128.212.19.1  1
fi
```

This starts the gateway daemon at run level 3. INTERACTIVE TCP/IP uses a superset of `routed` called `gated`. `routed` (the original gateway program) only understands one gateway protocol, Routing Information Protocol (RIP). `gated` can communicate not only with RIP, but also with Exterior Gateway Protocol (EGP) and HELLO as well. This is not a concern for most users; the file `gated.conf` is provided to perform the standard RIP protocol. If you have other gateways on your network, see section 7, "Configuring Gateways."

Before the gateway can route any packets, it must learn the current network topology. For single gateway systems, this information is obtained from interface data in the kernel itself; for multi-gateway systems, this information is obtained both from kernel interface data and from network broadcasts from other gateways. Gateways route packets based on the network that the packet is attempting to reach. These networks are addressed by their network number. Numeric representations can be difficult to remember, so the system provides a facility for mapping network numbers into network names. The file `/etc/networks` contains a mapping of network symbolic names to 32-bit network numbers. This file consists of the following:

```
#
# Comment lines begin with #
#
symbolic-net-name     Internet address of NET
symbolic-net-name     Internet address of NET
symbolic-net-name     Internet address of NET
```

The symbolic name can be any name used to identify subnetworks, but the net address has a very specific form. Internet hosts identify other hosts by a four byte (32-bit) number. Should any byte be missing, the internet assumes this number is a network address. For example, in a network with the address of the form 10.11.12.*x* (where *x* is a specific host), the number 10.11.12.15 refers to host 15 on the 10.11.12 NET. The number 10.11.12 requests that the packet be sent to *every host* on the 10.11.12.* host net. The number 10.11 means every host (i.e., 10.11.*.*) in the 10.11 network.

☛ Note that the numbers presented here are used as examples only and should *not* be used in an actual gateway. *10.** is the internet designation for the ARPANET. The ARPANET is used by the Department of Defense. *Do not use 10.* for a network.*

Consult with a network administrator for the proper network numbers and their names. Unless the network administrator does not intend to extend the network beyond a few sites, the network administrator should obtain a registered network number from SRI-NIC (Stanford Research Institute Name Information Center). There is no charge for this service. With this assigned number, organizations are assured:

1. No one else will use a network address even if the network is linked to a wide-area network like the internet.
2. Network numbers will be unique across a wide-area network.

The gateway is aware of the host names of the two networks it connects. It is also aware of any other gateways on these networks because gateways periodically broadcast routing information to each other over the network. The following example shows two networks, 10.11.12.* and 10.11.13.*: This file informs `routed` and `netstat` that the first-net (anything with 10.11.12) is given the name `first-net`. (The network is already aware of this information.)

```
#
# Sample /etc/networks
#
first-net       10.11.12
second-net      10.11.13
third-net       10.11.14
fourth-net      31
Bignet          64
```

The final step in starting gateway operations is to bring the system to run level 3. Both network boards will be activated, and `/etc/gated` will be activated as well. The kernel will detect two interfaces, and `/etc/gated` will be able to route between them. Should there be only one network interface in the system, `/etc/gated` has been designed to gather the routing information from other gateways on the network, and to not attempt to route out of a single interface.

For each leaf node (non-gateway node) in the network, a special program must be run to inform the node that the gateway exists. The simplest method is to have all nodes run `gated`. However, for

system efficiency, large installations may prefer to use the `route` command. Using `route` informs the kernel that a particular route exists to another network, but does not force the system to maintain routing information for all networks accessible from the gateway. In addition, `route` supports the notion of a "default route." When a packet has been sent to a destination outside the current network, if no other routes are available, it can be sent to the default route gateway. This allows leaf nodes to reduce routing overhead. The node need only be concerned with routing within its own network, and a gateway to anywhere else. To use the `route` command, add the line:

```
route add default 128.212.19.1 1
```

into the INTERACTIVE TCP/IP startup script `/etc/rc3.d/Sxxhbtcp` after the `rwho` lines, replacing *128.212.19.1* in the example above with the IP number of your gateway machine.

7. CONFIGURING GATEWAYS

As mentioned in section 6, `gated` uses a `gated.conf` control file which provides options to instruct it on issues such as gateway protocol and whether to broadcast. Options are one-per-line, but more than one option may be active in a file. While all of the options are not covered, some of the most important ones are discussed here:

- `RIP [NO ¦ QUIET ¦ YES]`
  This option determines whether the gateway uses the Internet Routed RIP protocol. This protocol is the oldest, most universal gateway protocol available. INTERACTIVE ships `gated.conf` with the `RIP` option enabled (`RIP YES`). This allows the 386/ix to work with a wide variety of dedicated routers and computers already in place on the internet. If this feature is not desired, placing `RIP NO` in the `gated.conf` file will disable it. The `RIP QUIET` line enables the `RIP` option, but the routing information broadcasts that normally occur with `RIP` are disabled. With `RIP QUIET` the gateway will process `RIP` information, but it will not inform other gateways on the network about its own routes. This option is useful if the network has a centralized gateway and does not want other sub-gateways sending out extraneous routing information since every site will use the central gateway. If the `gated.conf` file is missing, `/etc/gated` automatically uses `RIP YES`.

- `EGP [NO ¦ YES ]`
  This option allows `/etc/gated` to use the Exterior Gateway Protocol. This protocol is used for very large networks linked to wide-area networks such as the internet. With this option enabled, gateways can correct routing errors based not on the number of sites a packet must pass through (as in RIP), but on the real-time delay between when a packet is sent and when it will reach its destination. This allows gateways to change their routes based on information about links being slow due to congestion or link failure. The processor overhead is much higher than RIP, but it is a required gateway protocol for connection to the Internet Core Gateway System.

- `HELLO [NO ¦ YES]`
  This option determines whether `/etc/gated` will make use of the experimental `HELLO` protocol. The `HELLO` protocol was developed by the Defense Data Network (DDN) to aid in routing unstable networks. In these networks, routes can change in a matter of seconds. The `HELLO` protocol should only be used in network environments such as satallite links, packet radio links, etc. It is included primarily as a courtesy to older internet environments such as the DDN. Newer wide-area networks are using the Interior Gateway Routing Protocol (IGRP) to handle both `EGP` and `HELLO` functions as a unified protocol. Details on IGRP can be obtained from Cisco Systems Incorporated in San Francisco, California.

Appendix

In the following sample `netd.cf` files (sections 1 and 2 of this appendix), every line should be typed as a single line. However, due to the margin restrictions of this book, the lines in the `ifconfig` portion (under the comment "# initial configuration of interfaces") are too long to print as one line. These lines have been broken and indented here, but they should be typed as single lines.

1. AN EXAMPLE `netd.cf` FILE

```
#       @(#)netd.cf     1.4 - 88/11/15
#
# This is a configuration file for TCP/IP
# using two WD8003 boards as a gateway
#

# open control streams for transport protocols
tcp = "/dev/tcp"
udp = "/dev/udp"

# open an IP stream for each transport protocol
ip_tcp = "/dev/ip" nsap 6       # bind stream to TCP protocol id
ip_udp = "/dev/ip" nsap 17      # bind stream to UDP protocol id

# open ARP for ethernet use
arp     = "/dev/arp" lsap 0x800
arp2     = "/dev/arp" lsap 0x801

# open link level devices
loop    = "/dev/lo" lsap 0x800   # stream for loopback driver
wd_arp = "/dev/wd" lsap 0x806   # stream for ARP messages
wd_ip  = "/dev/wd" lsap 0x800   # stream for IP messages
wd2_arp = "/dev/wd2" lsap 0x806 # stream for ARP messages
wd2_ip  = "/dev/wd2" lsap 0x800 # stream for IP messages

# put it together
link ip_tcp under tcp
link ip_udp under udp

# names must not assume a unit number so use as part of name
link loop under ip_tcp name "lo0"
link arp under ip_tcp name "wd0"
link arp2under ip_tcp name "wd2"
link wd_arp under arp type 0x101
link wd_ip under arp type 0x1
link wd2_arp under arp2type 0x101
link wd2_ip under arp2type 0x1

# initial configuration of interfaces
ifconfig "wd0" hugo.ism.isc.com netmask 255.255.255.0 broadcast
      128.212.16.0 up
ifconfig "wd2" hugo.sos.ism.isc.com netmask 255.255.255.0
      broadcast 128.212.21.0 up
ifconfig "lo0" localhost up
```

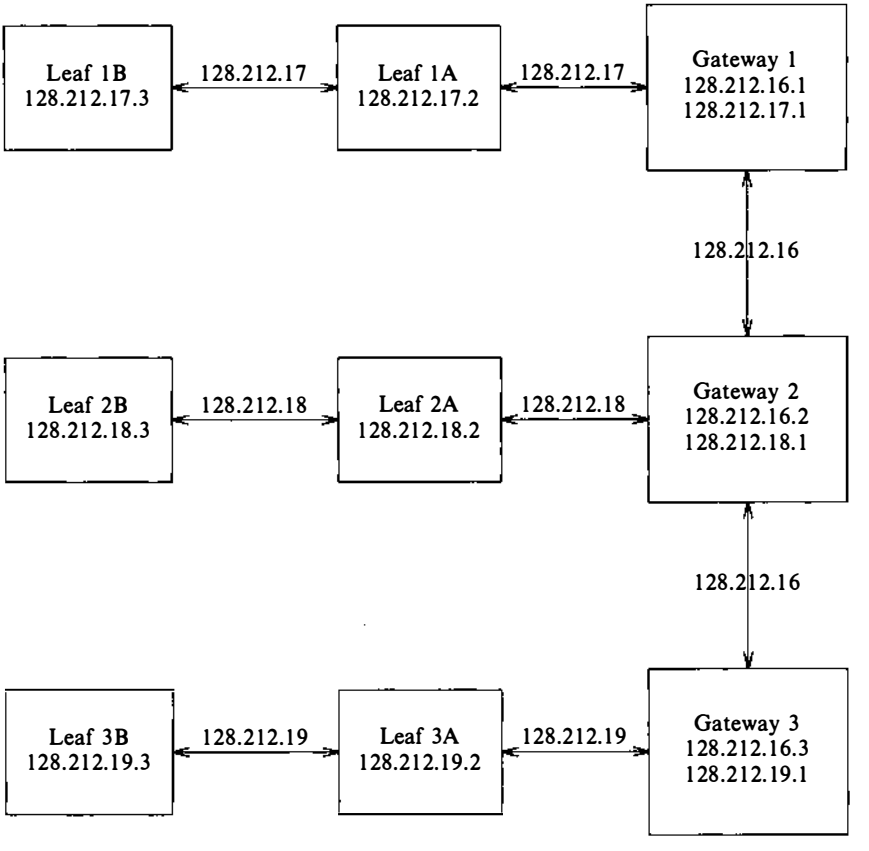
2. ANOTHER `netd.cf` FILE

```
#       @(#)netd.cf     1.4 - 88/11/15
#
# This is a configuration file for TCP/IP using
# the wd ethernet interface and the el interface
# configured to act as a gateway.

# open control streams for transport protocols
tcp = "/dev/tcp"
udp = "/dev/udp"

# open an IP stream for each transport protocol
ip_tcp = "/dev/ip" nsap 6       # bind stream to TCP protocol id
ip_udp = "/dev/ip" nsap 17      # bind stream to UDP protocol id

# open ARP for ethernet use
arp    = "/dev/arp" lsap 0x800
arp2   = "/dev/arp" lsap 0x801

# open link level devices
wd_arp = "/dev/wd" lsap 0x806   # stream for ARP messages
wd_ip  = "/dev/wd" lsap 0x800   # stream for IP messages
el_arp = "/dev/el" lsap 0x806   # stream for ARP messages
el_ip  = "/dev/el" lsap 0x800   # stream for IP messages
loop   = "/dev/lo" lsap 0x800   # stream for loopback driver

# put it together
link ip_tcp under tcp
link ip_udp under udp

# names must not assume a unit number so use as part of name
link loop under ip_tcp name "lo0"
link arp under ip_tcp name "wd0"
link arp2 under ip_tcp name "el0"
link wd_arp under arp type 0x101
link wd_ip under arp type 0x1
link el_arp under arp2 type 0x101
link el_ip under arp2 type 0x1

# initial configuration of interfaces
ifconfig "wd0" mchale netmask 255.255.255.0 broadcast
      128.212.16.0 up
ifconfig "el0" 128.212.19.1 netmask 255.255.255.0 broadcast
      128.212.19.0 up
ifconfig "lo0" localhost up
```

3. AN EXAMPLE space.c FILE

```
#ident "@(#)space.c     1.6.1.2 - 88/12/07"

/*
 * This is /etc/conf/pack.d/wd/space.c.  It is configured
 * to run two "wd" boards.
 */

/*
 * Configuration file for WD8003S (Starlan) / WD8003E (Ethernet).
 * All user configurable options are here (automatically set on PS-2).
 */

#include "sys/types.h"
#include "sys/stream.h"
#include "sys/wd.h"

#define WDDEBUG    0            /* trace transmit attempts */

#include "config.h"

#define NWDDEV     32           /* Number of WD 8003 (sub) devices */
#define NWD        2            /* Number of WD 8003 boards */
#define NSTR       (NWDDEV/NWD) /* Number of streams/board */

struct wddev wddevs[NSTR*NWD];
struct wdstat wdstats[NWD];
int    wd_boardcnt = NWD;

int    wd_debug = WDDEBUG;      /* can be enabled dynamically */

struct wdparam wdparams[NWD] = {
    {
        0,                      /* board index */
        3,                      /* interrupt level */
        0x280,                  /* I/O port for device */
        (caddr_t)0xd0000,       /* address of board's memory */
        8192,                   /* memory size */
        0,                      /* pointer to mapped memory */
        0,                      /* board type */
        0,                      /* board present flag */
        0,                      /* board status */
        0,                      /* number of streams open */
        35,                     /* major device number */
        NSTR                    /* number of minor devices allowed */
    },
    {
        1,                      /* board index */
        7,                      /* interrupt level */
        0x2a0,                  /* I/O port for device */
        (caddr_t)0xd2000,       /* address of board's memory */
        8192,                   /* memory size */
        0,                      /* pointer to mapped memory */
        0,                      /* board type */
        0,                      /* board present flag */
        0,                      /* board status */
        0,                      /* number of streams open */
        50,                     /* major device number */
        NSTR                    /* number of minor devices allowed */
    }
};
```

4. THE NETWORK SCHEME



INTERACTIVE TCP/IP
Programmer's Supplement

CONTENTS

INTERACTIVE TCP/IP
Programmer's Supplement

1. INTRODUCTION

This document presents supplemental information about how to program two interfaces of the Transmission Control Protocol and Internet Protocol (TCP/IP) product under the INTERACTIVE UNIX* Operating System, enhanced by INTERACTIVE Systems Corporation. The Internet Protocol (IP) is not described here. Refer to *ip*(7) for more information about the Internet Protocol. This supplement specifically describes the INTERACTIVE TCP/IP product, but it is generally applicable to the NP622 product as well.

1.1 The TCP/IP Interfaces

TCP/IP supports two programming interfaces: the USL Transport Layer Interface (also referred to as TLI or the Transport Interface) and the Berkeley Software Distribution (BSD) socket interface. The TLI is a library of routines and state transition rules that provides the basic data transfer service required by higher layer protocols, as supported by TCP/IP. For more information about TLI, refer to the *Network Programmer's Guide*. The socket interface is a program interface mechanism that provides endpoints for communication between processes. For more information about the socket interface, refer to Section 3i of the *UNIX Programmer's Reference Manual* [1].

Both of these interfaces provide similar access to the data transfer services of the underlying network protocols but differ in programming techniques. Because of their equivalence, the use of one interface rather than the other is largely a matter of preference, based on the user's programming experience or on the requirements of the environment in which an application is to run. Further comparison is not made here, but more details about each interface may be obtained from the reference manuals listed in the next section.

1.2 Prerequisites

For the Transport Interface, users must have a working knowledge of the networking support functions described in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*, Section 3N, including all functions except `t_rcvrel`. In addition, users

should be well acquainted with the corresponding version of the *Network Programmer's Guide*, which this guide is designed to supplement. In the *Network Programmer's Guide*, examples are based on a client-server paradigm; familiarity with this paradigm is assumed. For more information about this and other reference documentation, see the "Documentation Roadmap" in the *INTERACTIVE UNIX Operating System Guide* and section 4 of this document.

The TCP/IP socket interface is similar to the Release 4.3BSD socket mechanism, as documented in *socket*(2) of the 4.3BSD *UNIX Programmer's Reference Manual*. Section 3I provides documentation for the INTERACTIVE TCP/IP version of the 4.3BSD socket mechanism. Readers should have working knowledge of this material.

1.3 Overview of This Document

This document has four main sections:

1. INTRODUCTION
   This section introduces the interfaces, specifies the prerequisite knowledge necessary, lists reference documentation, and describes the organization of this document.
2. THE TRANSPORT LAYER INTERFACE
   This section presents supplemental details about the Transport Layer Interface and provides examples of how to use this interface. It also provides an explanation of the two types of services available: connection mode and connectionless mode.
3. THE SOCKET INTERFACE
   This section describes how to program the socket interface. It provides instructions for using two types of sockets, SOCK_STREAM and SOCK_DGRAM, that correspond to the connection-mode and connectionless-mode services of the Transport Interface.
4. REFERENCES
   This section provides information on related documentation.

2. THE TRANSPORT LAYER INTERFACE

The TCP/IP Transport Layer Interface supports two types of services: connection mode and connectionless mode. Connection-mode service reliably transfers data in proper sequence through a single, two-way data stream. This type of service is provided by the Transmission Control Protocol (TCP) module of TCP/IP, which runs on top of the Internet Protocol (IP) layer. Connectionless mode provides a message-passing mechanism in which a transport endpoint can independently address each message sent. In this mode, delivery of messages may not be reliable. In the TCP/IP product, the User Datagram Protocol (UDP) and IP modules are such unreliable services. UDP also runs on top of the IP layer.

2.1 Connection-Mode Service

The connection-mode service provided by the TCP module has four phases as described in Chapter 3 of the *Network Programmer's Guide*: local management, connection establishment, data transfer, and connection release. The following sections present supplemental details about these phases.

2.1.1 Local Management

This section supplements "Local Management" in Chapter 3 of the *Network Programmer's Guide*.

2.1.1.1 `t_open`. `t_open` establishes a TCP transport endpoint – a local channel to the transport provider. For example, a transport provider with connection-mode service could be opened by specifying `/dev/tcp` as the path to the transport provider (the first argument to `t_open`) and `O_RDWR` as the open flag (the second argument to `t_open`).

If a `t_info` structure is specified as the third argument of the `t_open` call, `t_open` sets the fields of the `t_info` structure with the following values, which characterize this protocol:

Field	*Value*	*Comment*
addr	8	
options	−1	Although the option size has no set limit, the number of options possible defines a practical limit.
tsdu	0	
etsdu	0	
connect	-1	
discon	-1	
servtype	T_COTS	

Otherwise, if these values are not needed by the user, `info` may be set to `NULL` so no values are returned. Refer to Chapter 3 of the *Network Programmer's Guide* for more information about these fields.

2.1.1.2 `t_bind`. `t_bind` is called to bind a transport provider stream (address) to a transport endpoint. The TCP provider uses the following structure in the `addr.buf` field of the *req* parameter:

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

This structure is defined in `<netinet/in.h>`. Only the first eight bytes of the structure are relevant to TCP. The `sin_family` field must be set to the value `AF_INET` as defined in `<sys/socket.h>`. The `sin_port` and `sin_addr` fields must be in network byte order, which on the Intel* 80386* microprocessor is the exact reverse of native byte order. (For information about library routines that manipulate byte order, see section 3.1.1.1 of this guide.) The last member of the structure, `sin_zero`, is a filler that is not required by the Transport Interface.

If the programmer specifies the *req* parameter as `NULL` or the port and address values as zero, the TCP provider assigns an address to the transport endpoint. TCP assigns the port or the address value if either of these is unspecified.

If a `ret` field is provided, the TCP provider returns the address bound to the transport endpoint. The `addr` field of `ret` will

normally return a zero value unless a nonzero address value was provided in the `req` parameter. The reason for the zero value is that TCP defers choosing an endpoint address until a connection is established, at which point the address of the interface the connection will use becomes the bound address. If there was only one possible address, this would not be necessary, but many TCP/IP implementations, including INTERACTIVE TCP/IP, have at least two: the LAN interface and the loopback interface.

Some restrictions affect the port number to which an endpoint can bind; for example, only a privileged process can bind to a port number less than 1024. In general, only a server process must bind to a particular port number. Client processes and other uses commonly allow the system to assign the port.

2.1.1.3 `t_optmgmt`. Usually, local management of the TCP transport endpoint does not require setting any protocol options; however, the `t_optmgmt` function is available to change some of the options and operating parameters that TCP and IP use. Because these options are specific both to the protocols and to their implementations, setting options reduces portability.

For the INTERACTIVE TCP/IP product, the protocol options are classified in these categories:

- Socket: program-interface related
- TCP: TCP-specific
- UPD: UDP-specific
- IP: IP-related

The option values used by TCP are identical to those used by the socket interface. Socket options are defined in `<sys/socket.h>`. The TCP, UDP, and IP options are found in the files `<sys/tcp.h>`, `<sys/udp.h>`, and `<sys/ip.h>`, respectively. For information about these options, see *tcp*(7), *udp*(7), and *ip*(7), which are included in this guide.

Protocol options are specified as a sequence of structures in the `opt` field of the *req* parameter in the `t_optmgmt` call:

```
struct inetopt {
   short      len;          /* length of option */
                            /* (including lead-in) */
   ushort     name;         /* option identifier */
   ushort     level;        /* socket TCP, UDP, or IP, etc. */
   short      fill;         /* filler to align word */
   union inetoptval {
      short       sval;     /* shortint value */
      int         ival;     /* int value */
      long        lval;     /* long int value */
      caddr_t    *cval;     /* byte string value */
      unchar      vval[1]; /* arbitrary length data */
   }     value;
};
```

Boolean values are represented by a nonzero `ival` for true and by a zero `ival` for false. Variable-length data is an arbitrary number of bytes referenced by the `vval` field. The length of the data in the `vval` field is determined by the `len` field of the structure minus the size of the structure, excluding the size of the union. All current options use either the `ival` or the `vval` data forms.

For TCP, the most useful options are `SO_SNDBUF` and `SO_RCVBUF`, which change the size of the *send* and *receive* buffers, and the value of the TCP flow-control window. The option `TCP_NODELAY`, a Boolean, can modify the algorithm used to delay sending small packets in an attempt to improve performance.

2.1.1.4 Client Example. This example indicates how a client creates a transport endpoint before connection is established:

```
#include <tiuser.h>
#include <fcntl.h>
#include <stropts.h>
#include <sys/types.h>
#include <sys/bsdtypes.h>
#include <netinet/in.h>
#include <sys/socket.h>


main()
{
   int fd;

   /* open a TCP stream */
   if ((fd = t_open("/dev/tcp", O_RDWR, NULL))<0){
      t_error("t_open");
      return -1;
   }

   /* bind to whatever TCP assigns by default */
   if (t_bind(fd, NULL, NULL)<0){
      t_error("t_bind");
      t_close(fd);
      return -1;
   }
```

Refer to the *Network Programmer's Guide* for more information.

2.1.1.5 Server Example. This example indicates how a server creates a transport endpoint before connection is established:

```
#include <tiuser.h>
#include <fcntl.h>
#include <stropts.h>
#include <sys/types.h>
#include <sys/bsdtypes.h>
#include <netinet/in.h>
#include <sys/socket.h>

struct t_bind *bind;
struct t_call *call;
#define ADDRESS struct sockaddr_in *

main()
{
   int fd, newfd;

   /* open a TCP stream */
   if ((fd = t_open("/dev/tcp", O_RDWR, NULL))<0){
      t_error("t_open");
      /* some type of error recovery */
   }
```

```
/* allocate data structure for a bind to specific address */
/* then fill in the information required                  */
if ((bind = (struct t_bind *)t_alloc(fd, T_BIND, T_ADDR))
== NULL){
   t_error("t_alloc");
   /* some type of error recovery */
}
bind->qlen = 5;
bind->addr.len = ADDRESS_SIZE;
((ADDRESS)bind->addr.buf)->sin_port = LOCAL_PORT;
((ADDRESS)bind->addr.buf)->sin_addr.s_addr = LOCAL_ADDRESS;

/* bind to the specified address */
/* return value of "bind" is address that was assigned */
if (t_bind(fd, bind, bind)<0){
   t_error("t_bind");
   exit(1);
}
```

In this example, `ADDRESS_SIZE`, `LOCAL_PORT`, and `LOCAL_ADDRESS` represent appropriate values, which can be obtained as described in section 3.1.1.1.

2.1.2 Connection Establishment

This section supplements "Connection Establishment" in Chapter 3 of the *Network Programmer's Guide*.

TCP, a connection-oriented protocol, establishes transport endpoints that take one of two forms:

- Active: performs a `t_connect` operation.
- Passive: waits for a connection request through `t_listen` and acts upon the request with `t_accept`.

Active endpoints are typically used in network client applications, and passive endpoints are typically used in network server applications.

2.1.2.1 Client. A network client application uses an active endpoint. As noted in the *Network Programmer's Guide*, `t_connect` requires a *sndcall* argument that is a pointer to a `t_call` structure. For TCP, this `t_call` structure must have, at a minimum, a `T_ADDR` field. (In `t_call`, the `buf` field of the *addr* `netbuf` structure must provide the address of the remote TCP server in the form of a `sockaddr_in` structure.) The `sin_addr` field of the `sockaddr_in` structure must contain the network-byte-order representation of the remote system's internet address. The `sin_port` field of the structure must contain the network-byte-order representation of the remote server's

well-known port address. The `sin` family field must contain `AF_INET`.

If the connection request is rejected or the remote system is unreachable, `t_connect` will fail and set `t_errno` to `TLOOK`. In this case, the `t_look` routine can be used to determine the event causing the failure. The `T_DISCONNECT` event indicates connection failure. To determine the exact cause of failure, the programmer can use the `t_rcvdis` function and examine the `reason` field. Possible reasons for failure include:

ETIMEDOUT	Connection not established before the time-out limit.
ECONNREFUSED	Connection refused by the remote system (which may not support the service or may have restricted access to the service).
ENETUNREACH	The network specified in the destination address cannot be reached from this system.

Additional events that may set `t_errno` to `TLOOK` are described in the *Network Programmer's Guide*. Exceptions are `T_ORDREL` and `T_UDERR`, which are not supported in INTERACTIVE TCP/IP.

2.1.2.2 Client Connection Example. In the following example, the client requests a connection:

```
/* allocate data structure for connect address */
/* then fill in the necessary information      */
if ((call = (struct t_call *)t_alloc(fd, T_CALL, T_ADDR))
== NULL){
   t_error("t_alloc");
   t_close(fd);
   return -1;
}
call->addr.len = ADDRESS_SIZE;
((ADDRESS)call->addr.buf)->sin_family = AF_INET;
((ADDRESS)call->addr.buf)->sin_port = REMOTE_PORT;
((ADDRESS)call->addr.buf)->sin_addr.s_addr = REMOTE_ADDRESS;

/* attempt the connection */
if (t_connect(fd, call, NULL) < 0){
   t_error("connect");
   t_close(fd);
   return -1;
}
return fd;
}
```

In this example, ADDRESS_SIZE, REMOTE_PORT, and REMOTE_ADDRESS represent appropriate values, which can be obtained as described in section 3.1.1.1. ADDRESS_SIZE is the same size as sockaddr_in, which is described in section 2.1.1.2

2.1.2.3 Server. Typically, a network server application uses a passive endpoint, accepting on a new stream any incoming connect requests received by the t_listen function. After a new stream is opened by the t_accept function, the server can accept another connect request.

The server must allocate a t_call structure including, at a minimum, a T_ADDR field. The t_listen function normally waits for an event and returns with success on receipt of a connect request from another endpoint. The t_listen function also returns the t_call structure which specifies the address of the requesting system. The sockaddr_in structure contained in the address buffer of the t_call structure will provide both the port and the internet addresses of the remote system. When a connect request is received, the application either accepts the connection with a t_accept or rejects it with a t_snddis call.

2.1.2.4 Server Connection Example. In the following example, a server calls t_listen to listen for an incoming connection request, and t_accept to accept a request for connection:

```
/* allocate data structure for the accept address */
if ((call = (struct t_call *)t_alloc(fd, T_CALL, T_ADDR)) == NULL){
   t_error("t_alloc");
   /* some type of error recovery */
}

/* listen for a connection */
/* "call" will have the requesting address */
if (t_listen(fd, call)<0){
   t_error("listen");
   /* some type of error recovery */
}

/* open new TCP stream to associate the connection with */
if ((newfd = t_open("/dev/tcp", O_RDWR, NULL))<0){
   t_error("t_open");
   exit(1);
}

/* bind it to any address */
if (t_bind(newfd, NULL, NULL)<0){
   t_error("t_bind");
   t_close(newfd);
   /* probably a fatal error */
   exit(1);
}
```

```
/* accept the requested connection on the new stream */
/* descriptor if possible */
if (t_accept(fd, newfd, call) < 0){
   t_error("accept");
   if (t_errno == TLOOK){
      /* there is some possibility of a disconnect */
      /* request to this address */
      if (t_rcvdis(fd, NULL) < 0){
         t_error("rcvdis");
         exit(1);  /* fatal error */
      }
   }
   t_close(newfd);
    newfd = -1;
}

/* newfd is either an error indication or the accepted connection */
```

A connection can be accepted on the same transport endpoint that receives it. If this occurs, the stream so opened must disconnect before the endpoint can accept additional connection requests.

2.1.3 Data Transfer

This section supplements "Data Transfer" in Chapter 3 of the *Network Programmer's Guide.*

After a connection is established, data can be transferred by using the `t_snd` and `t_rcv` calls, which function as the `read` and `write` system calls, respectively. This example shows how normal data is sent:

```
if (t_snd(fd, buff, len, 0)<0)
   t_error("t_snd");
```

The TCP transport provider does not support transfer of expedited data, which has priority over other data. Instead, TCP supports *urgent data*. Urgent data is sent as part of the data stream, along with any other data queued but not yet sent. Urgent data has an indication of its end but not of its quantity (unlike expedited data). Accepted usage assumes one byte of data. To transfer urgent data, the programmer should use the `T_EXPEDITED` flag to set the TCP `URGENT` pointer. If multiple `URGENT` pointers are sent close together, they can be lost or overwritten by a receiving TCP.

TCP supports data transfer in byte stream mode, but not in Transport Service Data Units (TSDUs). Consequently, the `T_MORE` flag is ignored on send and is always zero on receive.

2.1.4 Connection Release

This section supplements "Connection Release" in Chapter 3 of the *Network Programmer's Guide*.

A connection can be terminated by either user at any point during data transfer. Currently, TCP provides only an abortive disconnect, initiated by the `t_snddis` call. Data may be sent with the disconnect request, but its delivery is not guaranteed.

☛ Note that the INTERACTIVE TCP/IP product will guarantee the delivery of data accompanying the disconnect request when connected to a host running the INTERACTIVE TCP/IP product. When connected to a host with a different TCP/IP implementation, however, the delivery of data accompanying a disconnect request may not be supported. It is safe to assume, in general, that the delivery of data sent with the disconnect request is not guaranteed.

If the remote endpoint requests the disconnection, the local endpoint receives an error that causes `TLOOK` to notify the application of the event. The `TLOOK` is a `T_DISCONNECT` event. `t_rcvdis` can be used to return the disconnect request and the associated reason. Commonly, the reason is zero, which indicates a normal disconnect.

2.2 Connectionless-Mode Service

Connectionless-mode service is provided by the User Datagram Protocol (UDP) module of the TCP/IP product. This mode has two phases: local management and data transfer.

2.2.1 Local Management

This section supplements "Local Management" in Chapter 4 of the *Network Programmer's Guide*.

2.2.1.1 `t_open`. `t_open` is called to create a UDP transport endpoint. Specify the file `/dev/udp` as the STREAMS clone device node and `O_RDWR` as the open flag.

If the third parameter of `t_open`, the `info` field, is supplied, the `t_info` structure has the following characteristic values:

Field	*Value*	*Comment*
addr	8	
options	-1	Although no limit is set on the option size, the number of options possible defines a practical limit.
tsdu	0	
etsdu	0	
connect	-1	
discon	-1	
servtype	T_CLTS	

2.2.1.2 t_bind. The transport endpoint must be bound to an address using the same conventions as for TCP, described in section 2.1.1.2.

2.2.2 Data Transfer

This section supplements "Data Transfer" in Chapter 4 of the *Network Programmer's Guide.*

UDP data transfer uses the t_sndudata and the t_rcvudata calls. A T_UNITDATA structure must be allocated. Each datagram sent must specify the destination address of the data and reference the data. Operational values may be specified in the options field.

One important option is the Boolean SO_BROADCAST, which is required to send a message to the broadcast address. IP options may also be used with UDP datagrams. For more information about protocol options, see section 2.1.1.3 and *udp*(7).

2.3 Advanced Topics

The Transport Interface has additional features that are useful in some applications. Of particular interest are those that support asynchronous processing. All of the examples in Chapter 6 of the *Network Programmer's Guide* apply to the TCP/IP product. In addition, this section describes how to prevent blocking for asynchronous processing by using the the STREAMS poll system call.

The `poll` system call can be used to determine which stream file descriptors have data available for receiving and which are not blocked from sending. `poll`, a relatively easy-to-use asynchronous-event interface, allows the programmer to multiplex input/output over a set of stream file descriptors without blocking on any stream. Refer to the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

`poll` takes an array whose elements are `pollfd` structures of the following form:

```
struct pollfd {
        int     fd;       /* file descriptor */
        short   events;   /* events of interest on fd */
        short   revents;  /* events that occurred on fd */
};
```

For a specified file descriptor, `fd`, the above structure is used to define the events of interest.

Two strategies enable an application to perform tasks rather than to remain in the `poll` call. One is to specify the `poll` time-out value as zero, which immediately returns the system call instead of allowing it to wait for an event that might otherwise block. Alternatively, the programmer can use the `I_SETSIG ioctl` command with a signal handler set to catch `SIGPOLL` signals. The `SIGPOLL` signal is sent only when a condition specified in the `I_SETSIG` command is satisfied. Usually, this condition is a message arriving at the stream head.

In addition to using `poll`, the programmer can prevent blocking by using the `O_NDELAY` flag on the `t_open` call when creating a file descriptor, so that the system calls will return errors of `TNODATA` when a receive would otherwise block, or `TFLOW` when a send would otherwise block.

3. THE SOCKET INTERFACE

3.1 Introduction

A *socket* is an abstraction of a communications endpoint that functions as a program interface. The TCP/IP socket interface is identical to the Release 4.3BSD socket mechanism, except as noted in this guide. For information about the 4.3BSD-compatible interface and the *–linet* library, refer to Section 3I of the manual entries included in this guide.

The TCP/IP socket interface requires these `include` files:

File	*Description*
`<sys/socket.h>`	Socket-addressing information
`<netinet/in.h>`	Socket-addressing information
`<sys/bsdtypes.h>`	4.3BSD data types used to define various data structures
`<net/errno.h>`	Network-specific errno values

The first two files in this list are familiar to programmers experienced with the 4.3BSD socket. The other files listed are additional files required for the present implementation of TCP/IP.

A socket is created with the `socket` call. The socket descriptor is created in a specified communications domain conforming to a particular protocol model. This TCP/IP implementation supports the `AF_INET` (internet protocol family) domain with the protocol models `SOCK_STREAM` and `SOCK_DGRAM`.

`SOCK_STREAM` is a reliable, connection-based, byte stream model that does not preserve any records of `write` boundaries. In contrast, `SOCK_DGRAM` is an unreliable datagram, or message, model. In function and use, `SOCK_STREAM` corresponds to the TCP interface, and `SOCK_DGRAM`, to the UDP interface.

A third model, `SOCK_RAW`, provides direct access to the IP layer. Because of the infrequency of its use and its similarity to `SOCK_DGRAM`, `SOCK_RAW` is not documented here.

3.1.1 `SOCK_STREAM`*/TCP Socket Use*

`SOCK_STREAM` sockets can exist in two forms: active, in which the socket actively establishes connections, and passive, in which the socket passively waits for an incoming connection request. In both

forms, a connection must be established before data can be transferred.

3.1.1.1 Creating a TCP Socket. A TCP socket is created in this way:

```
s = socket(AF_INET, SOCK_STREAM, 0);
```

After the socket is created, it should be bound to an address. Addresses are defined in the `sockaddr_in` structure, documented in <`netinet/in.h`>, which has this definition:

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

struct in_addr {
        unsigned long s_addr;
};
```

In this structure, the `sin_family` field is always `AF_INET`. The `sin_addr` field is an internet address in network byte order. The `sin_port` field is a TCP port address in network byte order. The value of the `sin_zero` field is ignored.

☛ Note that the `sockaddr_in` structure is used to pass socket address information to the socket interface routines. The programmer who is not familiar with these routines, which are described in Section 3I of the manual entries, will notice that another structure, the `sockaddr` structure, is expected. Although these are two separately defined structures, the data they contain are the same.

Network byte order specifies that byte zero is the most significant in a word. On the Intel 80386 microprocessor, network byte order is the exact reverse of native byte order. Although the examples in this guide use a symbolic constant to represent the host and port addresses (assumed in network byte order), these values are usually determined dynamically.

The *–linet* library provides several functions for manipulating byte order, as well as for determining host and port addresses from their symbolic names. For more information about these functions, consult the following manual entries included in this document:

Function	*Manual Entry*	*Description*
ntohs	*byteorder*(3I)	Convert short int from network to host order.
ntohl	*byteorder*(3I)	Convert long int from network to host order.
htons	*byteorder*(3I)	Convert short int from host to network order.
htonl	*byteorder*(3I)	Convert long int from host to network order.
gethostbyname	*gethostent*(3I)	Get host address by name.
getservbyname	*getservent*(3I)	Get service (port) by name.

Section 3I of the manual entries included in this guide describes additional utility functions of interest.

To bind an address to the socket, a `sockaddr_in` structure is set up with appropriate values, and a call is made to `bind`:

```
struct sockaddr_in myaddr;
   .
   .
   .
myaddr.sin_family = AF_INET;
myaddr.sin_port = LOCAL_PORT;
myaddr.sin_addr.s_addr = 0;
if (bind(s, &myaddr, sizeof myaddr)<0){
    perror("bind");
    exit(1);
}
```

In this example, `s` is a descriptor returned by a *socket*(3I) call and `LOCAL_PORT` represents an appropriate value. If the value of `LOCAL_PORT` is zero, `bind` will select an unused port with a value of 1024 or greater. Typically, only a server must bind to a specific port address. If either the port or internet address is nonzero, it must be valid for the system on which the bind is done. Refer to section 2.1.1.2 for more information.

3.1.1.2 Connection Establishment. The client uses an active socket to establish a connection through the `connect` call. The remote address must be specified fully (that is, the *port* and *addr* values must be nonzero). The following example illustrates a client establishing connection:

```
struct sockaddr_in remote;
   .
   .
   .
remote.sin_family = AF_INET;
remote.sin_port = REMOTE_PORT;
remote.sin_addr.s_addr = REMOTE_ADDRESS;
if (connect(s, &remote, sizeof(remote))<0){
   perror("connect");
   exit(1);
}
```

In this example, `s` is a descriptor returned by a *socket*(3I) call and `REMOTE_PORT` and `REMOTE_ADDRESS` represent appropriate values, which can be obtained by making calls such as the `gethostbyname` and `getservbyname` calls, as described in section 3.1.1.1. TLI does not provide a standard mechanism for determining these values.

The server uses a passive socket to listen for a client's request for a connection. A passive socket must wait for a connection request from a remote system. To do this, the socket first enters the passive mode with a `listen` call, then waits in an `accept` call. When a connect request arrives, the `accept` call accepts the connection and returns a new socket descriptor that is connected to the remote system. `accept` also returns the requestor's address. The original socket descriptor can then listen for a new connection request.

This example demonstrates the server listening for a request and then waiting in the `accept` call:

```
struct sockaddr_in remote;
int addrlen;
int newsocket;
   .
   .
   .
listen(s, 5);
remote.sin_family = AF_INET;
if ((newsocket = accept(s, &remote, &addrlen))<0){
   perror("accept");
   exit(1);
}
```

3.1.1.3 Data Transfer. After a connection is established, the programmer simply uses the `read` and `write` system calls to exchange data. TCP does not preserve record boundaries, so it is unwise to assume, for example, that when an application does 512-byte `writes`, the remote system receives data 512 bytes at a time. The remote system may receive the data in smaller or larger quantities, depending upon timing and various network

parameters. (Note that the size of the buffer used in the `sendto` call is completely arbitrary.)

3.1.2 `SOCK_DGRAM`*/UDP Socket Use*

`SOCK_DGRAM` sockets differ from `SOCK_STREAM` sockets in that they lack active and passive forms. In general, `SOCK_DGRAM` sockets do not establish connections; therefore, each outgoing message must have an associated destination address provided, and each incoming message, an associated source address. The `connect` call can be used to associate a destination address with a `SOCK_DGRAM` socket permanently, relieving the application of the need to provide an address with each message. Permanent association of an address enables use of the `write` call on a `SOCK_DGRAM` socket.

3.1.2.1 Creating a UDP Socket. A UDP socket is created in this way:

```
s = socket(AF_INET, SOCK_DGRAM, 0);
```

After the socket is created, it should be bound to an address. Addresses are specified in a struct `sockaddr_in`, as described for TCP in section 2.1.1.2.

3.1.2.2 Connection Establishment and Data Transfer. The UDP does not establish connections. Each datagram sent has an explicit destination address associated with it, and each datagram received has a source address associated with it.

To send a datagram, the programmer commonly uses the `sendto` call:

```
struct sockaddr_in remote;
char buffer[512];
  .
  .
  .
remote.sin_family = AF_INET;
remote.sin_port = REMOTE_PORT;
remote.sin_addr.s_addr = REMOTE_ADDRESS;
cc = sendto(s, buffer, sizeof(buffer), 0, &remote, sizeof(remote));
```

In this example, `REMOTE_PORT` and `REMOTE_ADDRESS` represent appropriate values, which can be obtained as described in section 3.1.1.1.

Typically, a datagram is received by using the `recvfrom` call:

```
struct sockaddr_in remote;
char buffer[512];
int addrlen;
  .
  .
  .
addrlen = sizeof(remote);
cc = recvfrom(s, buffer, sizeof(buffer), 0, &remote, &addrlen);
```

In this example, *remote* will contain the source address of the datagram on return from `rcvfrom`. Note that the address of the *addrlen* variable is passed to `recvfrom`. The actual length of the address is returned; the original value is the size of the buffer that stores the address. The value `cc` returned by `recvfrom` is the amount of data received.

3.2 Advanced Topics

The socket interface provides additional features for the experienced programmer. Some of these are described here, and additional details are found in the relevant manual entries.

3.2.1 Options

In the socket model, options can be set on a socket descriptor. An option may affect the socket interface or a protocol below the socket interface. All options are set by the `setsockopt` call, which has this form:

```
int buflen;
  .
  .
  .
buflen = 8192;
setsockopt(s, SOL_SOCKET, SO_SNDBUF, &buflen, sizeof buflen);
```

The example adjusts the buffersize allocated for output buffers to 8192 bytes (the default is 4096). A possible use for this is in high-volume connections.

For TCP, the `TCP_NODELAY` option may be useful, especially in applications that send data but get no echo. Refer to *tcp*(7) for more information.

Current option values are obtained with the `getsockopt` call. The option values used by the socket interface are identical to those used by TLI. Socket options are defined in `<sys/socket.h>`. For more information on the `getsockopt` and `setsockopt` calls and for a list of the socket level options, refer to *getsockopt*(3I).

3.2.2 *Using* `select`

The `select` call is used to determine whether a socket can be read or written without blocking. `select` takes bitmasks that specify the file descriptor(s) of interest. Three bitmasks can be used to check input, output, and urgent conditions. A time-out value is also specified. Normally, `select` blocks until one of the conditions being checked comes true. Blocking occurs when the time-out pointer is `NULL`. If the time-out pointer is non-null, the call blocks for the time specified in the `timeval` structure. A time-out value of zero causes `select` to return immediately.

In the INTERACTIVE UNIX System, `select` essentially provides the same functions as the `poll` system call but uses a different syntax. `select` has the same restrictions as `poll`, but under the INTERACTIVE UNIX System, `poll` has been extended to support non-STREAMS file descriptors. In particular, `pipes`, `pty`, and `tty` devices are supported.

4. REFERENCES

[1] *UNIX Programmer's Reference Manual*, 4.3 Berkeley Software Distribution Virtual VAX-1 Version. Berkeley: University of California, 1986.

In addition, the following documents are recommended:

Defense Communications Agency. *DDN Protocol Handbook, Vol I: DOD Military Standard Protocols*. "Military Standard Internet Protocol," U.S. Department of Defense MIL-STD-1777. Menlo Park, CA: DDN Network Information Center, 1985.

Defense Communications Agency. *DDN Protocol Handbook, Vol I: DOD Military Standard Protocols*. "Military Standard Transmission Control Protocol," U.S. Department of Defense MIL-STD-1778. Menlo Park, CA: DDN Network Information Center, 1985.

Defense Communications Agency. *DDN Protocol Handbook, Vol II: DARPA Internet Protocols*. "Internet Protocol – DARPA Internet Program Protocol Specification" by J. Postel, RFC 791. Menlo Park, CA: DDN Network Information Center, 1985.

Defense Communications Agency. *DDN Protocol Handbook, Vol II: DARPA Internet Protocols*. "Transmission Control Protocol – DARPA Internet Program Protocol Specification" by J. Postel, RFC 793. Menlo Park, CA: DDN Network Information Center, 1985.

INDEX

Name Server Operations Guide for BIND

CONTENTS

Name Server Operations Guide for BIND

1. INTRODUCTION

The Berkeley Internet Name Domain (BIND) server implements the DARPA Internet name server for the UNIX* System.

A name server is a network service that enables clients to name resources or objects and share this information with other objects in the network. This in effect is a distributed database system for objects in a computer network. BIND is fully integrated into INTERACTIVE UNIX System network programs for use in storing and retrieving host names and addresses. The system administrator can configure the system to use BIND as a replacement for the original host table lookup of information in the network hosts file `/etc/hosts`. The default configuration for the INTERACTIVE UNIX System has the BIND capability built into the released software.

This article is reprinted (with format and editorial changes) from the University of California, Berkeley document entitled *Name Server Operations Guide for BIND*, Release 4.3, by Kevin J. Dunlap, with permission of the University of California Regents pursuant to INTERACTIVE's License Agreement for Berkeley Software, Version 6.1.

2. A SYSTEM WITH A NAME SERVER

BIND is comprised of two parts. The first part is the user interface called the `resolver`, which consists of a group of routines that reside in the C library `/usr/lib/libinet.a`. The second part is the actual server called `named`. This is a daemon that runs in the background and services queries on a given network port. The standard port for UDP and TCP is specified in `etc/services`.

2.1 `resolver` Routines in `libc`

The `resolver` is comprised of a few routines that build query packets and exchange them with the name server. The name server changes the way `gethostbyname`, `gethostbyaddr`, and `sethostent` (see *ghostnamed*(3I)) do their functions. The library routines are built to use the name server specified in `/etc/resolv.conf` (see *resolver*(5)).

2.2 The Name Service

The basic function of the name server is to provide information about network objects by answering queries. The specifications for this name server are defined in the following documents: RFC882, RFC883, RFC973, and RFC974. These documents can be `ftped` from `sri-nic.arpa`. It is also recommended that you read the related manual entries *resolver*(3I) and *resolver*(5), and *named*(8) in the *UNIX System Manager's Manual.*

It is advantageous to use a name server over the host table lookup for host name resolution because this avoids the need for a single centralized clearinghouse for all names. The authority for this information can be delegated to the different organizations on the network responsible for it.

The host table lookup routines require that the master file for the entire network be maintained at a central location by a few people. This works fine for small networks where there are only a few machines and the different organizations responsible for them cooperate. But this does not work well for large networks where machines cross organizational boundaries.

With the name server, the network can be broken into a hierarchy of domains. The name space is organized as a tree according to organizational or administrative boundaries. Each node, called a domain, is given a label, and the name of the domain is the concatenation of all the labels of the domains from the root to the current

domain, listed from right to left and separated by dots. A label need only be unique within its domain. The whole space is partitioned into several areas called zones, each starting at a domain and extending down to the leaf domains or to domains where other zones start. Zones usually represent administrative boundaries.

An example of a host address for a host at the University of California, Berkeley would look as follows:

```
monet.Berkeley.EDU
```

The top-level domain for educational organizations is `EDU`; `Berkeley` is a subdomain of `EDU`, and `monet` is the name of the host.

3. TYPES OF SERVERS

There are three types of servers: master, caching only, and remote.

3.1 Master Servers

A master server for a domain is the authority for that domain. This server maintains all the data corresponding to its domain. Each domain should have at least two master servers: a primary master and some secondary masters to provide backup service if the primary is unavailable or overloaded. A server may be a master for several domains, acting as a primary master for some domains and as a secondary master for others.

3.1.1 Primary

A primary master server is one that loads its data from a file on disk. This server may also delegate authority to other servers in its domain.

3.1.2 Secondary

A secondary master server is one that is delegated authority and receives its data for a domain from a primary master server. At boot time, the secondary server requests all the data for the given zone from the primary master server. This server then periodically checks with the primary server to see if it needs to update its data.

3.2 Caching Only Server

All servers are caching servers. This means that the server caches the information that it receives for use until the data expires. A caching only server is a server that is not authoritative for any domain. This server services queries and asks other servers, who have the authority, for the information needed. All servers keep data in their cache until the data expires, based on a "time to live" (*ttl*) field attached to the data when it is received from another server.

3.3 Remote Server

A remote server is an option given to people who would like to use a name server on their workstation or on a machine that has a limited amount of memory and CPU cycles. With this option you can run all of the networking programs that use the name server without the name server running on the local machine. All of the queries are serviced by a name server that is running on another machine on the network.

4. SETTING UP YOUR OWN DOMAIN

When setting up a domain that is going to be on a public network, the site administrator should contact the organization in charge of the network and request the appropriate domain registration form. An organization that belongs to multiple networks (such as CSNET, DARPA Internet, and BITNET) should register with only one network. The contacts are given in the following subsections.

4.1 DARPA Internet

Sites that are already on the DARPA Internet and need information on setting up a domain should contact:

```
HOSTMASTER@SRI-NIC.ARPA.
```

You may also want to be placed on the BIND mailing list, which is a mail group for people on the DARPA Internet running BIND. The group discusses future design decisions, operational problems, and other related topics. The address to request being placed on this mailing list is:

```
bind-request@ucbarpa.Berkeley.EDU
```

4.2 CSNET

A CSNET member organization that has not registered its domain name should contact the CSNET Coordination and Information Center (CIC) for an application and information about setting up a domain.

An organization that already has a registered domain name should keep the CIC informed about how it would like its mail routed. In general, the CSNET relay will prefer to send mail via CSNET (as opposed to BITNET or the Internet) if possible. For an organization on multiple networks, this may not always be the preferred behavior. The CIC can be reached via electronic mail at:

```
cic@sh.ch.net
```

4.3 BITNET

If you are on the BITNET and need to set up a domain, contact:

```
INFO@BITNIC
```

5. FILES

The name server uses several files to load its database. This section covers the files and their formats needed for `named`.

5.1 Boot File

This is the file that is first read when `named` starts up. It tells the server what type of server it is, which zones it has authority over, and where to get its initial data. The default location for this file is `/etc/named.boot`. However, this can be changed by setting the `BOOTFILE` variable when you compile `named`, or by specifying the location on the command line when `named` is started up.

5.1.1 Domain

The line in the boot file that designates the default domain for the server looks as follows:

```
domain          Berkeley.Edu
```

The name server uses this information when it receives a query for a name without a ".". When it receives one of these queries, it appends the name in the second field to the query name.

5.1.2 Primary Master

The line in the boot file that designates the server as a primary server for a zone looks as follows:

```
primary         Berkeley.Edu    /etc/ucbhosts
```

The first field specifies that the server is a primary one for the zone stated in the second field. The third field is the name of the file from which the data is read.

5.1.3 Secondary Master

The line for a secondary server is similar to the line for a primary, except that the word `secondary` replaces `primary` and the third field contains network addresses:

```
secondary  Berkeley.Edu  128.32.0.10  128.32.0.4
```

The first field specifies that the server is a secondary master server for the zone stated in the second field. The rest of the line lists the network addresses for the name servers that are primary for the zone. The secondary server gets its data across the network from the listed servers. Each server is tried in the order listed until it successfully receives the data from a listed server.

5.1.4 Caching Only Server

You do not need a special line to designate that a server is a caching server. What denotes a caching only server is the absence of authority lines, such as `secondary` or `primary`, in the boot file.

All servers should have a line as follows in the boot file to prime the name server's cache:

```
cache   /etc/named.ca
```

For more information on the cache file see section 5.2, "Cache Initialization."

5.1.5 Remote Server

To set up a host that will use a remote server instead of a local server to answer queries, the file `/etc/resolv.conf` must be created. This file designates the name servers on the network to which queries should be sent. It is not advisable to create this file if you have a local server running. If this file exists, it is read almost every time `gethostbyname` or `gethostbyaddr` (see *ghostnamed*(3I)) is called.

5.2 Cache Initialization

5.2.1 `named.ca`

The name server needs to know which server is the authoritative name server for the network. To do this the name server's cache has to be primed with the address of these higher authorities. The location of this file is specified in the boot file. This file uses the Standard Resource Record Format covered later in this document.

5.3 Domain Data Files

There are three standard files for specifying the data for a domain. These are `named.local`, `hosts`, and `host.rev`. These files use the Standard Resource Record Format covered later in this document.

5.3.1 `named.local`

This file specifies the address for the local loopback interface, better known as *localhost* with the network address `127.0.0.1`. The location of this file is specified in the boot file.

5.3.2 hosts

This file contains all the data about the machines in this zone. The location of this file is specified in the boot file.

5.3.3 hosts.rev

This file specifies the IN-ADDR.ARPA domain. This is a special domain for allowing address to name mapping. As Internet host addresses do not fall within domain boundaries, this special domain was formed to allow inverse mapping. The IN-ADDR.ARPA domain has four labels preceding it. These labels correspond to the four octets of an Internet address. All four octets must be specified even if one is zero. The Internet address 128.32.0.4 is located in the domain 4.0.32.128.IN-ADDR.ARPA. This reversal of the address is awkward to read but allows for the natural grouping of hosts in a network.

5.4 Standard Resource Record Format

The records in the name server data files are called resource records. The Standard Resource Record Format (RR) is specified in RFC882 and RFC973. The following is a general description of these records:

[name] *[ttl]* *class* *record type* *record-specific data*

Resource records have a standard format as shown above. The first field is always the name of the domain record. For some RRs the name may be left blank; in that case it takes on the name of the previous RR. The second field is an optional "time to live" (*ttl*) field. This specifies how long this data will be stored in the database. By leaving this field blank, the default *ttl* is specified in the Start of Authority resource record (see section 5.4.3). The third field is the address class; there are currently two classes: IN for Internet addresses and ANY for all address classes. The fourth field states the type of the resource record. The fields after that are dependent on the type of RR. Case is preserved in names and data fields when loaded into the name server. All comparisons and lookups in the name server database are case insensitive.

The following characters have special meanings:

- A free-standing dot in the *name* field refers to the current domain.

@ A free-standing @ in the *name* field denotes the current origin.

. . Two free-standing dots represent the null domain name of the root when used in the *name* field.

\^*X* Where *X* is any character other than a digit (0-9), quotes that character so that its special meaning does not apply. For example, "\." can be used to place a dot character in a label.

\^*DDD* Where each *D* is a digit, is the octet corresponding to the decimal number described by *DDD*. The resulting octet is assumed to be text and is not checked for special meaning.

( ) Parentheses are used to group data that crosses a line. In effect, line terminations are not recognized within parentheses.

; A semicolon starts a comment; the remainder of the line is ignored.

* An asterisk signifies a wildcard.

Most resource records will have the current origin appended to names if they are not terminated by a ".". This is useful for appending the current domain name to the data, such as machine names, but may cause problems where you do not want this to happen. A good rule of thumb is that if the name is not in the domain for which you are creating the data file, end the name with a ".".

5.4.1 `$INCLUDE`

An include line begins with `$INCLUDE`, starting in column 1, and is followed by a file name. This feature is particularly useful for separating different types of data into multiple files. An example would be:

```
$INCLUDE    /usr/named/data/mailboxes
```

The line would be interpreted as a request to load the file `/usr/named/data/mailboxes`. The `$INCLUDE` command does not cause data to be loaded into a different zone or tree. This is simply a way to allow data for a given zone to be organized in separate files. For example, mailbox data might be kept separately from host data using this mechanism.

5.4.2 $ORIGIN

The origin is a way of changing the origin in a data file. The line starts in column 1 and is followed by a domain origin. This is useful for putting more than one domain in a data file.

5.4.3 SOA – Start of Authority

```
[name]  [ttl]  class  SOA  Origin                  Person in charge
@              IN     SOA  ucbvax.Berkeley.Edu.    kjd.ucbvax.Berkeley.Edu. (
                           1.1                     ; serial
                           3600                    ; refresh
                           300                     ; retry
                           3600000                 ; expire
                           3600 )                  ; minimum
```

The Start of Authority (SOA) record designates the start of a zone. The *name* is the name of the zone. *Origin* is the name of the host on which this data file resides. *Person in charge* is the mailing address for the person responsible for the name server. The `serial` number is the version number of this data file; this number should be incremented whenever a change is made to the data. The name server cannot handle numbers over 9999 after the decimal point. `refresh` indicates how often, in seconds, a secondary name server is to check with the primary name server to see if an update is needed. `retry` indicates how long, in seconds, a secondary server is to retry after a failure to check for a refresh. `expire` is the upper limit, in seconds, that a secondary name server is to use the data before it expires for lack of getting a refresh. `minimum` is the default number of seconds to be used for the *ttl* field on resource records. There should only be one SOA record per zone.

5.4.4 NS – Name Server

```
[name]          [ttl]          class  NS   Name server name
                               IN     NS   ucbarpa.Berkeley.edu
```

The name server record (NS) lists a name server responsible for a given domain. The first *name* field lists the domain that is serviced by the listed name server. There should be one NS record for each primary master server for the domain.

5.4.5 A – Address

```
[name]          [ttl]          class  A    address
ucbarpa                        IN     A    128.32.0.4
                               IN     A    10.0.0.78
```

The address record (A) lists the address for a given machine. The *name* field is the machine name, and the *address* is the network

address. There should be one A record for each address of the machine.

5.4.6 HINFO — Host Information

[name]	*[ttl]*	*class*	*HINFO*	*Hardware*	*OS*
		ANY	HINFO	VAX-11/780	UNIX

The host information record (HINFO) is for host-specific data. This lists the hardware and operating system that are running at the listed host. It should be noted that only a single space separates the hardware info and the operating system info. If you want to include a space in the machine name, you must quote the name. Since host information is not specific to any address class, ANY may be used for the address class. There should be one HINFO record for each host.

5.4.7 WKS — Well-Known Services

[name]	*[ttl]*	*class*	*WKS*	*address*	*protocol*	*list of services*
		IN	WKS	128.32.0.10	UDP	who route timed domain
		IN	WKS	128.32.0.10	TCP	(echo telnet discard sunrpc sftp uucp-path systat daytime netstat qotd nntp link chargen ftp auth time whois mtp pop rje finger smtp supdup hostnames domain nameserver)

The well-known services record (WKS) describes the well known services supported by a particular protocol at a specified address. The list of services and port numbers come from the list of services specified in /etc/services. There should be only one WKS record per protocol per address.

5.4.8 CNAME — Canonical Name

[name]	*[ttl]*	*class*	*CNAME*	*canonical name*
ucbmonet		IN	CNAME	monet

The canonical name record (CNAME) specifies an alias for a canonical name. An alias should be unique, and all other resource records should be associated with the canonical name and not with the alias. Do not create an alias and then use it in other resource records.

5.4.9 PTR – Domain Name Pointer

[name]	[ttl]	class	PTR	real name
7.0		IN	PTR	monet.Berkeley.Edu.

The domain name pointer record (PTR) allows special names to point to some other location in the domain. The above example of a PTR record is used in setting up reverse pointers for the special IN-ADDR.ARPA domain. This line is from the example hosts.rev file. PTR names should be unique to the zone.

5.4.10 MB – Mailbox

[name]	[ttl]	class	MB	machine
miriam		IN	MB	vineyd.DEC.COM.

The mailbox record (MB) lists the machine where a user wants to receive mail. The *name* field is the user's login; the *machine* field denotes the machine to which mail is to be delivered. Mailbox names should be unique to the zone.

5.4.11 MR – Mail Rename Name

[name]	[ttl]	class	MR	corresponding MB
Postmistress		IN	MR	miriam

The mail rename record (MR) can be used to list aliases for a user. The *name* field lists the alias for the name listed in the fourth field, which should have a corresponding MB record.

5.4.12 MINFO – Mailbox Information

[name]	[ttl]	class	MINFO	requests	maintainer
BIND		IN	MINFO	BIND-REQUEST	kjd.Berkeley.Edu.

The mailbox information record (MINFO) creates a mail group for a mailing list. This resource record is usually associated with a mail group (see MG below), but may be used with a mailbox (MB) record. The *name* specifies the name of the mailbox. The *requests* field is where mail such as requests to be added to a mail group should be sent. The *maintainer* is a mailbox that should receive error messages. This is particularly appropriate for mailing lists when errors in members names should be reported to a person other than the sender.

5.4.13 MG – Mail Group Member

[name]	*[ttl]*	*class*	*MG*	*member name*
		IN	MG	Bloom

The mail group record (MG) lists members of a mail group. An example for setting up a mailing list is as follows:

```
Bind            IN      MINFO   Bind-Request kjd.Berkeley.Edu.
                IN      MG      Ralph.Berkeley.Edu.
                IN      MG      Zhou.Berkeley.Edu.
                IN      MG      Painter.Berkeley.Edu.
                IN      MG      Riggle.Berkeley.Edu.
                IN      MG      Terry.pa.Xerox.Com.
```

5.4.14 MX – Mail Exchanger

[name]	*[ttl]*	*class*	*MX*	*preference*	*mailer exchanger*
Munnari.OZ.AU.		IN	MX	0	Seismo.CSS.GOV.
*.IL.		IN	MX	0	RELAY.CS.NET.

The mail exchanger record (MX) is used to specify a machine that knows how to deliver mail to a machine that is not directly connected to the network. In the first line of the example above, Seismo.CSS.GOV. is a mail gateway that knows how to deliver mail to Munnari.OZ.AU., but other machines on the network can not deliver mail directly to Munnari. These two machines may have a private connection or use a different transport medium. The *preference* value is the order that a mailer should follow when there is more then one way to deliver mail to a single machine. See RFC974 for more detailed information.

Wildcard names containing the character "*" may be used for mail routing with MX records. There are likely to be servers on the network that simply state that any mail to a domain is to be routed through a relay. In the second line of the example above, all mail to hosts in the domain IL is routed through RELAY.CS.NET. This is done by creating a wildcard resource record, which states that *.IL has an MX of RELAY.CS.NET.

5.5 Sample Files

The following section contains sample files for the name server. This covers example boot files for the different types of servers and example domain database files.

5.5.1 Boot File

5.5.1.1 Primary Master Server.

```
;
; Boot file for Primary Master Name Server
;

; type      domain                  source file or host

domain      Berkeley.Edu
primary     Berkeley.Edu            /etc/ucbhosts
cache       .                       /etc/named.ca
primary     32.128.in-addr.arpa     /etc/ucbhosts.rev
primary     0.0.127.in-addr.arpa    /etc/named.local
```

5.5.1.2 Secondary Master Server.

```
;
; Boot file for Primary Master Name Server
;

; type    domain                source file or host

domain    Berkeley.Edu
secondary Berkeley.Edu          128.32.0.4 128.32.0.10 128.32.136.22
cache     .                     /etc/named.ca
secondary 32.128.in-addr.arpa   128.32.0.4 128.32.0.10 128.32.136.22
primary   0.0.127.in-addr.arpa /etc/named.local
```

5.5.1.3 Caching Only Server.

```
;
; Boot file for Primary Master Name Server
;

;type       domain                   source file or host

domain      Berkeley.Edu
cache       .                        /etc/named.ca
primary     0.0.127.in-addr.arpa     /etc/named.local
```

5.5.2 Remote Server

5.5.2.1 /etc/resolv.conf.

```
domain                              Berkeley.Edu
nameserver  128.32.0.4
nameserver  128.32.0.10
```

5.5.3 named.ca

```
;
; Initial cache data for root domain servers.
;

                99999999        IN      NS      USC-ISIB.ARPA.
                99999999        IN      NS      BRL-AOS.ARPA.
                99999999        IN      NS      SRI-NIC.ARPA.
; Prep the cache (hotwire the addresses).
SRI-NIC.ARPA.   99999999        IN      A       10.0.0.51
USC-ISIB.ARPA.  99999999        IN      A       10.3.0.52
USC-ISIC.ARPA.  99999999        IN      A       10.0.0.52
BRL-AOS.ARPA.   99999999        IN      A       128.20.1.2
BRL-AOS.ARPA.   99999999        IN      A       192.5.22.8
```

5.5.4 named.local

```
@       IN      SOA     ucbvax.Berkeley.Edu. kjd.ucbvax.Berkeley.Edu. (
                        1       ; serial
                        3600    ; refresh
                        300     ; retry
                        3600000 ; expire
                        3600 )  ; minimum
        IN      NS      ucbvax.Berkeley.Edu.
1       IN      PTR     localhost.
```

5.5.5 Hosts

```
;
;  @(#)ucb-hosts     1.1     (berkeley)     86/02/05
;

@       IN      SOA     ucbvax.Berkeley.Edu. kjd.monet.Berkeley.Edu. (
                        1.1       ; serial
                        3600      ; refresh
                        300       ; retry
                        3600000   ; expire
                        3600 )    ; minimum
                IN      NS      ucbarpa.Berkeley.Edu.
                IN      NS      ucbvax.Berkeley.Edu.
localhost       IN      A       127.1
ucbarpa         IN      A       128.32.4
                IN      A       10.0.0.78
                ANY     HINFO   VAX-11/780 UNIX
arpa            IN      CNAME   ucbarpa
ernie           IN      A       128.32.6
                ANY     HINFO   VAX-11/780 UNIX
ucbernie        IN      CNAME   ernie
monet           IN      A       128.32.7
                IN      A       128.32.130.6
                ANY     HINFO   VAX-11/750 UNIX
ucbmonet        IN      CNAME   monet
ucbvax          IN      A       10.2.0.78
                IN      A       128.32.10
                ANY     HINFO   VAX-11/750 UNIX
```

```
            IN      WKS     128.32.0.10 UDP syslog route timed domain
            IN      WKS     128.32.0.10 TCP ( echo telnet
                            discard sunrpc sftp
                            uucp-path systat daytime
                            netstat qotd nntp
                            link chargen ftp
                            auth time whois mtp
                            pop rje finger smtp
                            supdup hostnames
                            domain
                            nameserver )
vax         IN      CNAME   ucbvax
toybox      IN      A       128.32.131.119
            ANY     HINFO   Pro350 RT11
toybox      IN      MX      0  monet.Berkeley.Edu
miriam      ANY     MB      vineyd.DEC.COM.
postmistress ANY    MR      Miriam
Bind        ANY     MINFO   Bind-Request kjd.Berkeley.Edu.
            ANY     MG      Ralph.Berkeley.Edu.
            ANY     MG      Zhou.Berkeley.Edu.
            ANY     MG      Painter.Berkeley.Edu.
            ANY     MG      Riggle.Berkeley.Edu.
            ANY     MG      Terry.pa.Xerox.Com.
```

5.5.6 host.rev

```
;
; @(#)ucb-hosts.rev    1.1    (Berkeley)    86/02/05
;

@       IN      SOA     ucbvax.Berkeley.Edu. kjd.monet.Berkeley.Edu. (
                        1.1     ; serial
                        3600    ; refresh
                        300     ; retry
                        3600000 ; expire
                        3600 )  ; minimum
        IN      NS      ucbarpa.Berkeley.Edu.
        IN      NS      ucbvax.Berkeley.Edu.
4.0     IN      PTR     ucbarpa.Berkeley.Edu.
6.0     IN      PTR     ernie.Berkeley.Edu.
7.0     IN      PTR     monet.Berkeley.Edu.
10.0    IN      PTR     ucbvax.Berkeley.Edu.
6.130   IN      PTR     monet.Berkeley.Edu.
```

6. DOMAIN MANAGEMENT

This section contains information for starting, controlling, and debugging `named`.

6.1 `/etc/rc.local`

The host name should be set to the full domain style name in `/etc/rc.local` using `hostname` (see *hostname*(1) in the *UNIX User's Manual Reference Guide*). The following entry should be added to `/etc/rc.local` to start up `named` at system boot time:

```
if [ -f /etc/named ]; then
 /etc/named [ options ] & echo -n ' named' > /dev/console
```

This usually directly follows the lines that start `syslogd`.

☛ Do not attempt to run `named` from `inetd`. This will continuously restart the name server and defeat the purpose of having a cache.

6.2 `/etc/named.pid`

When `named` is successfully started up, it writes its process ID into the file `/etc/named.pid`. This is useful to programs that want to send signals to `named`. The name of this file may be changed by defining `PIDFILE` to the new name when compiling `named`.

6.3 `/etc/hosts`

The `gethostbyname( )` library call can detect if `named` is running. If it is determined that `named` is not running, it will look in `/etc/hosts` to resolve an address. This option was added to allow `ifconfig` (see *ifconfig*(8C) in the *UNIX System Manager's Manual*) to configure the machines' local interfaces and to enable a system manager to access the network while the system is in single-user mode. It is advisable to put the local machine's interface addresses and a couple of machine names and addresses in `/etc/hosts` so the system manager can `rcp` files from another machine when the system is in single-user mode. The format of `/etc/hosts` has not changed. See *hosts*(5) for more information. Since the process of reading `/etc/hosts` is slow, it is not advisable to use this option when the system is in multi-user mode.

6.4 Signals

There are several signals that can be sent to the `named` process to have it do tasks without restarting the process.

6.4.1 Reload

SIGHUP causes named to read named.boot and reload the database. All previously cached data is lost. This is useful when you have made a change to a data file and you want named's internal database to reflect the change.

6.4.2 Debugging

When named is running incorrectly, look first in /usr/adm/messages and check for any messages logged by syslog. Next send it a signal to see what is happening.

SIGINT dumps the current database and cache to /usr/tmp/named_dump.db. This should give you an indication as to whether the database was loaded correctly. The name of the dump file may be changed by defining DUMPFILE to the new name when compiling named.

☛ Note that the following two signals only work when named is built with DEBUG defined.

SIGUSR1 turns on debugging. Each following USR1 increments the debug level. The output goes to /usr/tmp/named.run. The name of this debug file may be changed by defining DEBUGFILE to the new name before compiling named.

SIGUSR2 turns off debugging completely. For more detailed debugging, define DEBUG when compiling the resolver routines into /lib/libc.a.

ACKNOWLEDGMENTS

Many thanks to the users at U.C. Berkeley for falling into many of the holes involved with integrating BIND into the system so that others would be spared the trauma. I would also like to extend gratitude to Jim McGinness and Digital Equipment Corporation for permitting me to spend most of my time on this project.

Ralph Campbell, Doug Kingston, Craig Partridge, Smoot Carl-Mitchell, Mike Muuss, and everyone else on the DARPA Internet who has contributed to the development of BIND. To the members of the original BIND project, Douglas Terry, Mark Painter, David Riggle, and Songnian Zhou.

Anne Hughes, Jim Bloom and Kirk McKusick, and the many others who have reviewed this paper giving considerable advice.

This work was sponsored by the Defense Advanced Research Projects Agency (DoD), Arpa Order No. 4871 monitored by the Naval Electronics Systems Command under contract No. N00039-84-C-0089. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Research Projects Agency, of the U.S. Government, or of Digital Equipment Corporation.

REFERENCES

[Birrell] Birrell, A. D., Levin, R., Needham, R. M., and Schroeder, M.D., "Grapevine: An Exercise in Distributed Computing." In *Comm. A.C.M. 25*, 4:260-274 April 1982.

[RFC819] Su, Z. Postel, J., "The Domain Naming Convention for Internet User Applications." *Internet Request For Comment 819* Network Information Center, SRI International, Menlo Park, California. August 1982.

[RFC882] Mockapetris, P., "Domain Names – Concept and Facilities." *Internet Request For Comment 882* Network Information Center, SRI International, Menlo Park, California. November 1983.

[RFC883] Mockapetris, P., "Domain Names – Implementation and Specification." *Internet Request For Comment 883* Network Information Center, SRI International, Menlo Park, California. November 1983.

[RFC973] Mockapetris, P., "Domain System Changes and Observations." *Internet Request For Comment 973* Network Information Center, SRI International, Menlo Park, California. February 1986.

[RFC974] Partridge, C., "Mail Routing and the Domain System." *Internet Request For Comment 974* Network Information Center, SRI International, Menlo Park, California. February 1986.

[Terry] Terry, D. B., Painter, M., Riggle, D. W., and Zhou, S., *The Berkeley Internet Name Domain Server.* Proceedings USENIX Summer Conference, Salt Lake City, Utah. June 1984, pages 23-31.

[Zhou] Zhou, S., *The Design and Implementation of the Berkeley Internet Name Domain BIND Servers.* UCB/CSD 84/177. University of California, Berkeley, Computer Science Division. May 1984.

Domain Administrator's Operations Guide

CONTENTS

Domain Administrator's Operations Guide

1. INTRODUCTION

This document provides guidelines for domain administrators in operating a domain server and maintaining their portion of the hierarchical database. It assumes you are familiar with the domain system.

A domain server requires a few files to get started. It will normally have some number of boot/startup files (also known as the "safety belt" files). One section will contain a list of possible root servers that the server will use to find the up-to-date list of root servers. Another section will list the zone files to be loaded into the server for your local domain information. A zone file typically contains all the data for a particular domain. This document describes the data formats that can be used in zone files and suggested parameters to use for certain fields. If you are attempting to do anything advanced or tricky, consult the appropriate domain RFCs for more details.

Note that each implementation of domain software may require different files. Zone files are standardized, but some servers may require other startup files. See the appropriate documentation that comes with your software. See section 8, "BIND EXAMPLES," for some specific examples.

This article is reprinted (with format and editorial changes) with permission from the original document entitled *Domain Administrators Operations Guide*, by M. Lottor, which is produced and distributed by the DDN Network Information Center.

2. ZONES

A zone defines the contents of a contiguous section of the domain space, usually bounded by administrative boundaries. There will typically be a separate data file for each zone. The data contained in a zone file is composed of entries called Resource Records (RRs). You may only put data in the domain server for which you are the administrator. You must not add entries for domains other than your own (except in the case of glue records).

A domain server will probably read a file on startup that lists the zones it should load into its database. The format of this file is not standardized and is different for most domain server implementations. For each zone it will normally contain the domain name of the zone and the file name that contains the data to load for the zone.

3. ROOT SERVERS

A `resolver` will need to find the root servers when it first starts. When the `resolver` boots, it will typically read a list of possible root servers from a file.

The `resolver` will cycle through the list trying to contact each one. When it finds a root server, it will ask it for the current list of root servers. It will then discard the list of root servers it read from the data file and replace it with the current list.

Root servers will not change very often. You can get the names of current root servers from the NIC by `ftp`ing the file `NETINFO:ROOT-SERVERS.TXT` or sending a mail request to `NIC@SRI-NIC.ARPA`.

As of this date (June 1987) they are:

```
SRI-NIC.ARPA    10.0.0.51       26.0.0.73
C.ISI.EDU       10.0.0.52
BRL-AOS.ARPA    192.5.25.82     192.5.22.82     128.20.1.2
A.ISI.EDU       26.3.0.103
```

4. RESOURCE RECORDS

Records in the zone data files are called Resource Records (RRs). They are specified in RFC883 and RFC973. An RR has a standard format as shown:

name [*ttl*] [*class*] *type* *data*

The record is divided into fields which are separated by white space.

name The *name* field defines what domain name applies to the given RR. In some cases the *name* field can be left blank and it will default to the *name* field of the previous RR.

ttl *ttl* stands for "time to live." It specifies how long a domain `resolver` should cache the RR before it throws it out and asks a domain server again. See section 4.2, "*ttl*s – Time to Live." If you leave the *ttl* field blank, it will default to the minimum time specified in the `SOA` record (described later).

class The *class* field specifies the protocol group. If left blank, it will default to the last class specified.

type The *type* field specifies what type of data is in the RR. See section 4.4, "Types."

data The *data* field is defined differently for each type and class of data. Popular RR data formats are described later.

The domain system does not guarantee to preserve the order of resource records. Listing RRs (such as multiple address records) in a certain order does not guarantee they will be used in that order.

Case is preserved in names and data fields when loaded into the name server. All comparisons and lookups in the name server are case insensitive.

The following characters have special meanings:

( ) Parentheses are used to group data that crosses a line boundary.

; A semicolon starts a comment; the remainder of the line is ignored.

* The asterisk is used for wildcarding.

@ The at sign denotes the current default domain name.

4.1 Names

A domain name is a sequence of labels separated by dots.

Domain names in the zone files can be one of two types: either absolute or relative. An absolute name is the fully qualified domain name and is terminated with a period. A relative name does not terminate with a period, and the current default domain is appended to it. The default domain is usually the name of the domain that was specified in the boot file that loads each zone.

The domain system allows a label to contain any 8-bit character. Although the domain system has no restrictions, other protocols such as SMTP do have name restrictions. Because of other protocol restrictions, only the following characters are recommended for use in a host name (besides the dot separator): A-Z, a-z, 0-9, dash, and underscore.

4.2 *ttl*s – Time to Live

It is important that *ttl*s are set to appropriate values. The *ttl* is the length of time (in seconds) that a `resolver` will use the data it got from your server before it asks your server again. If you set the value too low, your server will get loaded down with lots of repeat requests. If you set it too high, then information you change will not get distributed in a reasonable amount of time. If you leave the *ttl* field blank, it will default to what is specified in the `SOA` record for the zone.

Most host information does not change much over long time periods. A good way to set up your *ttl*s would be to set them at a high value, and then lower the value if you know a change will be coming soon. You might set most *ttl*s to anywhere between a day (`86400`) and a week (`604800`). Then, if you know some data will be changing in the near future, set the *ttl* for that RR down to a lower value (an hour to a day) until the change takes place, and then put it back up to its previous value.

Also, all RRs with the same *name*, *class*, and *type* should have the same *ttl* value.

4.3 Classes

The domain system was designed to be protocol independent. The *class* field is used to identify the protocol group that each RR is in.

The class of interest to people using TCP/IP software is the class `Internet`. Its standard designation is `IN`. A zone file should only contain RRs of the same class.

4.4 Types

There are many defined RR types. For a complete list, see the domain specification RFCs. Here is a list of current commonly used types. The data for each type is described in the data section.

Designation	*Description*
`SOA`	Start of Authority
`NS`	Name Server
`A`	Address
`CNAME`	Canonical Name (nickname pointer)
`HINFO`	Host Information
`WKS`	Well-Known Services
`MX`	Mail Exchanger
`PTR`	Pointer

4.4.1 `SOA` – *Start of Authority*

The Start of Authority record designates the start of a zone. The zone ends at the next `SOA` record.

```
name        [ttl]   [class]     SOA     origin          person (
                    ; serial>
                    ; refresh>
                    ; retry>
                    ; expire>
                    ; minimum> )
```

name is the name of the zone.

origin is the name of the host on which the master zone file resides.

person is a mailbox for the person responsible for the zone. It is formatted like a mailing address but the at sign (@) that normally separates the user from the host name is replaced with a dot.

The `serial` number is the version number of the zone file. It should be incremented whenever a change is made to data in the zone.

`refresh` indicates how often, in seconds, a secondary name server is to check with the primary name server to see if an update is needed. A good value would be 1 hour (`3600`).

`retry` indicates how long, in seconds, a secondary name server is to retry after a failure to check for a refresh. A good value would be 10 minutes (`600`).

`expire` is the upper limit, in seconds, that a secondary name server is to use the data before it expires for lack of getting a refresh. You want this to be rather large, and a good value is `3600000`, about 42 days.

`minimum` is the minimum number of seconds to be used for *ttl* values in RRs. A minimum of at least 1 day is a good value (`86400`).

There should only be one `SOA` record per zone. A sample `SOA` record would look something like:

```
@   IN      SOA     SRI-NIC.ARPA.  HOSTMASTER.SRI-NIC.ARPA. (
                    45          ; serial
                    3600        ; refresh
                    600         ; retry
                    3600000     ; expire
                    86400 )     ; minimum
```

4.4.2 `NS` — *Name Server*

The `NS` record lists the name of a machine that provides domain service for a particular domain.

domain [*ttl*] [*class*] *NS* *server*

The name associated with the RR is the domain name, and the data portion is the name of a host that provides the service. If machines `SRI-NIC.ARPA` and `C.ISI.EDU` provide name lookup service for the domain `COM`, then the following entries would be used:

```
COM.    NS   SRI-NIC.ARPA.
        NS   C.ISI.EDU.
```

Note that the machines providing name service do not have to live in the named domain. There should be one `NS` record for each server for a domain. Also note that the name `COM` defaults for the second `NS` record.

`NS` records for a domain exist in both the zone that delegates the domain and in the domain itself.

4.4.2.1 Glue Records. If the name server host for a particular domain is itself inside the domain, then a glue record will be needed. A glue record is an `A` (address) RR (see section 4.4.3 below) that specifies the address of the server. Glue records are only needed in the server delegating the domain, not in the domain

itself. If for example the name server for domain SRI.COM was KL.SRI.COM, then the NS record would look like this, but you will also need to have the following A record:

```
SRI.COM.     NS
KL.SRI.COM.      KL.SRI.COM. A    10.1.0.2.
```

4.4.3 A – *Address*

The data for an A record is an Internet address in dotted decimal form.

host [*ttl*] [*class*] *A* *address*

A sample A record might look like:

```
SRI-NIC.ARPA.            A        10.0.0.51
```

There should be one A record for each address of a host.

4.4.4 CNAME – *Canonical Name*

The CNAME record is used for nicknames.

nickname [*ttl*] [*class*] *CNAME* *host*

The name associated with the RR is the nickname. The data portion is the official name. For example, a machine named SRI-NIC.ARPA may want to have the nickname NIC.ARPA. In that case, the following RR would be used:

```
NIC.ARPA.        CNAME        SRI-NIC.ARPA.
```

There must not be any other RRs associated with a nickname of the same class.

Nicknames are also useful when a host changes its name. In that case, it is usually a good idea to have a CNAME pointer so that people still using the old name will get to the right place.

4.4.5 HINFO – *Host Info*

The HINFO record gives information about a particular host. The data is comprised of two strings separated by white space. The first string is a hardware description, and the second names the software. The hardware string is usually a manufacturer name followed by a dash and model designation. The software string is usually the name of the operating system.

host [*ttl*] [*class*] *HINFO* *hardware* *software*

Official HINFO types can be found in the *Assigned Numbers RFC*, the latest of which is RFC1010. The hardware type is called the machine name, and the software type is called the system name.

Following are some sample HINFO records:

```
SRI-NIC.ARPA.              HINFO   DEC-2060        TOPS20
UCBARPA.Berkeley.EDU.      HINFO   VAX-11/780      UNIX
```

4.4.6 WKS – Well-Known Services

The WKS record is used to list well-known services a host provides. WKSs are defined to be services on port numbers below 256. The WKS record lists what services are available at a certain address using a certain protocol. The common protocols are TCP or UDP.

host [*ttl*] [*class*] *WKS* *address* *protocol* *services*

A sample WKS record for a host offering the same services on all address would look like:

```
SRI-NIC.ARPA.   WKS     10.0.0.51   TCP     TELNET FTP SMTP
                WKS     10.0.0.51   UDP     TIME
                WKS     26.0.0.73   TCP     TELNET FTP SMTP
                WKS     26.0.0.73   UDP     TIME
```

Official protocol names can be found in the latest *Assigned Numbers RFC*, the latest of which is RFC1010.

4.4.7 MX – Mail Exchanger

(See RFC974 for more details.)

name [*ttl*] [*class*] *MX* *preference* *host*

MX records specify where mail for a domain name should be delivered. There may be multiple MX records for a particular name. The *preference* value specifies the order in which a mailer should try multiple MX records when delivering mail. Zero is the highest preference. Multiple records for the same name may have the same preference.

The host BAR.FOO.COM may want its mail to be delivered to the host PO.FOO.COM and would then use the MX record:

```
BAR.FOO.COM.    MX      10          PO.FOO.COM.
```

The host BAZ.FOO.COM may want its mail to be delivered to one of three different machines, in the following order:

```
BAZ.FOO.COM.    MX      10          PO1.FOO.COM.
                MX      20          PO2.FOO.COM.
                MX      30          PO3.FOO.COM.
```

An entire domain of hosts not connected to the Internet may want their mail to go through a mail gateway that knows how to deliver mail to them. If they would like mail addressed to any host in the domain FOO.COM to go through the mail gateway, they might use:

```
FOO.COM.        MX      10      RELAY.CS.NET.
*.FOO.COM.      MX      20      RELAY.CS.NET.
```

Note that you can specify a wildcard in the MX record to match on anything in FOO.COM, but that it won't match a plain FOO.COM.

4.4.8 PTR — *Pointer*

4.4.8.1 IN-ADDR.ARPA. The structure of names in the domain system is set up in a hierarchical way such that the address of a name can be found by tracing down the domain tree contacting a server for each label of the name. Because of this "indexing" based on name, there is no easy way to translate a host address back into its host name.

In order to do the reverse translation easily, a domain was created that uses hosts' addresses as part of a name that then points to the data for that host. In this way, there is now an index to hosts' RRs based on their address. This address mapping domain is called IN-ADDR.ARPA. Within that domain are subdomains for each network, based on network number. Also, for consistency and natural groupings, the four octets of a host number are reversed.

For example, the ARPANET is net 10. That means there is a domain called 10.IN-ADDR.ARPA. Within this domain there is a PTR (pointer) RR at 51.0.0.10.IN-ADDR that points to the RRs for the host SRI-NIC.ARPA (whose address is 10.0.0.51). Since the NIC is also on the MILNET (Net 26, address 26.0.0.73), there is also a PTR RR at 73.0.0.26.IN-ADDR.ARPA that points to the same RRs for SRI-NIC.ARPA. The format of these special pointers is defined below, along with the examples for the NIC:

special_name [*ttl*] [*class*] *PTR* *name*

The PTR record is used to let special names point to some other location in the domain tree. They are mainly used in the IN-ADDR.ARPA records for translation of addresses to names. PTRs should use official names and not aliases.

For example, host SRI-NIC.ARPA with addresses 10.0.0.51 and 26.0.0.73 would have the following records in the respective zone files for net 10 and net 26:

```
51.0.0.10.IN-ADDR.ARPA.     PTR     SRI-NIC.ARPA.
73.0.0.26.IN-ADDR.ARPA.     PTR     SRI-NIC.ARPA.
```

4.4.8.2 Gateway PTRs. The IN-ADDR tree is also used to locate gateways on a particular network. Gateways have the same kind of PTR RRs as hosts (as above), but in addition they have other PTRs used to locate them by network number alone. These records have only one, two, or three octets as part of the name depending on whether they are class A, B, or C networks respectively.

The SRI-CSL gateway, for example, connects three different networks, one class A, one class B, and one class C. It will have the standard RRs for a host in the CSL.SRI.COM zone:

```
GW.CSL.SRI.COM.     A       10.2.0.2
                    A       128.18.1.1
                    A       192.12.33.2
```

Also, in three different zones (one for each network), it will have one of the following number-to-name translation pointers:

```
2.0.2.10.IN-ADDR.ARPA.      PTR     GW.CSL.SRI.COM.
1.1.18.128.IN-ADDR.ARPA.    PTR     GW.CSL.SRI.COM.
1.33.12.192.IN-ADDR.ARPA.   PTR     GW.CSL.SRI.COM.
```

In addition, in each of the same three zones will be one of the following gateway location pointers:

```
10.IN-ADDR.ARPA.            PTR     GW.CSL.SRI.COM.
18.128.IN-ADDR.ARPA.        PTR     GW.CSL.SRI.COM.
33.12.192.IN-ADDR.ARPA.     PTR     GW.CSL.SRI.COM.
```

5. INSTRUCTIONS

5.1 Adding a Subdomain

To add a new subdomain to your domain:

- Set up the other domain server and/or the new zone file.
- Add an `NS` record for each server of the new domain to the zone file of the parent domain.
- Add any necessary glue RRs.

5.2 Adding a Host

To add a new host to your zone files:

- Edit the appropriate zone file for the domain the host is in.
- Add an entry for each address of the host.
- Optionally add `CNAME`, `HINFO`, `WKS`, and `MX` records.
- Add the reverse `IN-ADDR` entry for each host address in the appropriate zone files for each network the host in on.

5.3 Deleting a Host

To delete a host from the zone files:

- Remove all the hosts' resource records from the zone file of the domain the host is in.
- Remove all the hosts' `PTR` records from the `IN-ADDR` zone files for each network the host was on.

5.4 Adding Gateways

- Follow instructions for adding a host.
- Add the gateway location `PTR` records for each network the gateway is on.

5.5 Deleting Gateways

- Follow instructions for deleting a host.
- Also delete the gateway location `PTR` records for each network the gateway was on.

6. COMPLAINTS

These are the suggested steps you should take if you are having problems that you believe are caused by someone else's name server:

1. Complain privately to the responsible person for the domain. You can find the person's mailing address in the `SOA` record for the domain.
2. Complain publicly to the responsible person for the domain.
3. Ask the NIC for the administrative person responsible for the domain. Complain. You can also find domain contacts on the NIC in the file `NETINFO:DOMAIN-CONTACTS.TXT`.
4. Complain to the parent domain authorities.
5. Ask the parent authorities to excommunicate the domain.

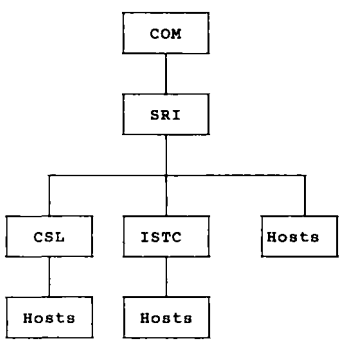
7. EXAMPLE DOMAIN SERVER DATABASE FILES

The following examples show how zone files are set up for a typical organization. SRI will be used as the example organization. SRI has decided to divided their domain SRI.COM into a few subdomains, one for each group that wants one. The subdomains are CSL and ISTC.

Note the following interesting items:

- There are both hosts and domains under SRI.COM.
- CSL.SRI.COM is both a domain name and a host name.
- All the domains are serviced by the same pair of domain servers.
- All hosts at SRI are on net 128.18 except hosts in the CSL domain which are on net 192.12.33. Note that a domain does not have to correspond to a physical network.
- The examples do not necessarily correspond to actual data in use by the SRI domain.

SRI Domain Organization



```
;
; File: config.cmd
; Since bootstrap files are not standardized, this file is presented
; using a pseudo configuration file syntax
load root server list                    from file ROOT.SERVERS
load zone SRI.COM.                       from file SRI.ZONE
load zone CSL.SRI.COM.                   from file CSL.ZONE
load zone ISTC.SRI.COM.                  from file ISTC.ZONE
load zone 18.128.IN-ADDR.ARPA.           from file SRINET.ZONE
load zone 33.12.192.IN-ADDR.ARPA.        from file SRI-CSL-NET.ZONE


;
; File: root.servers
; The format of this file is not standardized
; list of possible root servers
SRI-NIC.ARPA            10.0.0.51       26.0.0.73
C.ISI.EDU               10.0.0.52
BRL-AOS.ARPA            192.5.25.82     192.5.22.82     128.20.1.2
A.ISI.EDU               26.3.0.103


;
; File: sri.zone
SRI.COM.    IN          SOA     KL.SRI.COM.     DLE.STRIPE.SRI.COM. (
                                870407; serial
                                1800; refresh every 30 minutes
                                600; retry every 10 minutes
                                604800; expire after a week
                                86400; default of an hour
                                )


SRI.COM.    NS          KL.SRI.COM.
            NS          STRIPE.SRI.COM.
            MX          10      KL.SRI.COM.


;SRI.COM hosts
KL          A           10.1.0.2
            A           128.18.10.6
            MX          10      KL.SRI.COM.
STRIPE      A           10.4.0.2
STRIPE      A           128.18.10.4
            MX          10      STRIPE.SRI.COM.

NIC         CNAME       SRI-NIC.ARPA.

Blackjack   A           128.18.2.1
            HINFO       VAX-11/780          UNIX
            WKS         128.18.2.1          TCP TELNET FTP

CSL         A           192.12.33.2
            HINFO       FOONLY-F4           TOPS20
            WKS         192.12.33.2         TCP TELNET FTP SMTP FINGER
            MX          10      CSL.SRI.COM.
```

```
; File: csl.zone
;

CSL.SRI.COM.    IN      SOA     KL.SRI.COM.  DLE.STRIPE.SRI.COM. (
                                870330      ;serial
                                1800        ;refresh every 30 minutes
                                600         ;retry every 10 minutes
                                604800      ;expire after a week
                                86400       ;default of a day
                                )

CSL.SRI.COM.    NS              KL.SRI.COM.
                NS              STRIPE.SRI.COM.
                A               192.12.33.2

; CSL.SRI.COM hosts

A               CNAME   CSL.SRI.COM.
B               A       192.12.33.3
                HINFO   FOONLY-F4   TOPS20
                WKS     192.12.33.3 TCP TELNET FTP SMTP
GW              A       10.2.0.2
                A       192.12.33.1
                A       128.18.1.1
                HINFO   PDP-11/23   MOS
SMELLY          A       192.12.33.4
                HINFO   IMAGEN      IMAGEN
SQUIRREL        A       192.12.33.5
                HINFO   XEROX-1100  INTERLISP
VENUS           A       192.12.33.7
                HINFO   SYMBOLICS-3600                      LISPM
HELIUM          A       192.12.33.30
                HINFO   SUN-3/160   UNIX
ARGON           A       192.12.33.31
                HINFO   SUN-3/75    UNIX
RADON           A       192.12.33.32
                HINFO   SUN-3/75    UNIX
```

```
; File: istc.zone
;

ISTC.SRI.COM.   IN      SOA     KL.SRI.COM. roemers.JOYCE.ISTC.SRI.COM. (
                        870406  ;serial
                        1800    ;refresh every 30 minutes
                        600     ;retry every 10 minutes
                        604800  ;expire after a week
                        86400   ;default of a day
                        )

ISTC.SRI.COM.   NS              KL.SRI.COM.
                NS              STRIPE.SRI.COM.
                MX              10 SPAM.ISTC.SRI.COM.

; ISTC hosts

joyce           A               128.18.4.2
                HINFO           VAX-11/750 UNIX
bozo            A               128.18.0.6
                HINFO           SUN        UNIX
sundae          A               128.18.0.11
                HINFO           SUN        UNIX
tsca            A               128.18.0.201
                A               10.3.0.2
                HINFO           VAX-11/750 UNIX
                MX              10         TSCA.ISTC.SRI.COM.
tsc             CNAME           tsca
prmh            A               128.18.0.203
                A               10.2.0.51
                HINFO           PDP-11/44  UNIX
spam            A               128.18.4.3
                A               10.2.0.107
                HINFO           VAX-11/780 UNIX
                MX              10         SPAM.ISTC.SRI.COM.
```

```
; File: srinet.zone
;

18.128.IN-ADDR.ARPA.    IN     SOA     KL.SRI.COM  DLE.STRIPE.SRI.COM. (
                               870406  ;serial
                               1800    ;refresh every 30 minutes
                               600     ;retry every 10 minutes
                               604800  ;expire after a week
                               86400   ;default of a day
                               )

18.128.IN-ADDR.ARPA.    NS  KL.SRI.COM.
                        NS  STRIPE.SRI.COM.
                        PTR GW.CSL.SRI.COM.

; SRINET [128.18.0.0] Address Translations

; SRI.COM Hosts
1.2.18.128.IN-ADDR.ARPA.    PTR     Blackjack.SRI.COM.

; ISTC.SRI.COM Hosts
2.4.18.128.IN-ADDR.ARPA.    PTR     joyce.ISTC.SRI.COM.
6.0.18.128.IN-ADDR.ARPA.    PTR     bozo.ISTC.SRI.COM.
11.0.18.128.IN-ADDR.ARPA.   PTR     sundae.ISTC.SRI.COM.
201.0.18.128.IN-ADDR.ARPA.  PTR     tsca.ISTC.SRI.COM.
203.0.18.128.IN-ADDR.ARPA.  PTR     prmh.ISTC.SRI.COM.
3.4.18.128.IN-ADDR.ARPA.    PTR     spam.ISTC.SRI.COM.

; CSL.SRI.COM Hosts
1.1.18.128.IN-ADDR.ARPA.    PTR     GW.CSL.SRI.COM.

; File: sri-csl-net.zone
;

33.12.192.IN-ADDR.ARPA.  IN   SOA     KL.SRI.COM  DLE.STRIPE.SRI.COM. (
                              870404  ;serial
                              1800    ;refresh every 30 minutes
                              600     ;retry every 10 minutes
                              604800  ;expire after a week
                              86400   ;default of a day
                              )

33.12.192.IN-ADDR.ARPA.     NS              KL.SRI.COM.
                            NS              STRIPE.SRI.COM.
                            PTR             GW.CSL.SRI.COM.

; SRI-CSL-NET [192.12.33.0] Address Translations

; SRI.COM Hosts
2.33.12.192.IN-ADDR.ARPA.   PTR     CSL.SRI.COM.

; CSL.SRI.COM Hosts
1.33.12.192.IN-ADDR.ARPA.   PTR     GW.CSL.SRI.COM.
3.33.12.192.IN-ADDR.ARPA.   PTR     B.CSL.SRI.COM.
4.33.12.192.IN-ADDR.ARPA.   PTR     SMELLY.CSL.SRI.COM.
5.33.12.192.IN-ADDR.ARPA.   PTR     SQUIRREL.CSL.SRI.COM.
7.33.12.192.IN-ADDR.ARPA.   PTR     VENUS.CSL.SRI.COM.
30.33.12.192.IN-ADDR.ARPA.  PTR     HELIUM.CSL.SRI.COM.
31.33.12.192.IN-ADDR.ARPA.  PTR     ARGON.CSL.SRI.COM.
32.33.12.192.IN-ADDR.ARPA.  PTR     RADON.CSL.SRI.COM.
```

8. BIND EXAMPLES

BIND is the Berkeley Internet Name Domain server distributed with the UNIX* System.

This section describes two BIND implementation-specific files; the boot file and the cache file. BIND has other options, files, and specifications that are not described here. See the "Name Server Operations Guide for BIND" in this guide for details.

The boot file for BIND is usually called `named.boot`. This corresponds to the file `CONFIG.CMD` in section 7.

```
; File: named.boot
;

cache       .                       named.ca
primary     SRI.COM                 SRI.ZONE
primary     CSL.SRI.COM             CSL.ZONE
primary     ISTC.SRI.COM            ISTC.ZONE
primary     18.128.IN-ADDR.ARPA     SRINET.ZONE
primary     33.12.192.IN-ADDR.ARPA  SRI-CSL-NET.ZONE
```

The cache file for BIND is usually called `named.ca`. This corresponds to the file `ROOT.SERVERS` in section 7.

```
;
        ; File: named.ca
        ;

        ; list of possible root servers
        .                   1       IN      NS      SRI-NIC.ARPA.
                                            NS      C.ISI.EDU.
                                            NS      BRL-AOS.ARPA.
                                            NS      C.ISI.EDU.
        ; and their addresses
        SRI-NIC.ARPA.                       A       10.0.0.51
                                            A       26.0.0.73
        C.ISI.EDU.                          A       10.0.0.52
        BRL-AOS.ARPA.                       A       192.5.25.82
                                            A       192.5.22.82
                                            A       128.20.1.2
        A.ISI.EDU.                          A       26.3.0.103
```

REFERENCES

[Dunlap] Dunlap, K., *Name Server Operations Guide for BIND*, CSRG, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California.

[Partridge] Partridge, C., "Mail Routing and the Domain System," RFC-974, CSNET CIC BBN Laboratories, January 1986.

[Mockapetris] Mockapetris, P., "Domain Names – Concepts and Facilities," RFC-1034, USC/Information, November 1987.

[Mockapetris] Mockapetris, P., "Domain Names – Implementations Specification," RFC-1035, USC/Information, Sciences Institute.

Line Printer Spooler Manual

CONTENTS

Line Printer Spooler Manual

1. INTRODUCTION

This document describes the structure and installation procedure for the line printer spooling system.

The line printer system supports:

- Multiple printers and spooling queues
- Both local and remote printers
- Printers attached via serial lines that require line initialization, such as the baud rate

Raster output devices, such as a Varian or Versatec, and laser printers, such as an IMAGEN*, are also supported by the line printer system.

The line printer system consists mainly of the following files and commands:

`/etc/printcap`	printer configuration and capability database
`/usr/lib/lpd`	line printer daemon which does all the real work
`/usr/ucb/lpr`	program to enter a job in a printer queue
`/usr/ucb/lpq`	spooling queue examination program
`/usr/ucb/lprm`	program to delete jobs from a queue
`/etc/lpc`	program to administer printers and spooling queues
`/dev/printer`	socket on which `lpd` listens

The file `/etc/printcap` is a master database that describes line printers that are directly attached to a machine as well as printers that are accessible across a network. The manual entry *printcap*(5) provides the authoritative definition of the format of this database and specifies default values, such as the directory in which spooling is performed. This document introduces some of the information that can be placed in `printcap`.

This article is reprinted (with format and editorial changes) from the University of California, Berkeley document entitled *4.3BSD Line Printer Spooler Manual*, by Ralph Campbell.

2. COMMANDS

2.1 `lpd` – Line Printer Daemon

The program *lpd*(1M), usually invoked at boot time from the `/etc/rc` file, acts as a master server for coordinating and controlling the spooling queues configured in the `printcap` file. When `lpd` is started, it makes a single pass through the `printcap` database, restarting any printers that have jobs. In normal operation `lpd` listens for service requests on multiple sockets, one in the UNIX* System domain (named `/dev/printer`) for local requests and one in the Internet domain (under the "printer" service specification) for requests for printer access from off machine; see *socket*(7) and *services*(5) for more information on sockets and service specifications, respectively. `lpd` spawns a copy of itself to process the request; the master daemon continues to listen for new requests.

Clients communicate with `lpd` using a simple transaction-oriented protocol. Authentication of remote clients is done based on the "privilege port" scheme employed by *rshd*(1M) and *rcmd*(3I). The following table shows the requests understood by `lpd`. In each request the first byte indicates the "meaning" of the request, followed by the name of the printer to which it should be applied. Additional qualifiers may follow, depending on the request.

Request	*Interpretation*
^Aprinter\n	Check the queue for jobs and print any found
^Bprinter\n	Receive and queue a job from another machine
^Cprinter [users ...][jobs ...]\n	Return a short list of current queue state
^Dprinter [users ...][jobs ...]\n	Return a long list of current queue state
^Eprinter person [users ...][jobs ...]\n	Remove jobs from a queue

2.2 `lpr` – Enter Jobs in a Queue

The *lpr*(1) command is used by users to enter a print job in a local queue and to notify the local `lpd` that there are new jobs in the spooling area. `lpd` either schedules the job to be printed locally, or if printing remotely, attempts to forward the job to the appropriate machine. If the printer cannot be opened or the destination

machine is unreachable, the job will remain queued until it is possible to complete the work.

2.3 `lpq` – Show Line Printer Queue

The *lpq*(1) program works recursively backwards, displaying the queue of the machine with the printer and then the queue(s) of the machine(s) that lead to it. `lpq` has two forms of output: in the default (short) format it gives a single line of output per queued job; in the long format it shows the list of files (and their sizes) that comprise a job.

2.4 `lprm` – Remove Jobs From a Queue

The *lprm*(1) command deletes jobs from a spooling queue. If necessary, `lprm` will first kill off a running daemon that is servicing the queue and restart it after the required files are removed. When removing jobs destined for a remote printer, `lprm` acts similarly to `lpq` except it first checks locally for jobs to remove and then tries to remove files in queues off-machine.

2.5 `lpc` – Line Printer Control Program

The *lpc*(1M) program is used by the system administrator to control the operation of the line printer system. For each line printer configured in `/etc/printcap`, `lpc` may be used to:

- Disable or enable a printer
- Disable or enable a printer's spooling queue
- Rearrange the order of jobs in a spooling queue
- Find the status of printers and their associated spooling queues and printer daemons

3. ACCESS CONTROL

The printer system maintains protected spooling areas so that users cannot circumvent printer accounting or remove files other than their own. The strategy used to maintain protected spooling areas is as follows:

- The spooling area is writable only by a `daemon` user and `daemon` group.
- The `lpr` program runs set-user-ID to `root` and set-group-ID to group `daemon`. The `root` access permits reading any file required. Accessibility is verified with an *access*(2) call. The group ID is used in setting up proper ownership of files in the spooling area for `lprm`.
- Control files in a spooling area are made with `daemon` ownership and group ownership `daemon`. Their mode is 0660. This ensures control files are not modified by a user and that no user can remove files except through `lprm`.
- The spooling programs, `lpd`, `lpq`, and `lprm` run set-user-ID to `root` and set-group-ID to group `daemon` to access spool files and printers.
- The printer server, `lpd`, uses the same verification procedures as *rshd*(1M) in authenticating remote clients. The host on which a client resides must be present in the `/etc/hosts.equiv` or `/etc/hosts.lpd` file, and the request message must come from a reserved port number.

In practice, `lpd`, `lpq`, or `lprm` would not have to run as user `root` if remote spooling were not supported. In previous incarnations of the printer system, `lpd` ran set-user-ID to `daemon`, and set-group-ID to group `spooling`, and `lpq` and `lprm` ran set-group-ID to group `spooling`.

4. SETTING UP

This release comes with the necessary programs installed and with the default line printer queue created. If the system must be modified, the `makefile` in the `/usr/src/usr.lib/lpr` directory should be used in recompiling and reinstalling the necessary programs.

The real work in setting up is to create the `printcap` file and any printer filters for printers not supported in the distribution system.

4.1 Creating a `printcap` File

The `printcap` database contains one or more entries per printer. A printer should have a separate spooling directory; otherwise, jobs will be printed on different printers depending on which printer daemon starts first. This section describes how to create entries for printers that do not conform to the default printer description (an LP-11 style interface to a standard, band printer).

4.1.1 Printers on Serial Lines

When a printer is connected via a serial communication line, it must have the proper baud rate and terminal modes set. The following example is for a DECwriter* III printer connected locally via a 1200 baud serial line:

```
lp|LA-180 DECwriter III:\
        :lp=/dev/lp:br#1200:fs#06320:\
        :tr=\f:of=/usr/lib/lpf:lf=/usr/adm/lpd-errs:
```

The `lp` entry specifies the file name to open for output. Here it could be left out since `/dev/lp` is the default. The `br` entry sets the baud rate for the tty line, and the `fs` entry sets CRMOD, no parity, and XTABS (see *tty*(1)). The `tr` entry indicates that a form-feed should be printed when the queue empties so the paper can be torn off without turning the printer off-line and pressing form-feed. The `of` entry specifies that the filter program `lpf` should be used for printing the files; more will be said about filters later. The last entry causes errors to be written to the file `/usr/adm/lpd-errs` instead of to the console. Most errors from `lpd` are logged using *syslogd*(1M) and will not be logged in the specified file. The filters should use `syslogd` to report errors; only those that write to standard error output will end up with errors in the `lf` file. (Occasionally errors sent to standard error output have not appeared in the log file; the use of `syslogd` is highly recommended.)

4.1.2 Remote Printers

Printers that reside on remote hosts should have an empty `lp` entry. For example, the following `printcap` entry would send output to the printer named `lp` on the machine `ucbvax`:

```
lp|default line printer:\
        :lp=:rm=ucbvax:rp=lp:sd=/usr/spool/vaxlpd:
```

The `rm` entry is the name of the remote machine to connect to; this name must be a known host name for a machine on the network. The `rp` capability indicates that the name of the printer on the remote machine is `lp`; here it could be left out since this is the default value. The `sd` entry specifies `/usr/spool/vaxlpd` as the spooling directory instead of the default value `/usr/spool/lpd`.

4.2 Output Filters

Filters are used to handle device dependencies and to do accounting functions. The output filtering of `of` is used when accounting is not being done or when all text data must be passed through a filter. It is not intended to do accounting since it is started only once, all text files are filtered through it, and no provision is made for passing owners' login names, identifying the beginning and ending of jobs, etc. The other filters (if specified) are started for each file printed and do accounting if there is an `af` entry. If entries for both `of` and other filters are specified, the output filter is used only to print the banner page; it is then stopped to allow other filters access to the printer. An example of a printer that requires output filters is the Benson-Varian:

```
va|varian|Benson-Varian:\
        :lp=/dev/va0:sd=/usr/spool/vad:of=/usr/lib/vpf:\
        :tf=/usr/lib/rvcat:mx#2000:pl#58:px=2112:py=1700:tr=\f:
```

The `tf` entry specifies `/usr/lib/rvcat` as the filter to be used in printing *troff*(1) output. This filter is needed to set the device into print mode for text and into plot mode for printing `troff` files and raster images. Note that the page length is set to 58 lines by the `pl` entry for 8½ x 11-inch fanfold paper. To enable accounting, the Varian entry would be augmented with an `af` filter as shown below:

```
va|varian|Benson-Varian:\
        :lp=/dev/va0:sd=/usr/spool/vad:of=/usr/lib/vpf:\
        :if=/usr/lib/vpf:tf=/usr/lib/rvcat:af=/usr/adm/vaacct:\
        :mx#2000:pl#58:px=2112:py=1700:tr=\f:
```

4.3 Access Control

Local access to printer queues is controlled with the `rg` `printcap` entry:

```
:rg=lprgroup:
```

Users must be in the group `lprgroup` to submit jobs to the specified printer. The default is to allow all users access. Note that once the files are in the local queue, they can be printed locally or forwarded to another host depending on the configuration.

Remote access is controlled by listing the hosts in either the file `/etc/hosts.equiv` or `/etc/hosts.lpd`, one host per line. Note that *rsh*(1C) and *rlogin*(1C) use the file `/etc/hosts.equiv` to determine which hosts are equivalent for allowing logins without passwords. The `/etc/hosts.lpd` file is only used to control which hosts have line printer access. Remote access can be further restricted to only allow remote users with accounts on the local host to print jobs by using the `rs` `printcap` entry:

```
:rs:
```

5. OUTPUT FILTER SPECIFICATIONS

The filters supplied handle printing and accounting for most common line printers, the Benson-Varian, and the wide (36-inch) and narrow (11-inch) Versatec printer/plotters. For other devices or accounting methods, it may be necessary to create a new filter.

Filters are spawned by `lpd` with the data to be printed as their standard input and with the printer as the standard output. The standard error is attached to the `lf` file for logging errors, or `syslogd` may be used for logging errors. A filter must return a 0 exit code if there are no errors, 1 if the job should be reprinted, and 2 if the job should be thrown away. When `lprm` sends a kill signal to the `lpd` process controlling printing, it sends a `SIGINT` signal to all filters and descendents of filters. This signal can be trapped by filters that need to do cleanup operations, such as deleting temporary files.

Arguments passed to a filter depend on the type of filter. The `of` filter is called with the following arguments:

filter `-w`*width* `-l`*length*

The *width* and *length* values come from the `pw` and `pl` entries in the `printcap` database. The `if` filter is passed the following parameters:

filter `[ -c ]` `-w`*width* `-l`*length* `-i`*indent* `-n` *login* `-h` *host accounting_file*

The `-c` flag is optional and only supplied when control characters are to be passed uninterpreted to the printer (when using the `-l` option of `lpr` to print the file). The `-w` and `-l` parameters are the same as for the `of` filter. The `-n` and `-h` parameters specify the login name and host name of the job owner. The last argument is the name of the accounting file from `printcap`.

All other filters are called with the following arguments:

filter `-x`*width* `-y`*length* `-n` *login* `-h` *host accounting_file*

The `-x` and `-y` options specify the horizontal and vertical page size in pixels (from the `px` and `py` entries in the `printcap` file). The rest of the arguments are the same as for the `if` filter.

6. LINE PRINTER ADMINISTRATION

The `lpc` program provides local control over line printer activity. The major commands and their intended uses are described here. The command format and remaining commands are described in *lpc*(1M).

`abort` and `start`
: `abort` terminates an active spooling daemon on the local host immediately and then disables printing (preventing new daemons from being started by `lpr`). This is normally used to forcibly restart a hung line printer daemon, i.e., `lpq` reports that there is a daemon present but nothing is happening. It does not remove any jobs from the queue; use the `lprm` command instead. `start` enables printing and requests `lpd` to start printing jobs.

`enable` and `disable`
: `enable` and `disable` allow spooling in the local queue to be turned on/off. This allows/prevents `lpr` from putting new jobs in the spool queue. It is frequently convenient to turn spooling off while testing new line printer filters since the `root` user can still use `lpr` to put jobs in the queue but no one else can. The other main use is to prevent users from putting jobs in the queue when the printer is expected to be unavailable for a long time.

`restart`
: `restart` allows ordinary users to restart printer daemons when `lpq` reports that there is no daemon present.

`stop`
: `stop` halts a spooling daemon after the current job completes; this also disables printing. This is a clean way to shut down a printer to do maintenance, etc. Note that users can still enter jobs in a spool queue while a printer is stopped.

`topq`
: `topq` places jobs at the top of a printer queue. This can be used to reorder high priority jobs since `lpr` only provides first-come-first-serve ordering of jobs.

7. TROUBLESHOOTING

There are several messages that may be generated by the line printer system. This section categorizes the most common and explains the causes for their generation. Where the message implies a failure, directions are given to remedy the problem.

In the examples below, the name *printer* is the name of the printer from the `printcap` database.

7.1 `lpr`

`lpr:` *printer*`: unknown printer`
: The *printer* was not found in the `printcap` database. Usually this is a typing mistake; however, it may indicate a missing or incorrect entry in the file `/etc/printcap`.

`lpr:` *printer*`: jobs queued, but cannot start daemon`
: The connection to `lpd` on the local machine failed. This usually means that the printer server started at boot time has died or is hung. Check the local socket `/dev/printer` to be sure it still exists (if it does not exist, there is no `lpd` process running). Usually it is enough to have the superuser type the following to restart `lpd`:

```
% /usr/lib/lpd
```

You can also check the state of the master printer daemon with the following:

```
% ps l`cat /usr/spool/lpd.lock`
```

Another possibility is that the `lpr` program is not set-user-ID to `root` and set-group-ID to group `daemon`. This can be checked with:

```
% ls -lg /usr/ucb/lpr
```

`lpr:` *printer*`: printer queue is disabled`
: This means the queue was turned off with:

```
% lpc disable printer
```

to prevent `lpr` from putting files in the queue. This is normally done by the system manager when a printer is going to be down for a long time. The printer can be turned back on by the superuser with `lpc`.

7.2 lpq

`waiting for` *printer* `to become ready (offline?)`
: The printer device could not be opened by the daemon. This can happen for several reasons, the most common being that the printer is turned off-line. This message is also be generated if the printer is out of paper, the paper is jammed, etc. The actual reason is dependent on the meaning of the error codes returned by the system device driver. Not all printers supply enough information to distinguish when a printer is off-line or having trouble, e.g., a printer is connected through a serial line. Another possible cause of this message is that some other process, such as an output filter, has an exclusive open on the device. In this instance, your only recourse is to kill the offending program(s) and restart the printer with `lpc`.

printer `is ready and printing`
: The `lpq` program checks to see if a daemon process exists for *printer* and prints the file *status* located in the spooling directory. If the daemon is hung, the superuser can use `lpc` to abort the current daemon and start a new one.

`waiting for` *host* `to come up`
: This implies there is a daemon trying to connect to the remote machine named *host* to send the files in the local queue. If the remote machine is up, `lpd` on the remote machine is probably dead or hung and should be restarted as mentioned for `lpr`.

`sending to` *host*
: The files should be in the process of being transferred to the remote *host*. If not, the local daemon should be aborted and started with `lpc`.

`Warning:` *printer* `is down`
: The printer has been marked as being unavailable with `lpc`.

`Warning: no daemon present`
: The `lpd` process overseeing the spooling queue, as specified in the "lock" file in that directory, does not exist. This normally occurs only when the daemon has

unexpectedly died. The error log file for the printer and the `syslogd` logs should be checked for a diagnostic from the deceased process. To restart an `lpd`, use:

```
% lpc restart printer
```

`no space on remote; waiting for queue to drain`
This implies that there is insufficient disk space on the remote. If the file is large enough, there will never be enough space on the remote (even after the queue on the remote is empty). The solution is to move the spooling queue or make more free space on the remote.

7.3 `lprm`

`lprm: printer: cannot restart printer daemon`
This case is the same as when `lpr` prints that the daemon cannot be started.

7.4 `lpd`

The `lpd` program can log many different messages using *syslogd*(1M). Most of these messages are about files that cannot be opened and usually imply that the `printcap` file or the protection modes of the files are incorrect. Files may also be inaccessible if users manually manipulate the line printer system, i.e., they bypass the `lpr` program.

In addition to messages generated by `lpd`, any of the filters that `lpd` spawns may log messages using `syslogd` or log them to the error log file (the file specified in the `lf` entry in `printcap`).

7.5 `lpc`

`couldn't start printer`
This case is the same as when `lpr` reports that the daemon cannot be started.

`cannot examine spool directory`
Error messages beginning with "cannot ..." are usually due to the incorrect ownership or protection mode of the lock file, spooling directory, or the `lpc` program.

INTERACTIVE TCP/IP
Reference Manual

CONTENTS

NAME

arp – address resolution display and control

SYNOPSIS

arp hostname
arp **–a**
arp **–d** hostname
arp **–s** hostname ether_addr [**temp**] [**pub**]
arp **–f** filename
arp **–t** [**interval** [**killcom** [**killinc**]]]

DESCRIPTION

The *arp* program displays and modifies the Internet-to-Ethernet address translation tables used by the Address Resolution Protocol (ARP) (see arp*(7)*).

With no flags, the program displays the current ARP entry for *hostname*. The host may be specified by name or by number, using Internet dot notation. With the **–a** flag, the program displays all of the current ARP entries by reading the table from the kernel. Each entry has the following displayed: table index, machine name, Internet address, Ethernet address, and the number of minutes since the last access.

With the **–d** flag, the superuser may delete an entry for the host called *hostname*.

The **–s** flag is given to 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 given in the command. If the word **pub** is given, the entry will be "published;" i.e., this system will act as an ARP server, responding to requests for *hostname* even though the host address is not its own.

The **–f** flag 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**]

with argument meanings as given above.

The **–t** flag, by itself, prints the current timeout interval, complete entry timeout, and incomplete entry timeout. Specifying values for these parameters with **–t** sets the parameters.

SEE ALSO

ifconfig(1M), inet(3I), arp(7).

NAME

bootpd – Internet Boot Protocol server

SYNOPSIS

/etc/bootpd [**–s** **–t**timeout **–d**] [configfile [dumpfile]]

DESCRIPTION

bootpd implements an Internet Boot Protocol server as defined in RFC951 and RFC1048. It is normally run by **/etc/inetd** by including the following line in the file **/etc/inetd.conf**:

bootps dgram udp wait root /etc/bootpd bootpd

This causes *bootpd* to be started only when a boot request arrives. If *bootpd* does not receive another boot request within 15 minutes of the last one it received, it will exit to conserve system resources. The **–t** switch may be used to specify a different *timeout* value in minutes, e.g., **–t***20*. A *timeout* value of zero means "forever."

It is also possible to run *bootpd* in a standalone configuration using the **–s** switch (for example, at boot time from **/etc/rc.local**). This is probably the desired mode of operation for large network installations with many hosts. In this case, the **–t** switch has no effect since *bootpd* will never exit.

Each instance of the **–d** switch increases the level of debugging output.

Upon startup, *bootpd* first reads its configuration file **/etc/bootptab**, then it begins listening for BOOTREQUEST packets. The configuration file has a format similar to that of *terminfo*(4), in which two-character case-sensitive tag symbols are used to represent host parameters. These parameter declarations are separated by colons (:). The general format is:

hostname:tg=value. . . :tg=value. . . :tg=value. . . .

where *hostname* is the actual name of a *bootp* client and *tg* is a two-character tag symbol. Most tags must be followed by an equal sign (=) and a value as shown above. Some tags may also appear in a boolean form with no value, i.e., :*tg*:. The currently recognized tags are:

bf Bootfile

bs Bootfile size in 512-octet blocks

cs Cookie server address list

ds Domain name server address list

gw Gateway address list

ha Host hardware address

hd Bootfile home directory

hn Send hostname

ht Host hardware type (see Assigned Numbers RFC)

im Impress server address list

ip Host IP address

lg Log server address list

lp LPR server address list

ns IEN-116 name server address list

rl Resource location protocol server address list

sm Host subnet mask

tc Table continuation (points to similar "template" host entry)

to Time offset in seconds from UTC

ts Time server address list

vm Vendor magic cookie selector

There is also a generic tag, **T***n*, where *n* is an RFC1048 vendor field tag number. Thus, it is possible to immediately take advantage of future extensions to RFC1048 without being forced to modify *bootpd* first. Generic data may be represented as either a stream of hexadecimal numbers or as a quoted string of ASCII characters. The length of the generic data is automatically determined and inserted into the proper field(s) of the RFC1048-style *bootp* reply.

The following tags take a list of IP addresses separated by white spaces: **cs**, **ds**, **gw**, **im**, **lg**, **lp**, **ns**, **rl**, and **ts**. The **ip** and **sm** tags each take a single IP address. All IP addresses are specified in standard Internet "dot" notation and may use decimal, octal, or hexadecimal numbers (octal numbers begin with **0**; hexadecimal numbers begin with **0x** or **0X**).

The **ht** tag specifies the hardware type code as either an unsigned decimal, octal, or hexadecimal integer or as one of the following symbolic names: **ethernet** or **ether** for 10Mb Ethernet, **ethernet3** or **ether3** for 3Mb experimental Ethernet, **ieee802**, **tr**, or **token-ring** for IEEE 802 networks, **pronet** for Proteon ProNET Token Ring, or **chaos**, **arcnet**, or **ax.25** for Chaos, ARCNET, and Amateur Radio networks, respectively. The **ha** tag takes a hardware address which *must* be specified in hexadecimal; optional periods and/or a leading **0x** may be included for readability. The **ha** tag must be preceded by the **ht** tag (either explicitly or implicitly; see **tc** below).

The hostname, home directory, and bootfile are ASCII strings which may optionally be surrounded by double quotes ("). The client's request and the values of the **hd** and **bf** symbols determine how the server fills in the *bootfile* field of the *bootp* reply packet.

If the client specifies an absolute path name and that file exists on the server machine, that path name is returned in the reply packet. If the file cannot be found, the request is discarded; no reply is sent. If the client specifies a relative path name, a full path name is formed by prepending the value of the **hd** tag and testing for the existence of the file. If the **hd** tag is not supplied in the configuration file or if the resulting bootfile cannot be found, then the request is discarded.

Clients that specify null bootfiles will always elicit a reply from the server. The exact reply will again depend upon the **hd** and **bf** tags. If the **bf** tag gives an absolute path name and the file exists, that path name is returned in the reply packet. Otherwise, if the **hd** and **bf** tags

together specify an accessible file, that file name is returned in the reply. If a complete file name cannot be determined or the file does not exist, the reply will contain a zeroed-out *bootfile* field.

In all these cases, existence of the file means that in addition to actually being present, the file must have its public read access bit set, since this is required by *tftpd*(1M) to permit the file transfer. Also, all file names are first tried as *filename.hostname* and then simply as *filename*, thus providing for individual per-host bootfiles.

The time offset **to** may be either a signed decimal integer specifying the client's time zone offset in seconds from UTC or the keyword **auto** which uses the server's time zone offset. Specifying the **to** symbol as a boolean has the same effect as specifying **auto** as its value.

The bootfile size **bs** may be either a decimal, octal, or hexadecimal integer specifying the size of the bootfile in 512-octet blocks or the keyword **auto** which causes the server to automatically calculate the bootfile size at each request. As with the time offset, specifying the **bs** symbol as a boolean has the same effect as specifying **auto** as its value.

The vendor magic cookie selector (the **vm** tag) may take one of the following keywords: **auto** (indicating that vendor information is determined by the client's request), **rfc1048** (which always forces an RFC1048-style reply), or **cmu** (which always forces a CMU-style reply).

The **hn** tag is strictly a boolean tag; it does not take the usual equal sign and value. Its presence indicates that the hostname should be sent to RFC1048 clients. *bootpd* attempts to send the entire hostname as it is specified in the configuration file; if this will not fit into the reply packet, the name is shortened to just the host field (up to the first period, if present) and then tried. In no case is an arbitrarily-truncated hostname sent (if nothing reasonable will fit, nothing is sent).

Often, many host entries share common values for certain tags (such as name servers, etc.). Rather than repeatedly specifying these tags, a full specification can be listed for one host entry and shared by others via the **tc** (table continuation) mechanism. Often, the template entry is a dummy host which does not actually exist and never sends *bootp* requests. This feature is similar to the **tc** feature of *terminfo*(4) for similar terminals. Note that *bootpd* allows the **tc** tag symbol to appear anywhere in the host entry, unlike *terminfo* which requires it to be the last tag. Information explicitly specified for a host always overrides information implied by a **tc** tag symbol, regardless of its location within the entry. The value of the **tc** tag may be the hostname or IP address of any host entry previously listed in the configuration file.

Sometimes it is necessary to delete a specific tag after it has been inferred via **tc**. This can be done using the construction *tag***@** which removes the effect of *tag* as in *terminfo*(4). For example, to completely undo an IEN-116 name server specification, use **:ns@:** at an appropriate place in the configuration entry. After removal with **@**, a tag is eligible to be set again through the **tc** mechanism.

Blank lines and lines beginning with **#** are ignored in the configuration file. Host entries are separated from one another by newlines; a single

host entry may be extended over multiple lines by ending each line with a backslash (\). It is also acceptable for lines to be longer than 80 characters. Tags may appear in any order, with the following exceptions: the hostname must be the very first field in an entry, and the hardware type must precede the hardware address.

An example **/etc/bootptab** file follows:

```
# Sample bootptab file
default1:\
        :hd=/usr/boot:bf=null:\
        :ds=128.2.35.50 128.2.13.21:\
        :ns=0x80020b4d 0x80020ffd:\
        :ts=0x80020b4d 0x80020ffd:\
        :sm=255.255.0.0:gw=0x8002fe24:\
        :hn:vm=auto:to=-18000:\
        :T37=0x12345927AD3BCF:T99="Special ASCIII string":
carnegie:ht=6:ha=7FF8100000AF:ip=128.2.11.1:tc=default1:
baldwin:ht=1:ha=0800200159C3:ip=128.2.11.10:tc=default1:
wylie:ht=1:ha=00DD00CADF00:ip=128.2.11.100:tc=default1:
arnold:ht=1:ha=0800200102AD:ip=128.2.11.102:tc=default1:
bairdford:ht=1:ha=08002B02A2F9:ip=128.2.11.103:tc=default1:
bakerstown:ht=1:ha=08002B0287C8:ip=128.2.11.104:tc=default1:
# Special domain name server for next host
gastonville:ht=6:ha=7FFF81000A47:ip=128.2.11.115:tc=default1:
hahntown:ht=6:ha=7FFF81000434:ip=128.2.11.117:tc=default1:
hickman:ht=6:ha=7FFF810001BA:ip=128.2.11.118:tc=default1:
lowber:ht=1:ha=00DD00CAF000:ip=128.2.11.121:tc=default1:
mtoliver:ht=1:ha=00DD00FE1600:ip=128.2.11.122:tc=default1:
```

bootpd looks in **/etc/services** to find the port numbers it should use. Two entries are extracted:

bootps – the *bootp* server listening port

bootpc – the destination port used to reply to clients

If the port numbers cannot be determined this way, they are assumed to be 67 for the server and 68 for the client.

bootpd rereads its configuration file when it receives a hangup signal, SIGHUP, or when it receives a *bootp* request packet and detects that the file has been updated. Hosts may be added, deleted, or modified when the configuration file is reread. *bootpd* is compiled with the **–DDEBUG** option, which means that receipt of a SIGUSR1 signal causes it to dump its memory-resident database to the file **/etc/bootpd.dump** or to the dumpfile specified on the command line.

FILES

/etc/bootptab
/etc/bootpd.dump
/etc/services

SEE ALSO

inetd(1M), tftpd(1M).
terminfo(4) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

DARPA Internet Request For Comments RFC951, RFC1048, RFC1084, Assigned Numbers.

BUGS

Individual host entries must not exceed 1024 characters.

NAME

finger – user information lookup program

SYNOPSIS

finger [options] name ...

DESCRIPTION

By default *finger* lists the login name, full name, terminal name and write status (as a * before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each current UNIX System user. (Idle time is minutes if it is a single integer, hours and minutes if a : is present, or days and hours if a **d** is present.)

A longer format also exists and is used by *finger* whenever a list of people's names is given. (Account names as well as first and last names of users are accepted.) This format is multi-line, and includes all the information described above as well as the user's home directory and login shell, any plan that the person has placed in the file **.plan** in their home directory, and the project on which they are working from the file **.project** also in the home directory.

finger may be used to look up users on a remote machine. The format is to specify the user as *user@host*. If the user name is left off, the standard format listing is provided on the remote machine.

finger options include:

–m Match arguments only on user name.

–l Force long output format.

–p Suppress printing of the **.plan** files.

–s Force short output format.

FILES

/etc/utmp	who file
/etc/passwd	for users names, offices, ...
/usr/adm/wtmp	login history
~/.plan	plans
~/.project	projects

SEE ALSO

passwd(1), who(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.

BUGS

Only the first line of the **.project** file is printed.

The encoding of the gcos field is UCB dependent – it knows that an office '197MC' is '197M Cory Hall', and that '529BE' is '529B Evans Hall'. It also knows that a four-digit office phone number should have a "x2-" prefixed to it.

There is no way to pass arguments to the remote machine as *finger* uses an internet standard port.

A user information database is under development that will radically alter the way the information that *finger* uses is stored. *finger* will require extensive modification when this is implemented.

NAME

fingerd – remote user information server

SYNOPSIS

/etc/fingerd [name ...]

DESCRIPTION

fingerd is a simple protocol based on RFC742 that provides an interface to the *name* and *finger* programs at several network sites. The program is supposed to return a friendly, human-oriented status report on either the system at the moment or a particular person in depth. There is no required format, and the protocol consists mostly of specifying a single "command line."

fingerd listens for TCP requests at port 79. Once connected it reads a single command line terminated by a **<CRLF>** which is passed to *finger*(1). *fingerd* closes its connections as soon as the output is finished.

If the line is null (i.e., just a **<CRLF>** is sent), then *finger* returns a "default" report that lists all people logged into the system at that moment.

If a user name is specified (e.g., **eric<CRLF>**), then the response gives more detailed information for the named user, even if that user is not logged in. Allowable *name*s in the command line include both "login names" and "user names." If a name is ambiguous, all possible derivations are returned.

SEE ALSO

finger(1).

BUGS

Connecting directly to the server from a TIP or an equally limited TELNET-protocol user program can result in meaningless attempts at option negotiation being sent to the server, which will disrupt the command line interpretation. *fingerd* should be taught to filter out IACs and perhaps even respond negatively (IAC WON'T) to all option commands received.

NAME

ftp – ARPANET file transfer program

SYNOPSIS

ftp [–v] [–d] [–i] [–n] [–g] [host]

DESCRIPTION

The *ftp* program is the user interface to the ARPANET standard File Transfer Protocol (FTP). The program allows a user to transfer files to and from a remote network site.

The client host with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* will immediately attempt to establish a connection to an FTP server on that host; otherwise, *ftp* will enter its command interpreter and await instructions from the user. When *ftp* is awaiting commands from the user, the prompt **ftp>** is displayed. The following commands are recognized by *ftp*:

! [*command* [*args*]]
: Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.

$ *macro-name* [*args*]
: Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro unglobbed.

account [*passwd*]
: Supply a supplemental password required by a remote system for access to resources once a *login* has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

append *local-file* [*remote-file*]
: Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file after being altered by any **ntrans** or **nmap** setting. File transfer uses the current settings for **type**, **form**, **mode**, and **struct**.

ascii
: Set the file transfer **type** to network ASCII. This is the default type.

bell
: Specify that a bell be sounded after each file transfer command is completed.

binary
: Set the file transfer **type** to support binary image transfer.

bye
: Terminate the FTP session with the remote server and exit *ftp*. An end-of-file will also terminate the session and exit.

case
: Toggle remote computer file name case mapping during **mget** commands. When **case** is **on** (the default is off), remote computer file names with all letters in uppercase are written in the local directory with the letters mapped to lowercase.

cd *remote-directory*
: Change the working directory on the remote machine to *remote-directory*.

cdup Change the remote machine working directory to the parent of the current remote machine working directory.

chmod *mode file-name*
Change the permission modes of the file *file-name* on the remote system to *mode*.

close Terminate the FTP session with the remote server and return to the command interpreter. Any defined macros are erased.

cr Toggle carriage return stripping during ASCII-type file retrieval. Records are denoted by a carriage return/line-feed sequence during ASCII-type file transfer. When **cr** is on (the default), carriage returns are stripped from this sequence to conform with the UNIX System single line-feed record delimiter. Records on non-UNIX System remote systems may contain single line-feed characters; when an ASCII-type transfer is made, these line-feed characters can be distinguished from a record delimiter only when **cr** is off.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified, it is used to set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string **-->**.

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, place the output in *local-file*. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **dir** output. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is **-**, output comes to the terminal.

disconnect
A synonym for **close**.

form *format*
Set the file transfer **form** to *format*. The default format is file.

get *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current **case**, **ntrans**, and **nmap** settings. The current settings for **type**, **form**, **mode**, and **struct** are used while transferring the file.

glob Toggle file name expansion for **mdelete**, **mget**, and **mput**. If globbing is turned off with **glob**, the file name arguments are taken literally and not expanded. Globbing for **mput** is done as in *csh*(1). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine, and the lists are

not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file; the exact result depends on the foreign operating system and FTP server, and can be previewed by using:

mls remote-files −

Note that **mget** and **mput** are not meant to transfer entire directory subtrees of files. That can be done by transferring a *tar*(1) archive of the subtree (in binary mode).

hash Toggle hash-sign (**#**) printing for each data block transferred. The size of a data block is 1024 bytes.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

idle [*seconds*]
Set the inactivity timer on the remote server to *seconds* seconds. If *seconds* is omitted, the current inactivity timer is printed.

image A synonym for **binary**.

lcd [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
Print a listing of the contents of a directory on the remote machine. The listing includes any system-dependent information that the server chooses to include; for example, most UNIX Systems will produce output from the command *ls* **−l**. (See also **nlist**.) If *remote-directory* is left unspecified, the current working directory is used. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **ls** output. If no local file is specified, or if *local-file* is −, the output is sent to the terminal.

macdef *macro-name*
Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive new-line characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed. The macro processor interprets the dollar sign (**$**) and backslash (\) as special characters. A **$** followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A **$** followed by an **i** signals that macro processor that the executing macro is to be looped. On the first pass **$i** is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A \ followed by any character is

replaced by that character. Use the \ to prevent special treatment of the **$**.

mdelete [*remote-files*]
Delete the *remote-files* on the remote machine.

mdir *remote-files local-file*
Like **dir**, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

mget *remote-files*
Expand the *remote-files* on the remote machine and do a **get** for each file name thus produced. See **glob** for details on the file name expansion. Resulting file names will then be processed according to **case, ntrans,** and **nmap** settings. Files are transferred into the local working directory, which can be changed with **lcd** *directory*; new local directories can be created with:

```
! mkdir directory
```

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Like **nlist**, except multiple remote files may be specified, and the *local-file* must be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mls** output.

mode [*mode-name*]
Set the file transfer **mode** to *mode-name*. The default mode is stream mode.

modtime *file-name*
Show the last modification time of the file on the remote machine.

mput *local-files*
Expand wildcards in the list of local files given as arguments and do a **put** for each file in the resulting list. See **glob** for details of file name expansion. Resulting file names will then be processed according to **ntrans** and **nmap** settings.

newer *file-name*
Get the file only if the modification time of the remote file is more recent than the file on the current system. If the file does not exist on the current system, the remote file is considered *newer*. Otherwise, this command is identical to **get**.

nlist [*remote-directory*] [*local-file*]
Print a list of the files of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **nlist** output. If no local file is specified, or if *local-file* is −, the output is sent to the terminal.

nmap [*inpattern outpattern*]
Set or unset the file name mapping mechanism. If no arguments are specified, the file name mapping mechanism is unset. If arguments are specified, remote file names are mapped during **mput** commands, and **put** commands are issued without a specified remote target file name. If arguments are specified, local file names are mapped during **mget** commands, and **get** are commands issued without a specified local target file name. This command is useful when connecting to a non-UNIX System remote computer with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. The *inpattern* is a template for incoming file names (which may have already been processed according to the **ntrans** and **case** settings). Variable templating is accomplished by including the sequences *$1*, *$2*, ..., *$9* in *inpattern*. Use \ to prevent this special treatment of the **$** character. All other characters are treated literally and are used to determine the **nmap** *inpattern* variable values. For example, given *inpattern $1.$2* and the remote file name *mydata.data*, *$1* would have the value *mydata*, and *$2* would have the value *data*. The *outpattern* determines the resulting mapped file name. The sequences *$1*, *$2*, ..., *$9* are replaced by any value resulting from the *inpattern* template. The sequence *$0* is replaced by the original file name. Additionally, the sequence:

[*seq1*,*seq2*]

is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*. For example, the command:

nmap $1.$2.$3 [$1,$2].[$2,file]

would yield the output file name *myfile.data* for input file names *myfile.data* and *myfile.data.old*, *myfile.file* for the input file name *myfile*, and *myfile.myfile* for the input file name *.myfile*. Spaces may be included in *outpattern*, as in the example:

nmap $1 |sed "s/ *$//" > $1 .

Use the backslash character (\) to prevent special treatment of the **$**, **[**, **]**, and **,** characters.

ntrans [*inchars* [*outchars*]]
Set or unset the file name character translation mechanism. If no arguments are specified, the file name character translation mechanism is unset. If arguments are specified, characters in remote file names are translated during **mput** commands, and

put commands are issued without a specified remote target file name. If arguments are specified, characters in local file names are translated during **mget** commands, and **get** commands are issued without a specified local target file name. This command is useful when connecting to a non-UNIX System remote computer with different file naming conventions or practices. Characters in a file name matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

open *host* [*port*]
Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case *ftp* will attempt to contact an FTP server at that port. If the *autologin* option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see below).

prompt Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default is on), any **mget** or **mput** will transfer all files, and any **mdelete** will delete all files.

proxy *ftp-command*
Execute an *ftp* command on a secondary control connection. This command allows simultaneous connection to two remote FTP servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command **proxy ?** to see other *ftp* commands executable on the secondary connection. The following commands behave differently when prefaced by **proxy**: **open** will not define new macros during the auto-login process, **close** will not erase existing macro definitions, **get** and **mget** will transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mput**, and **append** will transfer files from the host on the secondary control connection to the host on the primary control connection. Third-party file transfers depend upon support of the *ftp* protocol PASV command by the server on the secondary control connection.

put *local-file* [*remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any **ntrans** or **nmap** settings in naming the remote file. File transfer uses the current settings for **type**, **form**, **mode**, and **struct**.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for **bye**.

quote *arg1 arg2 ...*
The arguments specified are sent, verbatim, to the remote FTP server.

recv *remote-file* [*local-file*]
A synonym for **get**.

reget *remote-file* [*local-file*]
reget acts like **get**, except that if *local-file* exists and is smaller than *remote-file*, *local-file* is presumed to be a partially transferred copy of *remote-file* and the transfer is continued from the apparent point of failure. This command is useful when transferring very large files over networks that are prone to dropping connections.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified, it is supplied to the server as well.

remotestatus [*file-name*]
With no arguments, show status of remote machine. If *file-name* is specified, show status of *file-name* on remote machine.

rename [*from*] [*to*]
Rename the file *from* on the remote machine to the file *to*.

reset Clear reply queue. This command re-synchronizes command/reply sequencing with the remote FTP server. Resynchronization may be necessary following a violation of the *ftp* protocol by the remote server.

restart *marker*
Restart the immediately following **get** or **put** at the indicated *marker*. On UNIX Systems, *marker* is usually a byte offset into the file.

rmdir *directory-name*
Delete a directory on the remote machine.

runique Toggle storing of files on the local system with unique file names. If a file already exists with a name equal to the target local file name for a **get** or **mget** command, a **.1** is appended to the name. If the resulting name matches another existing file, a **.2** is appended to the original name. If this process continues up to **.99**, an error message is printed and the transfer does not take place. The generated unique file name will be reported. Note that **runique** will not affect local files generated from a shell command (see below). The default value is off.

send *local-file* [*remote-file*]
A synonym for **put**.

sendport
Toggle the use of PORT commands. By default, *ftp* will attempt to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, *ftp* will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they have been accepted.

site *arg1 arg2 ...*
The arguments specified are sent, verbatim, to the remote FTP server as a SITE command.

size *file-name*
Return size of *file-name* on remote machine.

status Show the current status of *ftp*.

struct [*struct-name*]
Set the file transfer **struct** to *struct-name*. By default stream structure is used.

sunique Toggle storing of files on remote machine under unique file names. The remote server must support *ftp* protocol STOU command for successful completion. The remote server will report the unique name. The default value is off.

system Show the type of operating system running on the remote machine.

tenex Set the file transfer type to that needed to talk to TENEX machines.

trace Toggle packet tracing.

type [*type-name*]
Set the file transfer **type** to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.

umask [*newmask*]
Set the default **umask** on the remote server to *newmask*. If *newmask* is omitted, the current **umask** is printed.

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an account field is not specified and the FTP server requires it, the user will be prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless *ftp* is invoked with auto-login disabled, this process is done automatically on initial connection to the FTP server.

verbose Toggle **verbose** mode. In **verbose** mode, all responses from the FTP server are displayed to the user. In addition, if **verbose** is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, **verbose** is on.

? [*command*]
A synonym for **help**.

Command arguments that have embedded spaces may be quoted with quote (") marks.

Aborting a File Transfer

To abort a file transfer, use the terminal interrupt key (usually **CTRL c**). Sending transfers will be immediately halted. Receiving transfers will be halted by sending a *ftp* protocol ABOR command to the remote server and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an **ftp>** prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when *ftp* has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above, or from unexpected behavior by the remote server, including violations of the *ftp* protocol. If the delay results from unexpected remote server behavior, the local *ftp* program must be killed by hand.

File Naming Conventions

Files specified as arguments to *ftp* commands are processed according to the following rules:

1) If the file name − is specified, the **stdin** (for reading) or **stdout** (for writing) is used.

2) If the first character of the file name is |, the remainder of the argument is interpreted as a shell command. *ftp* then forks a shell, using *popen*(3) with the argument supplied, and reads (writes) from the *out* (*in*). If the shell command includes spaces, the argument must be quoted; e.g., "**|** *ls* **−lt**". A particularly useful example of this mechanism is: *dir* **|***more*.

3) Failing the above checks, if globbing is enabled, local file names are expanded according to the rules used in *csh*(1), e.g., the **glob** command. If the *ftp* command expects a single local file (e.g., **put**), only the first file name generated by the globbing operation is used.

4) For **mget** commands and **get** commands with unspecified local file names, the local file name is the remote file name, which may be altered by a **case**, **ntrans**, or **nmap** setting. The resulting file name may then be altered if **runique** is on.

5) For **mput** commands and **put** commands with unspecified remote file names, the remote file name is the local file name, which may be altered by a **ntrans** or **nmap** setting. The resulting file name may then be altered by the remote server if **sunique** is on.

File Transfer Parameters

The FTP specification specifies many parameters which may affect a file transfer. The **type** may be one of **ascii**, **image** (binary), **ebcdic**, and "local byte size" (generally for PDP-10 and PDP-20 systems). *ftp* supports the **ascii** and **image** types of file transfer, plus local byte size 8 for **tenex** mode transfers.

ftp supports only the default values for the remaining file transfer parameters: **mode**, **form**, and **struct**.

Options

Options may be specified at the command line or to the command interpreter.

The **−v** (verbose on) option forces *ftp* to show all responses from the remote server, as well as to report on data transfer statistics.

The **−n** option restrains *ftp* from attempting auto-login upon initial connection. If auto-login is enabled, *ftp* will check the **.netrc** file (see below) in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* will prompt for the remote machine login name (default is the user identity on the local machine) and, if necessary, prompt for a password and an account with which to log in.

The **−i** option turns off interactive prompting during multiple file transfers.

The **−d** option enables debugging.

The **−g** option disables file name globbing.

The .netrc File

The **.netrc** file contains login and initialization information used by the auto-login process. It resides in the user's home directory. The following tokens are recognized; they may be separated by spaces, tabs, or new-line characters:

machine *name*
Identify a remote machine name. The auto-login process searches the **.netrc** file for a **machine** token that matches the remote machine specified on the *ftp* command line or as an **open** command argument. Once a match is made, the subsequent .netrc tokens are processed, stopping when the end-of-file is reached or another **machine** or a **default** token is encountered.

default This is the same as **machine** *name* except that **default** matches any name. There can be only one **default** token, and it must be after all **machine** tokens. This is normally used as:

default login anonymous password user@site

thereby giving the user *automatic* anonymous *ftp* login to machines not specified in **.netrc**. This can be overridden by using the **−n** flag to disable auto-login.

login *name*
Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

password *string*
Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the **.netrc** file for any user other than *anonymous*, *ftp* will abort the auto-login process if the **.netrc** file is readable by anyone besides the user.

account *string*
Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an ACCT command if it does not.

macdef *name*
Define a macro. This token functions like the *ftp* **macdef** command functions. A macro is defined with the specified name; its contents begin with the next **.netrc** line and continue until a null line (consecutive new-line characters) is encountered. If a macro named *init* is defined, it is automatically executed as the last step in the auto-login process.

SEE ALSO

ftpd(1M).

BUGS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2BSD UNIX System **ascii**-mode transfer code has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2BSD servers using the **ascii** type. To avoid this problem, use the binary **image** type.

NAME

ftpd – DARPA Internet File Transfer Protocol server

SYNOPSIS

/etc/ftpd [**–d**] [**–l**] [**–t***timeout*] [**–T***maxtimeout*]

DESCRIPTION

ftpd is the DARPA 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).

If the **–d** option is specified, debugging information is written to the *syslog*.

If the **–l** option is specified, each *ftp* session is logged in the *syslog*.

The FTP server will timeout an inactive session after 15 minutes. If the **–t** option is specified, the inactivity timeout period will be set to *timeout* seconds. A client may also request a different timeout period; the maximum period allowed may be set to *timeout* seconds with the **–T** option. The default limit is 2 hours.

The FTP server currently supports the following *ftp* requests; case is not distinguished:

Request	*Description*
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* **–l**)
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 following non-standard or UNIX System-specific commands are supported by the SITE request:

Request	*Description*
UMASK	Change umask, e.g., SITE UMASK 002
IDLE	Xet idle-timer, e.g., SITE IDLE 60
CHMOD	Change mode of a file, e.g., SITE CHMOD 755 file name
HELP	Give help information, e.g., SITE HELP

The remaining *ftp* requests specified in Internet RFC959 are recognized but not implemented. MDTM and SIZE are not specified in RFC959, but will appear in the next updated FTP RFC.

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 RFC959. If a STAT command preceded by Telnet IP and Synch signals is received during a data transfer, transfer status will be returned.

The *ftpd* program interprets file names according to the "globbing" conventions used by *csh*(1). This allows users to utilize the metacharacters *, ?, [,], {, }, ~.

ftpd authenticates users according to four rules:

1) The user name must be in the password database, **/etc/passwd**, and it must not have a null password. In this case, a password must be provided by the client before any file operations may be performed.

2) The user name must not appear in the file **/etc/ftpusers**.

3) The user must have a standard shell returned by *getusershell*(3).

4) If the user name is "anonymous" or "ftp," an anonymous *ftp* account must be present in the password file (user "ftp"). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, *ftpd* takes special measures to restrict the client's access privileges. The server performs a *chroot*(2) command to the home directory of the "ftp" user. In order that system security is not breached, it is recommended that the "ftp" subtree be constructed with care; the following rules are recommended:

~ftp The home directory should be owned by **ftp**, and it should be unwritable by anyone.

~ftp/bin
This directory should be owned by **root** (the superuser), and it should be unwritable by anyone. The program *ls*(1) must be present to support the list command. This program should have mode **111**.

~ftp/etc
The owner of this directory should be **root**, and the directory should be unwritable by anyone. The files *passwd*(4) and *group*(4) must be present for the *ls* command to be able to produce owner names rather than numbers. The password field in *passwd* is not used and should not contain real encrypted passwords. These files should be mode **444**.

~ftp/pub
This directory should be owned by **ftp**. The directory mode should be **777** . Users should then place files which are to be accessible via the anonymous account in this directory.

~ftp/dev
This directory should be owned by **root** and it should be unwritable by anyone. The device *tcp* must be present and have the same major/minor numbers as in **/dev/tcp**.

~ftp/shlib
This directory is necessary if the *ls*(1) command in **~ftp/bin** uses shared libraries. This would be the case if the *ls* command was taken from **/bin/ls**. In this case, a copy of **/shlib/libc_s** needs to be placed in this directory in order for the list commands to work properly.

SEE ALSO
ftp(1C), syslogd(1M), getusershell(3I), services(5).

BUGS
The anonymous account is inherently dangerous and should be avoided when possible.

The server must run as user **root** to create sockets with privileged port numbers. It maintains an effective user ID of the logged in user, reverting to **root** only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but the results are possibly incomplete.

NAME

gated - gateway routing daemon

SYNOPSIS

gated [−t[i][e][r][p][u][R][H]] [logfile]

DESCRIPTION

gated is a routing daemon that handles multiple routing protocols and is meant to be a replacement for *routed*(1M), *egpup*(1M), or any routing daemon that speaks the HELLO routing protocol. *gated* currently handles the RIP, EGP, and HELLO routing protocols. The *gated* process can be configured to perform all three routing protocols or any combination of the three. The configuration for *gated* is by default stored in the file **/etc/gated.conf**, but can be changed at compile time.

gated can be invoked with a number of debug flags and/or a log file. When debug flags are used, the process does not fork and detach itself from the controlling terminal. If there is no log file specified, all debug output is sent to the controlling terminal; otherwise, the debugging output is sent to the log file specified. The valid debugging flags are as follows:

−t If used alone, log all error messages, route changes and packets sent and received. Using this flag alone turns on the **i**, **e**, **r**, and **p** debug flags automatically. When used with another flag, the **−t** has no effect and only the accompanying flags are recognized. Note that when using other flags, you must use the **−t** with them.

i Log all internal errors and interior routing errors.

e Log all external errors due to EGP, exterior routing errors, and EGP state changes.

r Log all routing changes.

p Trace all EGP packets sent and received.

u Log all routing updates sent.

R Trace all RIP packets received.

H Trace all HELLO packets received.

The *gated* process always logs fatal errors. If no log file is specified and no debugging flags are set, all messages are sent to **/dev/null**.

Signal Processing

gated catches a number of signals and performs specific actions when caught. Currently *gated* does special processing with the SIGHUP and SIGINT signals. When a SIGHUP is sent to the *gated* process and *gated* was invoked with debug flags and a log file, debugging is toggled off and the log file is closed. At this point, the log file may be moved or removed. The next SIGHUP to *gated* will toggle the debugging on using the same debug flags in the command line. The log file specified in the command line is created, if necessary, and the debug output is sent to that file. The debug output is appended to an already existing log file. This is useful for having rotating log files much like the *syslog* daemon.

Sending *gated* a SIGINT will cause a memory dump to be scheduled within the next 60 seconds. The memory dump will be written to the file **/usr/tmp/gated_dump**. *gated* will finish processing any routing updates currently being worked on before performing the memory dump. The memory dump contains all relevant routing information and interface information. If the file **/usr/tmp/gated_dump** already exists, the memory dump will be appended to the existing file.

Configuration File Options for Routing Protocols

In this section, the numerous configuration options are explained. Each time the *gated* process is started, it reads the file **/etc/gated.conf** to obtain its instructions on how routing will be managed with respect to each protocol. The configuration options are as follows:

RIP {yes | no | supplier | pointopoint | quiet | gateway #}

This tells the *gated* process how to perform the RIP routing protocol. Only one of the above RIP arguments is allowed after the keyword "RIP". If more than one is specified, only the first one is recognized. The arguments are:

yes Perform the RIP protocol. Process all incoming RIP packets and supply RIP information every 30 seconds only if there are two or more network interfaces.

no Do not perform the RIP protocol. Do no RIP processing.

supplier Perform the RIP protocol. Process all incoming RIP packets and force the supplying of RIP information every 30 seconds, no matter how many network interfaces are present.

pointopoint
Perform the RIP protocol. Process all incoming RIP packets and force the supplying of RIP information every 30 seconds, no matter how many network interfaces are present. When this argument is specified, RIP information will not be sent out in a broadcast packet. The RIP information will be sent directly to the gateways listed in the "sourceripgateways" option described below.

quiet Process all incoming RIP packets, but do not supply any RIP information no matter how many network interfaces are present.

gateway #
Process all incoming RIP packets, supply RIP information every 30 seconds, and announce the default route (0.0.0.0) with a metric of #. The metric should be specified in a value that represents a RIP hopcount. With this option set, all other default routes coming from other RIP gateways will be ignored.

If no "RIP" clause is specified, RIP will not be performed.

HELLO {yes | no | supplier | pointopoint | quiet | gateway #}

This tells the *gated* process how to perform the HELLO routing protocol. The arguments parallel the RIP arguments, but do have some minor differences. Only one of the above HELLO arguments is allowed

after the keyword "HELLO". If more than one is specified, only the first one is recognized. The arguments:

yes Perform the HELLO protocol. Process all incoming HELLO packets and supply HELLO information every 15 seconds only if there are two or more network interfaces.

no Do not perform the HELLO protocol. Don't do HELLO processing.

supplier Perform the HELLO protocol. Process all incoming HELLO packets and force the supplying of HELLO information every 15 seconds, no matter how many network interfaces are present.

pointopoint
Perform the HELLO protocol. Process all incoming HELLO packets and force the supplying of HELLO information every 15 seconds no matter how many network interfaces are present. When this argument is specified, HELLO information will not be sent out in a broadcast packet. The HELLO information will be sent directly to the gateways listed in the "sourcehellogateways" option described below.

quiet Process all incoming HELLO packets, but do not supply any HELLO information no matter how many network interfaces are present.

gateway #
Process all incoming HELLO packets, supply HELLO information every 15 seconds, and announce the default route (10.0.0.0) with a time delay of #. The time delay should be specified in milliseconds. Net 10 is used as a default because the NSFnet backbone gateways use net 10 as default (not 0.0.0.0). This can create complex routing situations in a UNIX System environment, so when you are running as a default HELLO gateway, you will still be able to listen and act on other net 10 information from RIP or HELLO. However, you will still announce via HELLO, net 10 with time delay #. If you are not a HELLO gateway, you will not HELLO announce net 10. If you are not connected to the ARPAnet, you should not be using this option.

If no "HELLO" clause is specified, HELLO will not be performed.

EGP {yes | no}

This clause allows the processing of EGP by *gated* to be turned on or off.

no Do not perform any EGP processing.

yes Perform all EGP operations.

Please note that, by default, EGP processing will take place. Therefore, if no "EGP"clause is specified, all EGP operations will take place.

autonomoussystem #

If performing the EGP protocol, you must use this clause to specify your autonomous system number (#). If not specified, *gated* will exit and give a fatal error message.

egpmaxacquire #

If performing the EGP protocol, this clause specifies the number of EGP peers you will be performing EGP with. This cannot be 0– *gated* will exit on such a condition.

egpneighbor gateway

If performing the EGP protocol, this clause specifies who you will be performing EGP with. "Gateway" can be either a symbolic name in **/etc/hosts** or it can be in dot (a.b.c.d) notation. Dot notation is recommended. Each EGP neighbor will be acquired in the order listed in the configuration file.

Configuration File Options for Routing Information

The following configuration file options deal with how routing knowledge is dealt with. This includes incoming knowledge as well as the knowledge we will give out.

trustedripgateways gateway [gateway] [gateway] .....
trustedhellogateways gateway [gateway] [gateway] .....

With these clauses specified, you will only listen to RIP or HELLO information respectively, from these RIP or HELLO gateways respectively. "gateway" can be either a symbolic name from **/etc/hosts** or it can be in dot format (a.b.c.d). Again, dot format is recommended to eliminate confusion. Please note that the propagation of your knowledge is not restricted in any way by this clause.

sourceripgateways gateway [gateway] [gateway] .....
sourcehellogateways gateway [gateway] [gateway] .....

You will send RIP or HELLO information directly to the gateways specified. If "pointopoint" is specified in the "RIP" or "HELLO" clauses mentioned above, you will only send RIP or HELLO information to the specified gateways. You will *not* send out any information using the broadcast address. If "pointopoint" is not specified in those clauses and you are a supplier of RIP or HELLO information, you will send information to the specified gateways as well as broadcasting it using a broadcast address.

noripoutinterface intfaddr [intfaddr] [intfaddr] .....
nohellooutinterface intfaddr [intfaddr] [intfaddr] .....
noripfrominterface intfaddr [intfaddr] [intfaddr] .....
nohellofrominterface intfaddr [intfaddr] [intfaddr] .....

Turning on and off protocols on a per-interface basis is accomplished with the above clauses. "no{rip|hello}frominterface" means that no RIP or HELLO information will be accepted coming into the listed interfaces from another gateway. "no{rip|hello}outinterface" means that no RIP or HELLO knowledge will be sent out of the listed interfaces. "intfaddr" should be in dot notation (a.b.c.d).

passiveinterfaces intfaddr [intfaddr] [intfaddr] .....

In order to dynamically determine if an interface is properly functioning, *gated* will time out an interface when no RIP or HELLO packets are being received on that particular interface. IMP interfaces send a RIP or HELLO packet to themselves to determine if the interface is properly functioning. Interfaces that have timed out automatically have their routes re-installed when routing information is again received over the interface. The above clause stops *gated* from timing out the listed interfaces. The interfaces listed will always be considered up and working. If you are not a RIP or HELLO supplier, all interfaces will *not* be aged and the "passiveinterfaces" automatically applies to all interfaces.

interfacemetric intfaddr metric#

This feature allows you to specify an interface metric for the listed interface. On systems that support interface metrics, this clause will override the kernel's metric. On systems that do not have support for an interface metric, this feature allows you to use an interface metric. The interface metric is added to the true metric of each route that comes in via routing information from the listed interface. The interface metric is also added to the true metric of any information sent out via the listed interface. You will need one clause for each interface that you want to specify an interface metric for.

reconstmetric intfaddr metric#

This is a first attempt to throw hooks for fallback routing into *gated*. If the above clause is used, any RIP information coming into the listed interface will have the metrics of the routes contained in the RIP information set to the specified "metric#". Metric Reconstitution should not be used lightly as it could be a major contributor in the formation of routing loops. USE THIS WITH EXTREME CAUTION. Any route that has a metric of infinity will not be reconstituted and left as infinity.

donotlisten net intf addr [addr] ... proto {rip|hello}
donotlistenhost host intf addr [addr] ... proto {rip|hello}

This clause reads as follows: keyword "donotlisten" followed by a network number which should be in dot format followed by keyword "intf". Then there should be a list of interfaces in dot format followed by the keyword "proto" followed by "rip" or "hello".

This means that any information regarding "net" coming in via the specified protocols AND from the specified interfaces will be ignored. You can also put the keyword "all" after the keyword "intf" to specify all interfaces on the machine. For example:

donotlisten 10.0.0.0 intf 128.84.253.200 proto rip

means that any RIP information about net 10 coming in via interface 128.84.253.200 will be ignored. You must have one clause for each net that you want to put this restriction on.

donotlisten 26.0.0.0 intf all proto rip hello

means that any RIP and HELLO information about net 26 coming in via any interface will be ignored.

"donotlistenhost" can be described the same way as above except that you provide a host address instead of a network address. Restrictions of the nature described above are applied to the specified host route learned of by the specified routing protocol.

listen net gateway addr [addr] ... proto {rip|hello}
listenhost host gateway addr [addr] ... proto {rip|hello}

This clause reads as follows: keyword "listen" followed by a network number which should be in dot format followed by keyword "gateway". Then there should be a list of gateways in dot format followed by the keyword "proto" followed by "rip" or "hello".

This means to only listen to information about network "net" by the specified protocol(s) only from the listed "gateways". Ignore all other information from any other gateway regarding this network. For example:

listen 128.84.0.0 gateway 128.84.253.3 proto hello

means that any HELLO information about net 128.84 coming in via gateway 128.84.253.3 will be accepted. Any other information about 128.84 from any other gateway will be rejected. You must have one clause for each net that you want to put this restriction on.

listenhost 26.0.0.15 gateway 128.84.253.3 proto rip

means that any information about host 26.0.0.15 must come via RIP and from gateway 128.84.253.3. All other information regarding this host will be ignored.

announce net intf addr [addr] ... proto type [egpmetric #]
announcehost host intf addr ... proto type [egpmetric #]
noannounce net intf addr [addr] ... proto type [egpmetric #]
noannouncehost host intf addr ... proto type [egpmetric #]

These clauses allow you to place restrictions on the networks and hosts you announce and by what protocol they are announced. The "announce{host}" and "noannounce{host}" clauses may not be used together on the same interface. You can only have one or the other per interface. With the "announce{host}" clause, you will only announce the nets or hosts that have an associated "announce{host}" clause with the appropriate protocol. With the "noannounce{host}" clause, you will announce everything, EXCEPT those nets or hosts that have an associated "noannounce{host}" clause. Therefore, you have a choice of announcing only what is on your announce list or everything except those nets on your noannounce list on a per-interface basis.

The arguments are the same as in the "donotlisten" clause except you can specify "egp" in the "proto" field. "type" can either be rip, hello, egp, or any combination of the three. When egp is specified in the "proto" field, you must specify an egp metric. This is the metric at which you will announce, via EGP, the listed net.

Note that these are not static route entries. These restrictions only apply if the net or host is learned of via one of the routing protocols.

If a restricted network suddenly becomes unreachable and goes away, announcement of this net will of course stop until it is learned of again. Some examples:

announce 128.84 intf all proto rip hello egp egpmetric 0
announce 10.0.0.0 intf all proto rip
announce 0.0.0.0 intf 128.84.253.200 proto rip
announce 35.0.0.0 intf all proto rip egp egpmetric 3

With only these four "announce" clauses in the configuration file, this *gated* process will only announce these four nets. It will announce 128.84 via RIP and HELLO to all interfaces and announce it via EGP with a metric of 0. Net 10 will be announced via RIP to all interfaces. Net 0 (default) will be announced by RIP out interface 128.84.253.200 only. Net 35 will be announced via RIP to all interfaces and announced via EGP with a metric of 3. These are the only nets that will be broadcast by this gateway. Once you have one "announce" clause, only the nets with "announce" clauses will be broadcast; this includes local subnets. Once an "announce{host}" or "noannounce{host}" has an "all" specified after an "intf", that clause is applied globally and you lose the option of having per-interface restrictions. If no routing announcement restrictions are desired, do not include any "announce" clauses in your configuration file. All information gained will then be propagated out. Please note that this does not affect what information is being listened to. Any net that does not have an "announce" clause is still added to the kernel routing tables, but it is not announced anywhere else by your gateway. To stop nets from being added to the kernel see the "donotlisten" clause discussed earlier.

announce 128.84 intf 128.59.2.1 proto rip
noannounce 128.84 intf 128.59.1.1 proto rip

The above clauses mean that on interface 128.59.2.1 only information about 128.84 will be announced via RIP, but on interface 128.59.1.1, all information except 128.84 via RIP will be announced.

noannounce 128.84 intf all proto rip hello egp egpmetric 0
noannounce 10.0.0.0 intf all proto hello

These clauses mean that all nets except these two that are restricted will be propagated. Specifically, net 128.84 will not be announced out any interface with any protocols. Knowledge of 128.84 is not sent anywhere. Net 10 will not be announced via HELLO to any interface. This also means that net 10 will be announced to every interface via RIP. This net will also be broadcast via EGP with a metric specified in the "defaultegpmetric" clause.

defaultegpmetric #

This is a default EGP metric to use when you have no routing restrictions. Normally, with no routing restrictions, *gated* announces all networks learned via HELLO or RIP via EGP with this specified default EGP metric. If this clause is not used, the default EGP metric is set to 255, which would make any EGP advertised route of this nature be ignored. When there are no routing restrictions, any network that you have a direct interface to is announced via EGP with a metric of 0.

Note that this does not include subnets, but only the non-subnetted network.

defaultgateway gateway proto {active|passive}

This is the initial default gateway that is installed in your kernel during startup and initialization. If you are running EGP, as soon as you receive an update from your EGP neighbor, this route is deleted. If you are not running EGP, but running RIP and "active" is specified, this default route will be deleted and overwritten by any other default it hears from any other RIP gateway. If no default routes are being propagated throughout your local network, this default route will stay in your kernel's routing table. If "passive" is specified, the default route will be added to the kernel and will *not* be overwritten by any other default route broadcast. You will also *not* propagate the default route with the "passive" option set.

"gateway" should be an address in dot notation. "proto" should be either rip, egp, or hello. The "proto" field just initializes what protocol the route was learned by. In this case it is unused, but the field retained for consistency.

net netaddr gateway addr metric hopcnt {rip|egp|hello} host hostaddr gateway addr metric hopcnt {rip|egp|hello}

The following clauses install a static route to net "netaddr" or host "hostaddr" through gateway "addr" at a metric of "hopcnt" and learned via either RIP, HELLO, or EGP. As usual, dot notation is recommended for the addresses. This route will be installed in the kernel's routing table and will never be affected by any other gateway's RIP or HELLO announcements. The protocol by which it was learned is important when it comes to announcing via EGP. If the protocol is rip or hello and there are no routing restrictions, then this route will be announced by EGP with a metric of "defaultegpmetric". If the protocol is egp and there are no routing restrictions, then this route will be announced by EGP with a metric of "hopcnt".

egpnetsreachable net [net] [net] .....

This option was left in as a "soft restriction". It cannot be used when the "announce" or "noannounce" clauses are used. Normally, with no restrictions, *gated* announces all routes learned from RIP and HELLO via EGP. The "egpnetsreachable" clause restricts EGP announcement to those nets listed in the clause. The metric used for the HELLO and RIP learned routes is the value given in the "defaultegpmetric" clause. If this clause does not specify a value, the value is set to 255. With the "egpnetsreachable" clause, you cannot set individual unique EGP metrics for each net. The "defaultegpmetric" is used straight across the board except, of course, for any direct interface. In this case, a metric of 0 is used.

Notes on Configuration Options

gated stores its Process ID in the file **/etc/gated.pid**. If running EGP, supplying the default route (RIP gateway), and you lose all of your EGP neighbors, you will stop broadcasting the default route until your EGP neighbors regain contact. If routing restrictions are used, *gated* logs all invalid networks using *syslog* at log level LOG_WARNING and facility LOG_DAEMON.

With the complexity of the current network topology and with many back door paths to networks, the use of routing restrictions is recommended. With the current routing strategies, it is easy for illegal or invalid networks to penetrate into the ARPAnet Core or the NSFnet backbone. Using routing restrictions does take a little more maintenance time and routing restrictions are not the long-term answer, but for now, in order to be good internet players, we must use them.

Notes on Implementation Specifics

gated stores all metrics internally in milliseconds. The RIP metric is mapped to a millisecond-based metric and processed. This is so that we keep the granularity of the HELLO protocol time delay intact.

In the **gated.conf** file, all references to POINT-TO-POINT interfaces must use the DESTINATION address. This is the exact opposite of the old versions, which used the source address of the PTP link. This is the only change made to the **gated.conf** syntax. Otherwise, all old **gated.conf** files should be compatible.

A 2-minute hold down is in effect for all of the protocols.

Changes can be made to the interfaces and *gated* will notice them without having to restart the process. If you change an address, netmask, subnetmask, broadcast address, or interface metric, you must first *ifconfig* the interface down and then back up again. You may have to wait about 30 seconds before *gated* realizes the interface is down. *gated* checks the interfaces every 30 seconds. You need not bring the interface up or down for flag changes.

RIP does propagate and listen to host routes. This was done to handle PTP links more consistently. The RIP_TRACE commands are also supported in this version.

Subnet interfaces are supported. This means that subnets will not get propagated where they shouldn't be. For example, if a gateway is gatewaying two class B networks, the subnet routes for each respective class B net are not propagated into the opposite class B net. Only the class B network number is propagated to the appropriate place.

gated listens to host and network REDIRECTs and tries to take an action on the REDIRECT for its own internal tables that parallels the kernel's action. In this way, the redirect routine in *gated* parallels the Berkeley kernel redirect routine as closely as possible. Unlike the Berkeley kernel, *gated* times out routes learned of via a REDIRECT after 6 minutes. The route is then deleted from the kernel routing tables. This helps keep the routing tables on your gateway more consistent and clean. Any route that was learned of via a REDIRECT is *not* announced by any routing protocol.

FILES

/etc/gated.conf	configuration file
/etc/gated.pid	where the process ID is stored
/usr/tmp/gated_dump	memory dump file
/etc/gated	the gated process

SEE ALSO

ifconfig(1M).

RFC827 (EGP formal specs.), RFC891 (HELLO formal specs.), RFC911 (EGP under the UNIX System).

NAME

hostid – set or print identifier of current host system

SYNOPSIS

hostid [identifier]

DESCRIPTION

The *hostid* command prints the identifier of the current host in hexadecimal. The value returned will be 0 unless initialized at boot time or by the super-user. The system administrator can set the host ID in a script added to the **/etc/rc3.d** directory by calling *hostid* with the host name as a parameter.

SEE ALSO

gethostid(3I).

NAME

htable – convert NIC standard format host tables

SYNOPSIS

/etc/htable [**–c** connected-nets] [**–l** local-nets] file

DESCRIPTION

htable is used to convert host files in the format specified in Internet RFC810 to the format used by the network library routines. Three files are created as a result of running *htable*: **hosts**, **networks**, and **gateways**. The **hosts** file may be used by the *gethostbyname* (see *ghostnamed*(3I)) routines in mapping host names to addresses if the nameserver, *named*(1M), is not used. The **networks** file is used by the *getnetent*(3I) routines in mapping network names to numbers. The **gateways** file may be used by the routing daemon in identifying "passive" Internet gateways; see *gated*(1M) for more information.

If any of the files **localhosts**, **localnetworks**, or **localgateways** are present in the current directory, the file's contents is prepended to the output file. Of these, only the **gateways** file is interpreted. This allows sites to maintain local aliases and entries which are not normally present in the master database. Only one gateway to each network will be placed in the **gateways** file; a gateway listed in the **localgateways** file will override any in the input file.

If the **gateways** file is to be used, a list of networks to which the host is directly connected is specified with the **–c** flag. The networks, separated by commas, may be given by name or in Internet-standard dot notation, e.g., **–c** arpanet,128.32,local-ether-net. *htable* only includes gateways which are directly connected to one of the networks specified, or which can be reached from another gateway on a connected net.

If the **–l** option is given with a list of networks (in the same format as for **–c**), these networks will be treated as "local," and information about hosts on local networks is taken only from the **localhosts** file. Entries for local hosts from the main database will be omitted. This allows the **localhosts** file to completely override any entries in the input file.

htable is best used in conjunction with the *gettable* program, which retrieves the NIC database from a host.

SEE ALSO

named(1M).

BUGS

If the name-domain system provided network name mapping as well as host name mapping, *htable* would no longer be needed.

NAME

ifconfig – configure network interface parameters

SYOPNSIS

/etc/ifconfig interface address_family [address [dest_address]] [parameters]

/etc/ifconfig interface [protocol_family]

DESCRIPTION

ifconfig is used to assign an address to a network interface and/or configure network interface parameters. *ifconfig* must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address or other operating parameters. The *interface* parameter is a string of the form "name unit," for example, **en0**.

Since an interface may receive transmissions in differing protocols, each of which may require separate naming schemes, it is necessary to specify the *address_family*, which may change the interpretation of the remaining parameters. The address families currently supported are "inet" and "ns."

For the DARPA-Internet family, the address is either a host name present in the host name data base, *hosts*(5), or a DARPA Internet address expressed in 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 10 MB/s Ethernet interfaces, which use the hardware physical address, and on interfaces other than the first.

The following parameters may be set with *ifconfig*:

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.

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.

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 10 MB/s Ethernet addresses.

–arp Disable the use of the Address Resolution Protocol. (It is currently *not* possible to disable *arp*).

metric *n* Set the routing metric of the interface to *n*, default 0. The routing metric is used by the routing protocol *gated*(1M). Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.

debug Enable driver-dependent debugging code; usually, this turns on extra console error logging.

−debug Disable driver dependent debugging code.

netmask *mask* (Inet only) Specify how much of the address to reserve for subdividing networks into subnetworks. 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 (see *networks*(5)). The mask contains 1's for the bit positions in the 32-bit address that 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.

dstaddr Specify the address of the correspondent on the other end of a point to point link.

broadcast (Inet only) Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 1's.

ifconfig 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 superuser may modify the configuration of a network interface.

SEE ALSO

netstat(1), intro(7N).

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, or the user is not privileged and tried to alter an interface's configuration.

NAME

inetd – internet "super server"

SYNOPSIS

/etc/inetd [**–d**] [configuration file]

DESCRIPTION

inetd should be run at boot time by **/etc/rc3**. It then listens for connections on certain internet sockets. When a connection is found on one of its sockets, it decides what service the socket corresponds to and invokes a program to service the request. After the program is finished, it continues to listen on the socket (except in some cases that will be described below). Essentially, *inetd* allows one daemon to invoke several others, reducing the load on the system.

Upon execution, *inetd* reads its configuration information from a configuration file which, by default, is **/etc/inetd.conf**. *inetd* then attempts to read the file **/etc/default/inetd** for the default ULIMIT, TIMEZONE, and UMASK. If **/etc/default/inetd** is not found, the file **/etc/default/login** is used. There must be an entry for each field of the configuration file, with entries for each field separated by a tab or a space. Comments are denoted by a **#** at the beginning of a line. There must be an entry for each field. The fields of the configuration file are as follows:

service name
socket type
protocol
wait/nowait
user
server program
server program arguments

The *service name* entry is the name of a valid service in the file **/etc/services**. For "internal" services (discussed below), the service name *must* be the official name of the service (that is, the first entry in **/etc/services**).

The *socket type* should be one of "stream," "dgram," or "raw," depending on whether the socket is a stream, datagram, or raw socket.

The *protocol* must be a valid protocol as given in **/etc/protocols**. Examples might be "tcp" or "udp."

The *wait/nowait* entry is applicable to datagram sockets only (other sockets should have a "nowait" entry in this space). If a datagram server connects to its peer, freeing the socket so *inetd* can receive further messages on the socket, it is said to be a "multi-threaded" server and should use the "nowait" entry. For datagram servers that process all incoming datagrams on a socket and eventually time out, the server is said to be "single-threaded" and should use a "wait" entry. "Talk" is an example of the latter type of datagram server. *tftpd* is an exception; it is a datagram server that establishes pseudo-connections. It must be listed as "wait" in order to avoid a race; the server reads the first packet, creates a new socket, then forks and exits to allow *inetd* to check for new service requests to spawn new servers.

The *user* entry should contain the user name of the user as whom the server should run. This allows servers to be given less permission than **root**. The *server program* entry should contain the path name of the program that is to be executed by *inetd* when a request is found on its socket. If *inetd* provides this service internally, this entry should be "internal."

The arguments to the server program should be just as they normally are, starting with **argv[0]**, which is the name of the program. If the service is provided internally, no arguments are required.

inetd provides several "trivial" services internally by use of routines within itself. These services are "echo," "discard," "chargen" (character generator), "daytime" (human readable time), and "time" (machine readable time, in the form of the number of seconds since midnight, January 1, 1900). All of these services are TCP-based. For details of these services, consult the appropriate RFC from the Network Information Center.

inetd rereads its configuration file when it receives a hangup signal, SIGHUP. Services may be added, deleted or modified when the configuration file is reread.

SEE ALSO

ftpd(1M), rexecd(1M), rlogind(1M), rshd(1M), telnetd(1M), tftpd(1M).

NAME

logger − make entries in the system log

SYNOPSIS

logger [**−t** tag] [**−p** pri] [**−i**] [**−f** file] [message ...]

DESCRIPTION

logger provides a program interface to the *syslog*(3I) system log module.

A message can be given on the command line, which is logged immediately, or a file is read and each line is logged.

The following options are available:

−t *tag* Mark every line in the log with the specified *tag*.

−p *pri* Enter the message with the specified priority. The priority may be specified numerically or as a *facility.level* pair. For example, "**−p** *local3.info*" logs the message(s) as *info*rmational level in the *local3* facility. The default is *user.notice*.

−i Log the process ID of the logger process with each line.

−f *file* Log the specified file.

message The message to log; if not specified, the **−f** file or standard input is logged.

Examples

logger System rebooted

logger −p local0.notice −t HOSTIDM −f /dev/idmc

SEE ALSO

syslogd(1M), syslog(3I).

NAME

lpc – line printer control program

SYNOPSIS

/etc/lpc [command [argument ...]]

DESCRIPTION

lpc is used by the system administrator to control the operation of the line printer system. For each line printer configured in **/etc/printcap**, *lpc* can be used to:

Disable or enable a printer

Disable or enable a printer's spooling queue

Rearrange the order of jobs in a spooling queue

Find the status of printers and their associated spooling queues and printer daemons

If no arguments are given, *lpc* prompts for commands from the standard input. If arguments are supplied, *lpc* interprets the first argument as a command and the remaining arguments as parameters to the command. The standard input may be redirected, causing *lpc* to read commands from a file. Commands may be abbreviated. The following commands are recognized:

? [command ...]
help [command ...]
: Print a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

abort { all | printer ... }
: Terminate an active spooling daemon on the local host immediately, and then disable printing (preventing new daemons from being started by *lpr*) for the specified printers.

clean { all | printer ... }
: Remove any temporary files, data files, and control files that cannot be printed (i.e., do not form a complete printer job) from the specified printer queue(s) on the local machine.

disable { all | printer ... }
: Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by *lpr*.

down { all | printer } message ...
: Turn the specified printer queue off, disable printing, and put *message* in the printer status file (the message need not be enclosed in quotation marks). The remaining arguments are treated like *echo*(1). This is normally used to take a printer down and let others know why the action was taken (*lpq* will indicate the printer is down and print the status message).

enable { all | printer ... }
: Enable spooling on the local queue for the listed printers. This will allow *lpr* to put new jobs in the spool queue.

exit
quit Exit from *lpc*.

restart { all | printer ... }
Attempt to start a new printer daemon for the named printer or for all printers ("all"). This is useful when some abnormal condition causes the daemon to "die" unexpectedly, leaving jobs in the queue. *lpq* will report that there is no daemon present when this condition occurs. If the user is the superuser (**root**), it will try to abort the current daemon first, i.e., kill and restart a stuck daemon.

start { all | printer ... }
Enable printing and start a spooling daemon for the listed printers.

status { all | printer ... }
Display the status of daemons and queues on the local machine.

stop { all | printer ... }
Stop a spooling daemon after the current job completes and disable printing.

topq printer [jobnum ...] [user ...]
Place the jobs in the order listed at the top of the printer queue.

up { all | printer ... }
Enable everything and start a new printer daemon. This will undo the effects of *down*.

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/usr/spool/*/lock	lock file for queue control

SEE ALSO

lpd(1M), lpq(1), lpr(1), lprm(1), printcap(5).
"Line Printer Spooler Manual."

DIAGNOSTICS

?Ambiguous command	abbreviation matches more than one command; use the full command name
?Invalid command	no match was found
?Privileged command	command can be executed by **root** only

NAME

lpd – line printer daemon

SYNOPSIS

/usr/lib/lpd [**–l**] [port #]

DESCRIPTION

lpd is the line printer daemon (spool area handler) and is normally invoked at boot time. It makes a single pass through the *printcap*(5) file to find out about the existing printers and prints any files left after a crash. It then uses the system calls *listen*(3I) and *accept*(3I) to receive requests to print files in the queue, transfer files to the spooling area, display the queue, or remove jobs from the queue. In each case, it forks a child to handle the request so the parent can continue to listen for more requests. The Internet port number used to rendezvous with other processes is normally obtained with *getservent*(3I), but can be changed with the *port #* argument. The **–l** flag causes *lpd* to log valid requests received from the network using *syslog*(3I). This can be useful for debugging purposes.

Access control is provided by two means. First, all requests must come from one of the machines listed in the file **/etc/hosts.equiv** or **/etc/hosts.lpd**. Second, if the "rs" capability is specified in the *printcap* entry for the printer being accessed, *lpr* requests will only be honored for those users who have accounts on the machine to which the printer is attached.

If present, the file **minfree** in each spool directory contains the number of disk blocks to leave free so that the line printer queue won't completely fill the disk. The **minfree** file can be edited with your favorite text editor.

The file **lock** in each spool directory is used to prevent multiple daemons from becoming active simultaneously and to store information about the daemon process for *lpr*(1), *lpq*(1), and *lprm*(1). After the daemon has successfully set the lock, it scans the directory for files beginning with **cf**. Lines in each **cf** file specify files to be printed or non-printing actions to be performed. Each such line begins with a key character to specify what to do with the remainder of the line:

J Job name. String to be used for the job name on the burst page.

C Classification. String to be used for the classification line on the burst page.

L Literal. The line contains identification information from the password file and causes the banner page to be printed.

T Title. String to be used as the title for *pr*(1).

H Host name. Name of the machine where *lpr* was invoked.

P Person. Login name of the person who invoked *lpr*. This is used to verify ownership by *lprm*.

M Send mail to the specified user when the current print job completes.

f Formatted file. Name of a file to print which is already formatted.

l Like "f," but passes control characters and does not make page breaks.

p Name of a file to print using *pr*(1) as a filter.

t *troff* file (old *troff*). The file contains *troff*(1) output (C/A/T phototypesetter commands).

n *ditroff* file (new *troff*). The file contains device-independent *troff* output.

d DVI file. The file contains *tex*(l) output (DVI format from Stanford).

g Graph file. The file contains data produced by *plot*(3X).

c *cifplot* file. The file is assumed to contain data in a CIF format.

v The file contains a raster image.

r The file contains text data with FORTRAN carriage control characters.

1 *troff* font R. Name of the font file to use instead of the default.

2 *troff* font I. Name of the font file to use instead of the default.

3 *troff* font B. Name of the font file to use instead of the default.

4 *troff* font S. Name of the font file to use instead of the default.

W Width. Changes the page width (in characters) used by *pr*(1) and the text filters.

I Indent. The number of characters by which to indent the output (in ASCII).

U Unlink. The name of the file to remove upon completion of printing.

N File name. The name of the file that is being printed, or a blank for the standard input (when *lpr* is invoked in a pipeline).

If a file cannot be opened, a message will be logged via *syslog*(3I) using the LOG_LPR facility. *lpd* will try up to 20 times to reopen a file it expects to be there, after which it will skip the file to be printed.

lpd uses the file locking functions of *fcntl*(2) to provide exclusive access to the lock file and to prevent multiple daemons from becoming active simultaneously. If the daemon should be killed or die unexpectedly, the lock file need not be removed. The lock file is kept in a readable ASCII form and contains two lines. The first line is the process ID of the daemon, and the second line is the control file name of the current job being printed. The second line is updated to reflect the current status of *lpd* for the programs *lpq*(1) and *lprm*(1).

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/usr/spool/*/minfree	minimum free space to leave
/dev/lp*	line printer devices
/dev/printer	socket for local requests
/etc/hosts.equiv	lists machine names allowed printer access
/etc/hosts.lpd	lists machine names allowed printer access, but not under same administrative control

SEE ALSO

lpc(1M), lpq(1), lpr(1), lprm(1), accept(3I), getservent(3I), listen(3I), syslog(3I), printcap(5).
pr(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*
fcntl(2), plot(3X) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*
troff(1) in the *DOCUMENTER'S WORKBENCH Technical Discussion and Reference Manual.*
"Line Printer Spooler Manual."
tex(1) in *The TeX Book*, Donald E. Knuth, Addison-Wesley 1984.

NAME

lpq – spool queue examination program

SYNOPSIS

lpq [**–l**] [**–a**] [**–P**printer] [job # ...] [user ...]

DESCRIPTION

lpq examines the spooling area used by *lpd*(1M) for printing files on the line printer and reports the status of the specified jobs or all jobs associated with a user. *lpq* invoked without any arguments reports on any jobs currently in the queue. A **–P** flag may be used to specify a particular printer; otherwise the default line printer (or the value of the PRINTER variable in the environment) is used. The **–a** flag reports the queue for every printer listed in **/etc/printcap**. All other arguments supplied to these flags are interpreted as user names or job numbers to specify only those jobs of interest.

For each job submitted (i.e., invocation of *lpr*(1)), *lpq* reports the user's name, the current rank in the queue, the names of files comprising the job, the job identifier (a number which may be supplied to *lprm*(1) for removing a specific job), and the total size in bytes. The **–l** option causes information about each of the files comprising the job to be printed. Normally, only as much information as will fit on one line is displayed. Job ordering is dependent on the algorithm used to scan the spooling directory and is supposed to be FIFO (First in First Out). File names comprising a job may be unavailable (when *lpr*(1) is used as a sink in a pipeline), in which case the file is indicated as "standard input."

If *lpq* warns that there is no daemon present (i.e., due to some malfunction), the *lpc*(1M) command can be used to restart the printer daemon.

FILES

/etc/termcap	for manipulating the screen for repeated display
/etc/printcap	to determine printer characteristics
/usr/spool/*	the spooling directory, as determined from printcap
/usr/spool/*/cf*	control files specifying jobs
/usr/spool/*/lock	the lock file to obtain the currently active job

SEE ALSO

lpc(1M), lpd(1M), lpr(1), lprm(1).

"Line Printer Spooler Manual."

DIAGNOSTICS

The system is unable to open various files. The lock file is malformed. Garbled files are created when there is no daemon active, but files are in the spooling directory.

BUGS

Due to the dynamic nature of the information in the spooling directory, *lpq* may report unreliably. Output formatting is sensitive to the line length of the terminal; this can result in widely spaced columns.

NAME

lpr − off-line print

SYNOPSIS

lpr [−Pprinter] [−#num] [−C class] [−J job] [−T title] [−i [**numcols**]] [−**1234** font] [−wnum] [−**pltndgvcfrmhs**] [name ...]

DESCRIPTION

lpr uses a spooling daemon to print the named files when facilities become available. If no names appear, the standard input is assumed. The **−P** option may be used to force output to a specific printer. Normally, the default printer is used (site dependent), or the value of the environment variable PRINTER is used.

The following single-letter options are used to notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

−p Use *pr*(1) to format the files.

−l Use a filter that allows control characters to be printed and suppresses page breaks.

−t The files are assumed to contain data from *troff*(1) (old *troff*; C/A/T phototypesetter commands).

−n The files are assumed to contain data from *ditroff* (new *troff*; device-independent *troff*).

−d The files are assumed to contain data from *tex*(l) (DVI format from Stanford).

−g The files are assumed to contain standard plot data as produced by the *plot*(3X) routines.

−v The files are assumed to contain a raster image for devices like the Benson-Varian.

−c The files are assumed to contain data in a CIF format.

−f Use a filter that interprets the first character of each line as a standard FORTRAN carriage control character.

The remaining single-letter options have the following meanings:

−r Remove the file upon completion of spooling or upon completion of printing (with the **−s** option).

−m Send mail upon completion.

−h Suppress the printing of the burst page.

−s Use symbolic links if available. Usually files are copied to the spool directory.

The **−C** option takes the following argument as a job classification for use on the burst page. For example:

```
lpr −C EECS foo.c
```

causes the system name (the name returned by *hostname*(1)) to be replaced on the burst page by EECS and the file **foo.c** to be printed.

The **−J** option takes its argument as the job name to print on the burst page. Normally, the first file's name is used.

The **−T** option uses its argument as the title used by *pr*(1) instead of the file name.

To get multiple copies of output, use the **−#*num*** option, where *num* is the number of copies desired of each file named. For example,

```
lpr −#3 foo.c bar.c more.c
```

would result in three copies of the file **foo.c**, followed by three copies of the file **bar.c**, etc. On the other hand,

```
cat foo.c bar.c more.c | lpr −#3
```

will give three copies of the concatenation of the files.

The **−i** option causes the output to be indented. If the next argument is numeric, it is used as the number of blanks to be printed before each line; otherwise, eight characters are printed.

The **−w** option takes the immediately following number to be the page width for *pr*.

The **−s** option will use *symlink*(2) if available to link data files rather than trying to copy them so large files can be printed. This means the files should not be modified or removed until they have been printed.

The option **−1234** specifies a font to be mounted on font position *I*. The daemon will construct a **.railmag** file referencing **/usr/lib/vfont/name.size**.

FILES

/etc/passwd	personal identification
/etc/printcap	printer capabilities database
/usr/lib/lpd*	line printer daemons
/usr/spool/*	directories used for spooling
/usr/spool/*/cf*	daemon control files
/usr/spool/*/df*	data files specified in "cf" files
/usr/spool/*/tf*	temporary copies of "cf" files

SEE ALSO

lpc(1M), lpd(1M), lpq(1), lprm(1), printcap(5).
pr(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*
plot(3X) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*
troff(1) in the *DOCUMENTER'S WORKBENCH Technical Discussion and Reference Manual.*
"Line Printer Spooler Manual."
hostname (1) in the *UNIX User's Reference Manual* (4.3BSD).
tex(1) in *The TeX Book*, Donald E. Knuth, Addison-Wesley 1984.

DIAGNOSTICS

If you try to spool too large a file, it will be truncated. *lpr* will object to printing binary files. If a user other than **root** prints a file and spooling is disabled, *lpr* will print a message saying so and will not put jobs in the queue. If a connection to *lpd* on the local machine cannot be made, *lpr* will say that the daemon cannot be started. Diagnostics regarding missing spool files needed by *lpd* may be printed in the daemon's log file.

BUGS

Fonts for *troff* and *tex* reside on the machine to which the printer is attached. It is currently not possible to use local font libraries.

NAME

lprm − remove jobs from the line printer spooling queue

SYNOPSIS

lprm [−**P**printer] [−] [job # ...] [user ...]

DESCRIPTION

lprm removes a job, or jobs, from a printer's spool queue. Since the spooling directory is protected from users, using *lprm* is normally the only method by which a user may remove a job.

lprm without any arguments deletes the currently active job if it is owned by the user who invoked *lprm*.

If the − flag is specified, *lprm* removes all jobs that a user owns. If the superuser employs this flag, the spool queue is emptied entirely. The owner is determined by the user's login name and the host name of the machine where the *lpr* command was invoked.

Specifying a user's name, or list of user names, causes *lprm* to attempt to remove any jobs queued that belong to that user (or users). This form of invoking *lprm* is useful only to the superuser.

A user may remove an individual job from the queue by specifying its job number. This number may be obtained from the *lpq*(1) program. For example:

```
% lpq −l

1st: ken                        [job #013ucbarpa]
    (standard input)            100 bytes
% lprm 13
```

lprm announces the names of any files it removes and is silent if there are no jobs in the queue that match the request list.

lprm kills off an active daemon, if necessary, before removing any spooling files. If a daemon is killed, a new one is automatically restarted upon completion of file removals.

The **−P** option may be used to specify the queue associated with a specific printer; otherwise the default printer (or the value of the PRINTER variable in the environment) is used.

FILES

/etc/printcap	printer characteristics file
/usr/spool/*	spooling directories
/usr/spool/*/lock	lock file used to obtain the pid of the current daemon and the job number of the currently active job

SEE ALSO

lpd(1M), lpq(1), lpr(1).
"Line Printer Spooler Manual."

DIAGNOSTICS

"Permission denied" if the user tries to remove files other than his own.

BUGS

Since there are race conditions possible in the update of the lock file, the currently active job may be incorrectly identified.

NAME

named – Internet domain name server

SYNOPSIS

named [–d debuglevel **] [–p** port # **] [** bootfile **]**

DESCRIPTION

named is the Internet domain name server (see RFC883 for more details). Without any arguments, *named* will read the default *bootfile* **/etc/named.boot**, then read any initial data and listen for queries.

The following options are available:

–d Print debugging information. A number after the "d" determines the level of messages printed.

–p Use a different port number. The default is the standard port number as listed in **/etc/services**.

Any additional argument is taken as the name of the *bootfile*. The *bootfile* contains information about where the name server is to get its initial data. The following is a small example:

```
;
;       boot file for name server
;
; type          domain          source file or host
;
domain          berkeley.edu
primary         berkeley.edu    named.db
secondary       cc.berkeley.edu 10.2.0.78 128.32.0.10
cache           .               named.ca
```

The first line specifies that "berkeley.edu" is the domain for which the server is authoritative. The second line states that the file **named.db** contains authoritative data for the domain "berkeley.edu". The file **named.db** contains data in the master file format described in RFC883 except that all domain names are relative to the origin; in this case, "berkeley.edu" (see below for a more detailed description). The second line specifies that all authoritative data under "cc.berkeley.edu" is to be transferred from the name server at 10.2.0.78. If the transfer fails, it will try 128.32.0.10 and continue trying the address, up to 10, listed on this line. The secondary copy is also authoritative for the specified domain. The fourth line specifies data in **named.ca** is to be placed in the cache (i.e., well-known data such as locations of root domain servers). The file **named.ca** is in the same format as **named.db**.

The master file consists of entries of the form:

```
$INCLUDE <filename>
$ORIGIN <domain>
<domain> <opt_ttl> <opt_class> <type> <resource_record_data>
```

where *domain* is "." for **root**, "@" for the current origin, or a standard domain name. If *domain* is a standard domain name that does not end with ".", the current origin is appended to the domain. Domain names ending with "." are unmodified. The *opt_ttl* field is

an optional integer number for the time-to-live field. It defaults to zero. The *opt_class* field is the object address type; currently only one type is supported, IN, for objects connected to the DARPA Internet. The *type* field is one of the following tokens; the data expected in the *resource_record_data* field is in parentheses.

A	a host address (dotted quad)
NS	an authoritative name server (domain)
MX	a mail exchanger (domain)
CNAME	the canonical name for an alias (domain)
SOA	marks the start of a zone of authority (5 numbers (see RFC883))
MB	a mailbox domain name (domain)
MG	a mail group member (domain)
MR	a mail rename domain name (domain)
NULL	a null resource record (no format or data)
WKS	a well know service description (not implemented yet)
PTR	a domain name pointer (domain)
HINFO	host information (cpu_type OS_type)
MINFO	mailbox or mail list information (request_domain error_domain)

Notes

The following signals have the specified effect when sent to the server process using the *kill*(1) command:

SIGHUP	Causes server to read named.boot and reload database.
SIGINT	Dumps current database and cache to **/usr/tmp/named_dump.db**
SIGUSR1	Turns on debugging; each SIGUSR1 increments debug level.
SIGUSR2	Turns off debugging completely.

FILES

/etc/named.boot	name server configuration boot file
/etc/named.pid	the process id
/usr/tmp/named.run	debug output
/usr/tmp/named_dump.db	dump of the name servers database

SEE ALSO

ghostnamed(3I), resolver(3I), resolver(5).
kill(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.
signal(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.
RFC882, RFC883, RFC973, RFC974.
"Name Server Operations Guide for BIND" in this guide.

NAME

netd – network setup and control daemon

SYNOPSIS

/etc/netd [configfile]

DESCRIPTION

The *netd* daemon is used to set up and control a STREAMS-based network. *netd* reads its default configuration file if one is not specified on the command line and then constructs the network from the configuration. After the network is set up, *netd* waits until it is killed, keeping open the STREAMS file descriptor(s) necessary to hold the network together.

The configuration file follows a simple grammar to specify how the network is constructed. The grammar follows the basic STREAMS terminology for dealing with modules and drivers. The statements supported in the current version are:

> *<variable>* = *<string>* [lsap *<number>* | nsap *<number>* | snap *<number>* *<number>*]
>
> link *<variable>* under *<variable>* [name *<string>*] [type *<number>*]
>
> ifconfig *<string>* [*<address>*] [netmask *<address>*] [broadcast *<address>*] [up | down]

where:

<variable>
: is an identifier consisting of an alphabetic character followed by zero or more alphanumeric characters and including the underscore character.

<string>
: is an arbitrary string of characters bounded by double quote marks.

<number>
: is any decimal or hexadecimal number where hex numbers are prefixed with **0x** or **0X**.

<address>
: is either an internet address in dotted decimal format or a hostname found in the file **/etc/hosts.**

While *netd* is intended for TCP/IP networks, it is relatively general and could be used to set up other types of networks as well.

The keywords have the following meanings:

lsap
: is the *Link Service Access Point.* An *lsap* is either the IEEE 802.2 specified value if the network link layer is utilizing the Logical Link Control protocol or the XEROX Ethernet type value if using the Ethernet link level protocol.

nsap
: is the *Network Service Access Point.* In the case of an IP-based network, the NSAP is the value of the IP protocol identifier.

snap is the Sub-Network Access Point format for LSAPs. The first number is the SAP value to use, and the second number is the organization specific value. For an IEEE 802.3/802.2 network, IP would be represented by *snap* **0xAA 0x800**.

link is an operator specifying how to link two STREAMS together in order to construct the network.

name introduces a string which is used to define the network interface to the IP protocol and may be any unique string.

type introduces a numeric value to be used to inform the upper stream of the type of the lower stream. This value is module specific and is currently only used with the ARP module where 0x0001 specifies an IP stream on Ethernet hardware and 0x0101 represents ARP on Ethernet hardware.

ifconfig is an operator which follows the same conventions that the *ifconfig*(1M) command uses. The interface parameter must be quoted, however.

netmask introduces the network subnet mask if one is desired.

broadcast introduces the value to use as a broadcast address if the default is not desired.

up|*down* are keywords used to specify the state to initialize the interface to.

Sample configuration file:

```
#
# TCP/IP configuration for 3COM ethernet interface
#

# open control streams for transport protocols
tcp = "/dev/tcp"
udp = "/dev/udp"

# open an IP stream for each transport protocol
ip_tcp = "/dev/ip" nsap 6   # bind stream to TCP protocol id
ip_udp = "/dev/ip" nsap 17 # bind stream to UDP protocol id

# open ARP for ethernet use
arp    = "/dev/arp" lsap 0x800

# open link level devices
loop   = "/dev/lo" lsap 0x800
ec_arp = "/dev/ec" lsap 0x806   # stream for ARP messages
ec_ip  = "/dev/ec" lsap 0x800   # stream for IP messages

# put it together
link ip_tcp under tcp
link ip_udp under udp
```

```
link loop under ip_tcp name "lo0"
link arp under ip_tcp name "ec0"
link ec_arp under arp type 0x101  # ARP protocol stream
link ec_ip under arp type 0x001   # IP protocol stream

# initial configuration of interfaces
ifconfig "ec0" queso netmask 255.255.255.0 up
ifconfig "lo0" localhost up
```

SEE ALSO

arp(7), inet(7), ip(7), llc(7), tcp(7).

STREAMS Programmers Guide.

BUGS

Setup is reverse order of declaration, so care must be taken to get ordering correct if there are dependencies.

This is a preliminary implementation and may have a number of changes in later releases.

NAME

netstat – show network status

SYNOPSIS

netstat [**–Aan**] [**–f** address_family] [system] [core]
netstat [**–imnrs** [[**–f** address_family] [system] [core]
netstat [**–n**] [**–I** interface] interval [system] [core]

DESCRIPTION

The *netstat* command symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures, according to the option selected. Using the third form, with an *interval* specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces.

The options have the following meanings:

–A With the default display, show the address of any protocol control blocks associated with sockets; used for debugging.

–a With the default display, show the state of all sockets; normally sockets used by server processes are not shown.

–i Show the state of interfaces that have been configured.

–I *interface*
Show information only about this interface; used with an *interval* as described below.

–m Show statistics recorded by the memory management routines (the network manages a private pool of memory buffers).

–n Show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically). This option may be used with any of the display formats.

–s Show per-protocol statistics.

–r Show the routing tables. When **–s** is also present, show routing statistics instead.

–f *address_family*
Limit statistics or address control block reports to those of the specified *address family*. The following address families are recognized: *inet*, for AF_INET.

The arguments, *system* and *core* allow substitutes for the defaults **/unix** and **/dev/kmem**.

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form *host.port* or *network.port* if a socket's address specifies a network but no specific host address. When known, the host and network addresses are displayed symbolically according to the databases **/etc/hosts** and **/etc/networks**, respectively. If a symbolic name for an address is unknown, or if the **–n** option is specified, the address is printed

numerically, according to the address family. For more information regarding the Internet "dot format," refer to *inet*(3I). Unspecified, or "wildcard," addresses and ports appear as *.

The interface display provides a table of cumulative statistics regarding transferred packets, errors, and collisions. The network addresses of the interface and the maximum transmission unit ("mtu") are also displayed.

The routing table display indicates the available routes and the status of each. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route (**U** if up), whether the route is to a gateway (**G**), and whether the route was created dynamically by a redirect (**D**). Direct routes are created for each interface attached to the local host; the *gateway* field for such entries shows the address of the outgoing interface. The *refcnt* field gives the current number of active uses of the route. Connection-oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The *use* field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column for the primary interface (the first interface found during network setup) and a column summarizing information for all interfaces. The primary interface may be replaced with another interface with the **−I** option. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

The memory usage display provides information on STREAMS resources including buffers, streams and queues. It is organized in six columns with the first being the name of the resource and the remaining five have the following meanings:

alloc how many of the resource were allocated at configuration time.

inuse how many of the resource are in use at the moment.

total how many of the resource have been allocated since boot time.

max the maximum number of the resource that have been allocated at the same time.

fail the number of times an attempt to allocate the resource failed for any reason.

SEE ALSO

trpt(1M), inet(3I), hosts(5), networks(5), protocols(5), services(5).

BUGS

The notion of errors is ill-defined.

Only those network interfaces implementing the DLGSTAT *ioctl* display counts.

NAME

ntp – query an ntp clock

SYNOPSIS

ntp [–v] [–s] [–f] hosts ...

DESCRIPTION

ntp sends an *ntp* packet to the *ntp* daemon running on each of the given hosts. A daemon fills in fields of the *ntp* packet per RFC1119 and sends the packet back. *ntp* then formats and prints the result on the standard output.

The default output shows the delay, offset, and date in *ctime*(3C) format.

The following options can reset the time of the local system clock:

–v Verbose output, showing the full contents of received *ntp* packets, plus caluclated offset, displacement, etc.

–s Set system time-of-day clock. Will only happen if time offset is less than compiled-in constant WAYTOBIG (currently 1000 seconds). Will not happen if remote host is unsynchronized.

–f Force setting system clock regardless of offset. Must be used with **–s** option. Still will not reset clock if remote system is unsynchronized.

ntp Results

The default output for each host looks like this:

```
128.8.10.1: delay:1.845207 offset:-0.358460  Mon Mar 20 08:05:44 1989
```

The verbose output for each host looks like this:

```
Packet from: [128.8.10.1]
Leap 0, version 1, mode Server, poll 6, precision -10 stratum 1 (WWVB)
Synch Distance is 0000.1999  0.099991
Synch Dispersion is 0000.0000  0.000000
Reference Timestamp is a7bea6c3.88b40000 Tue Mar  7 14:06:43 1989
Originate Timestamp is a7bea6d7.d7e6e652 Tue Mar  7 14:07:03 1989
Receive Timestamp is  a7bea6d7.cf1a0000 Tue Mar  7 14:07:03 1989
Transmit Timestamp is a7bea6d8.0ccc0000 Tue Mar  7 14:07:04 1989
Input Timestamp is    a7bea6d8.1a77e5ea Tue Mar  7 14:07:04 1989
umd1: delay:0.019028 offset:-0.043890  Tue Mar  7 14:07:04 1989
```

The various fields are interpreted as follows:

Packet from: [*address*]
: The address from which this *ntp* packet was received.

Leap: *n* The leap second indicator. Nonzero if there is to be a leap second added or subtracted at the new year.

poll = *n*
: The desired poll rate of the peer.

precision = *exponent* (*dec*)
: The precision of this clock, given in seconds as a power of 2, e.g., a clock derived from the power line frequency (60 Hz) has a precision of 1/6 second (about 2^-6) and would be indicated by a precision of -6.

stratum: *n* (*source*)
The stratum of the clock in the *ntp* hierarchy, along with the source of the clock, either the name of a reference standard (such as WWVB or GOES) or the Internet address of the clock from which this clock is derived.

Synch Distance is *nnn* (calculated value).

Synch Dispersion is *nnn* (calculated value).

The next five timestamps are given as *ntp* fixed-point values, in both hexadecimal and *ctime*(3C). These are set either by this *ntp* process or by the server being queried.

Reference Timestamp is *hex-timestamp ctime string*
The last time the server clock was adjusted (remote time).

Originate Timestamp is *hex-timestamp ctime string*
When the *ntp* request was transmitted by us to the server (local time).

Receive Timestamp is *hex-timestamp ctime string*
When the *ntp* request was received at the server (remote time).

Transmit Timestamp is *hex-timestamp ctime string*
When the *ntp* response was transmitted by the server (remote time).

Input Timestamp is *hex-timestamp ctime string*
When the *ntp* response was received (local time).

hostname: delay: *time* offset: *time*
The summary of the results of the query, giving the host name of the responding clock (from the command line), the round-trip delay, and the offset between the two clocks (assuming symmetric round-trip times).

SEE ALSO

ntpd(1M), ntpdc(1M).
ctime(3C) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

RFC1119 Network Time Protocol.

WARNING

ntp(1M) will only work on INTERACTIVE UNIX System Version 3.0 or later.

BUGS

Using *ntp* with the current host will show inaccurate results.

CREDITS

This facility was developed by the University of Maryland.

NAME

ntpd – time synchronization daemon implementing ntp

SYNOPSIS

/etc/ntpd [**–a** threshold] [**–c** file] [**–d**] [**–D** level] [**–l**] [**–n**] [**–s**]

DESCRIPTION

ntpd is the network time synchronization daemon and is normally invoked at boot time from the **/etc/rc3.d/S05hbtcp** file. It implements the revision of the Network Time Protocol described in RFC1119. It maintains the host's time synchronized with a set of distributed time servers, each with varying accuracy and reliability. Multiple time server masters may exist, but there is no requirement for election of a single master.

The *ntpd* daemon uses the *adjtime*(3I) system call to slew the clock of the host by small amount in order to keep the clock synchronized. If the local clock exceeds the "correct" time by some threshold, then *settimeofday*(3I) is used to make a step adjustment of the local clock.

When *ntpd*(1M) is started on the machine, it reads configuration information from **/etc/ntpd.conf**, which contains information about other *ntp* time servers and host specific information. Configuration information is listed one entry per line, with fields separated by white space. Lines that begin with a "#" character are treated as comments. Here is a sample configuration file:

```
#
#                   Local clock parameters
#
#         Precision of the local clock to the nearest power of 2
#                   ex.
#                      60-HZ   = 2**–6
#                      100-HZ  = 2**–7
#                      1000-HZ = 2**–10
precision –7
#
peer      foo.umd.edu
peer      192.5.39.94
peer      bar.arpa
server    bogon.umd.edu
passive   bozo.umd.edu
#
# Configure a reference clock.
#      device        refid      stratum      precision       type
#    ----------      -------    ---------    -----------     -----
peer  /dev/tty03    WWV          1            –5            psti
```

There are two major types of information specified in the configuration file: local host information and remote time server specification. The local host information is used to describe the intrinsic properties of the local host's timekeeping machinery. The commands in this group are *precision* and *tickadj*.

The *precision* command takes a number which describes the resolution of the local clock, as a power of two. For example, a VAX or 386

system typically has a 100 HZ clock and thus a *precision* of −7. If the symbol "hz" is defined in the name list of **/unix**, this value is automatically set based on the value of HZ.

The *tickadj* command is used to specify the granularity of clock adjustment done by the *adjtime*(3I) system call. This should not be done on a 386/486 system.

The *driftfile* command can be used to specify the name of the file that the drift compensation register will be loaded from at initialization time and that updated values will be written into. The drift compensation value describes the intrinsic drift of your host's clock. By default, the file **/etc/ntp.drift** will be used.

Currently three time-server specifications are supported. They are *peer*, *server*, and *passive*. Each command takes either a dotted-quad internet address or a host name. Each host specified in any one of the three commands is eligible to be synchronized to, while random hosts which set up a peer relationship are not. The *peer* and *server* commands create an active polling situation; in the case of *peer*, the *ntp* packets are sourced in Symmetric-Active mode, while using *server* causes the packets to be in Client mode. When reachability is lost with a configured host in either of these two cases, the daemon will continue to poll to re-acquire that host. A host specified in the *passive* command will not continue to be polled. If that host begins to poll us, it will be eligible for synchronization but will not be polled if reachability is lost.

It is recommended that the bulk of the peers configured should be specified with the server keyword; this will minimize resource usage on the remote *ntp* server. If your host will be serving as a redistribution point for a cluster of hosts, you should set up peer relationships with higher quality clocks (lower stratums) and other equal stratum clocks. In other words, if you are not redistributing time to others, you shouldn't need to configure any peers in your *ntp* configuration; client specifications are more appropriate.

To configure a reference clock, you should use something like the example above. The first field after the peer keyword is the name of the file that the clock is connected to. This must be a complete path name with a leading slash (/) character. The next field is the reference ID that will be inserted into the packets generated from this *ntp* daemon. For a PSTI clock, this should be **WWV**. The next field is the stratum of the clock. Actually, it is really the stratum that will be placed in the packet if this clock is selected by the local *ntp* daemon as the reference clock. Following that is the precision that will be inserted into the packet when this clock is selected. The final field is the type of the clock. Currently, two types are supported: **psti** for the Precision Standard Time, Inc. WWV clock (RIP) and **local** for the local time of the system. The local type of clock can be used to declare one host in an isolated network as having the "correct" time and then the other hosts on that network can be synchronized to it.

The reference clock feature is new and will probably be enhanced in the future.

The following options are available:

−a *threshold*
Used to set the threshold which limits how far *ntpd* will change the system clock. It's used as a sort of ultimate sanity check to prevent your system time from being changed a great deal. By default, the *threshold* is 1000 seconds. *threshold* is to be specified in units of seconds, or the string *any* to defeat the sanity check.

−c *file* Used to specify the location of the *ntpd* configuration file. By default **/etc/ntp.conf** is used.

−d Bumps the debug level by one. May be specified more than once to increment debug level by one each time.

−D *level*
Sets the debug level to the value specified.

−l Causes *ntpd* to log a message each time the logical clock is changed. Normally, you would not specify this option unless you wanted to gather statistical information to analyze the logical clock behavior. If the **−l** option is specified, a message will be logged approximately every 2 minutes.

−n Sets the *no swap* option on the process to prevent *ntpd* from being swapped or paged out.

−s Causes *ntpd* to never adjust the local clock.

FILES

/etc/ntp.conf *ntp* daemon configuration file

SEE ALSO

adjtime(3I), settimeofday(3I).
RFC1119 Network Time Protocol.

WARNING

ntpd(1M) will only work on INTERACTIVE UNIX System Version 3.0 or later.

CREDITS

This facility was developed by the University of Maryland.

NAME

ntpdc – monitor operation of ntp daemon

SYNOPSIS

ntpdc [–n] [–v] hosts...

DESCRIPTION

ntpdc sends an INFO_QUERY packet to an *ntp* daemon running on the given hosts. Each daemon responds with information about each of its peers, which *ntpdc* formats on the standard output.

Normally, the name of the responding host and its peers are printed. The **–n** switch disables this, printing only internet addresses.

The default is a terse, table-style report. The **–v** switch generates a verbose report.

Terse Report

A typical terse report looks like this:

(rem) Address	(lcl)	Strat	Poll	Reach	Delay	Offset	Disp
-umd1	128.8.10.14	1	64	266	3.0	-65.0	0.0
*DCN1.ARPA	128.8.10.14	1	256	332	155.0	-4.0	0.0
128.8.251.92	128.8.10.14	2	64	367	-16.0	-61.0	0.0
idunno.Princeto	128.8.10.14	3	64	252	60.0	-53.0	0.0
leo	128.8.10.14	2	64	275	4.0	-273.0	1536.2

Fields are interpreted as follows:

- or *: The - mark indicates a pre-configured peer (mentioned in **ntp.conf**). The * mark shows which pre-configured peer (if any) is currently being used for synchronization.

(rem) Address:
The remote host name or internet address of a peer.

(lcl) Address:
The "local" host as specified as an argument to *ntpdc*.

Strat: The stratum level of the peer (as perceived by the local host).

Poll: Current polling interval in seconds for this peer.

Reach: Octal value of a shift register indicating which responses were received from the previous eight polls to this peer (see RFC1119).

Delay: Round-trip delay in milliseconds for this peer as of the latest poll.

Offset: The amount of time to adjust the local clock to bring it into correspondence with this peer.

Disp: Current value of dispersion (see RFC1119) in milliseconds for this peer.

Verbose Reports

When the **–v** flag is given, a series of verbose reports are presented. A typical one looks like this:

```
Neighbor address 128.4.0.6 port:123  local address 192.35.201.47
Reach: 0376 stratum: 1 poll int(HPI): 10 precision: -10
Sync distance: 0 disp: 0.014000 flags: 0 leap: 0
Reference clock ID: WWV  timestamp: a7c2832e.6f9d0000
Poll int(MPI): 10 threshold: 1024 timer: 1024 send: 266 received: 192 samples: 9
Delay(ms)  1144.00 1296.00 1118.00 1115.00 1225.00 1129.00 1086.00 1087.00
Offset(ms)  19.00  92.00 -17.00  12.00  41.00   4.00  -1.00 -14.00

        delay: 1086.000000 offset: -1.000000 dsp 0.014000
```

Fields are interpreted as follows:

Neighbor address...:
: The address and port number of this neighbor, followed by the local address.

Reach: *nn*
: Reachability in response to last eight polls (octal value of shift register).

stratum: *n*
: Stratum level.

poll int(erval): *time*

precision: *nn*
: The precision of this clock, given in seconds as a power of 2, e.g., a clock derived from the power line frequency (60 Hz) has a precision of 1/60 second (about 2^-6) and would be indicated by a precision of -6.

Sync distance: 0
: Synchronizing distance. Always zero in the current implementation.

disp: *nn*
: Dispersion.

flags: *nn*

leap: *flag*
: The leap second indicator. Nonzero if there is to be a leap second added or subtracted at the new year.

Reference clock ID: [*address*]

timestamp: *n*

Poll int(erval): *time*

threshold: *nn*

timer: *nn*

send: *nn*
: The number of *ntp* packets sent to this neighbor.

received: *nn*
: The number of *ntp* packets received from this neighbor.

samples: *nn*

Delay and Offset:
The round trip delay and clock offset for the last eight *ntp* packet exchanges. If there are fewer than eight valid samples, the delay field will be zero.

delay: *avg-delay* offset: *avg-offset* dsp *nnn* (calculated value)
Average delay, offset, and dispersion calculated from the above samples.

SEE ALSO

ntp(1M), ntpd(1M).
RFC1119 Network Time Protocol.

WARNING

ntpdc(1M) will only work on INTERACTIVE UNIX System Version 3.0 or later.

CREDITS

This facility was developed by the University of Maryland.

NAME

ping – send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

/etc/ping [**–r**] [**–v**] host [packetsize] [count]

DESCRIPTION

The DARPA Internet is a large and complex aggregation of network hardware, connected by gateways. Tracking a single-point hardware or software failure can often be difficult. *ping* utilizes the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams ("pings") have an IP and ICMP header, followed by a **struct** *timeval*, and then an arbitrary number of "pad" bytes used to fill out the packet. Default datagram length is 64 bytes, but this may be changed using the command-line option. Other options are:

–r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly attached network, an error is returned. This option can be used to *ping* a local host through an interface that has no route through it, e.g., after the interface was dropped by *gated*(1M).

–v Verbose output. ICMP packets other than ECHO_RESPONSE that are received are listed.

When using *ping* for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be "pinged." *ping* sends one datagram per second, and prints one line of output for every ECHO_RESPONSE returned. No output is produced if there is no response. If an optional *count* is given, only that number of requests is sent. Round-trip times and packet-loss statistics are computed. When all responses have been received or the program times out (with a *count* specified), or if the program is terminated with a SIGINT, a brief summary is displayed.

This program is intended for use in network testing, measurement and management. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is unwise to use *ping* during normal operations or from automated scripts.

SEE ALSO

ifconfig(1M), netstat(1).

NAME

rarpd – reverse ARP daemon

SYNOPSIS

/etc/rarpd [–c] [–l] [–s lsap **]**
[device [hostname]] [device [hostname]]

DESCRIPTION

rarpd reads messages from a set of network interfaces and responds to those reverse address resolution protocol (RARP) requests for which information on the hardware address to physical address translation is known. The address translation information is stored in **/etc/ethers**. If any device names are specified on the command line, *rarpd* will use those network interfaces; otherwise it will use all network interfaces which support the ARP capability. If the *hostname* is not specified for a particular interface, *rarpd* will determine the appropriate IP address using the SIOCGIFCONF *ioctl*.

The **–c** option may be used to specify an alternate configuration file.

The **–l** option enables the logging of all requests to *syslog*. Normally, only error conditions are logged.

The **–s** option allows the specification of an alternate service access point (SAP) or type field value than the default of 0x8035.

The configuration file contains lines of two types: comment lines and physical address to internet mapping. Comment lines begin with **#** and are ignored by *rarpd* . Mapping lines contain an internet address and physical address pair.

Internet addresses may be specified in either standard dot notation or as a host name. The host name form is preferred, as this allows using the name server (*named*(1M)) to determine IP addresses dynamically.

Physical addresses are specified as a string of hexadecimal digits. Each octet of the address should be separated from the others with a colon (**:**).

```
# physical          internet
00:00:c0:f7:30:10   128.212.32.200
00:00:0c:06:77:24   128.212.32.201
00:00:a7:00:06:8b   xterminal
00:00:c0:3c:ca:19   teresh
```

SEE ALSO

named(1M).
RFC903.

NAME

rcp − remote file copy

SYNOPSIS

rcp [−p] file1 file2
rcp [−p] [−r] file ... directory

DESCRIPTION

rcp copies files between machines. Each *file* or *directory* argument is either a remote file name of the form *rhost:path,* or a local file name (containing no : characters, or a / before any :s).

If the **−r** option is specified and any of the source files are directories, *rcp* copies each subtree rooted at that name; in this case the destination must be a directory.

By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the *umask*(2) on the destination host is used. The **−p** option causes *rcp* to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the *umask*.

If *path* is not a full path name, it is interpreted relative to your login directory on *rhost*. A *path* on a remote host may be quoted (using \, ', or ") so that the metacharacters are interpreted remotely.

rcp does not prompt for passwords; your current local user name must exist on *rhost* and allow remote command execution via *rsh*(1C).

rcp handles third-party copies, where neither source nor target files are on the current machine. Host names may also take the form *rname***@***rhost* to use *rname* rather than the current user name on the remote host. The destination host name may also take the form *rhost.rname* to support destination machines that are running 4.2 BSD versions of *rcp*. Use of *rname* follows the same rules for a **.rhosts** file that *rlogin* and *rsh* follow.

SEE ALSO

ftp(1C), rlogin(1C), rsh(1C).
cp(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.

BUGS

Does not detect all cases where the target of a copy might be a file in cases where only a directory should be legal.
Is confused by any output generated by commands in a **.login**, **.profile**, or **.cshrc** file on the remote host.

NAME

rexecd – remote execution server

SYNOPSIS

/etc/rexecd

DESCRIPTION

rexecd is the server for the *rexec*(3I) routine. The server provides remote execution facilities with authentication based on user names and passwords.

rexecd listens for service requests at the port indicated in the "exec" service specification; see *services*(5). When a service is received, the following protocol is initiated:

1) The server reads characters from the socket up to a null byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is nonzero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine.

3) A null-terminated user name of at most 16 characters is retrieved on the initial socket.

4) A null-terminated, encrypted password of at most 16 characters is retrieved on the initial socket.

5) A null-terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

6) *rexecd* then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail, the connection is aborted and a diagnostic message is returned.

7) A null byte is returned on the initial socket, and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rexecd*.

SEE ALSO

rexec(3I), services(5).

DIAGNOSTICS

Except for the last one listed below, all diagnostic messages are returned on the initial socket, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

username too long
The name is longer than 16 characters.

password too long
The passwd is longer than 16 characters.

command too long
The command line passed exceeds the size of the argument list (as configured into the system).

Login incorrect
No password file entry for the user name existed.

Password incorrect
The wrong password was supplied.

No remote directory
The *chdir* command to the home directory failed.

TRY again
A *fork* by the server failed.

<shellname>:...
The user's login shell could not be started. This message is returned on the connection associated with the **stderr** and is not preceded by a flag byte.

BUGS

Indicating "Login incorrect" as opposed to "Password incorrect" is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data and password exchanges to be encrypted should be present.

NAME

rlogin – remote login

SYNOPSIS

rlogin rhost [**–ec**] [**–8**] [**–l** username]
rhost [**–ec**] [**–8**] [**–l** username]

DESCRIPTION

rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost*.

Each host has a file, **/etc/hosts.equiv**, that contains a list of *rhosts* with which it shares account names. (The host names must be the standard names as described in *rsh*(1C).) When you *rlogin* as the same user on an equivalent host, you do not need to give a password. Each user may also have a private equivalence list in a file **.rhosts** in his login directory. Each line in this file should contain an *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in *login*(1). To avoid some security problems, the **.rhosts** file must be owned by either the remote user or **root**.

The remote terminal type is the same as your local terminal type (as given in your environment TERM variable). The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. All echoing takes place at the remote site, so that (except for delays) the *rlogin* is transparent. Flow control via **^S** and **^Q** and flushing of input and output on interrupts are handled properly. The optional argument **–8** allows an eight-bit input data path at all times; otherwise parity bits are stripped except when the remote side's stop and start characters are other than **^S**/**^Q**. A line of the form **~.** disconnects from the remote host, where **~** is the escape character. A different escape character may be specified by the **–e** option. There is no space separating this option flag and the argument character.

SEE ALSO

rsh(1C).

FILES

/usr/hosts/* for *rhost* version of the command

BUGS

More of the environment should be propagated.

NAME

rlogind – remote login server

SYNOPSIS

/etc/rlogind [–d]

DESCRIPTION

rlogind is the server for the *rlogin*(1C) program. The server provides a remote login facility with authentication based on privileged port numbers from trusted hosts.

rlogind listens for service requests at the port indicated in the "login" service specification; see *services*(5). When a service request is received, the following protocol is initiated:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection.

2) The server checks the client's source address and requests the corresponding host name (see *gethostbyaddr* in *ghostnamed*(3I), *hosts*(5), and *named*(1M)). If the host name cannot be determined, the dot-notation representation of the host address is used.

Once the source port and address have been checked, *rlogind* allocates a pseudo terminal (see *pty*(7)) and manipulates file descriptors so that the slave half of the pseudo terminal becomes the *stdin*, *stdout*, and *stderr* for a login process. The login process is an instance of the *login*(1) program. The login process then proceeds with the authentication process as described in *rshd*(1M), but if automatic authentication fails, it reprompts the user to log in as one finds on a standard terminal line.

The parent of the login process manipulates the master side of the pseudo terminal, operating as an intermediary between the login process and the client instance of the *rlogin* program. In normal operation, the packet protocol described in *pty*(7) is invoked to provide ^S/^Q type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, TERM; see *environ*(5). The screen or window size of the terminal is requested from the client, and window size changes from the client are propagated to the pseudo terminal.

SEE ALSO

named(1M), rlogin(1C), rshd(1M), ghostnamed(3I), hosts(5), services(5), pty(7).
environ(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the *stderr*, after which any network connections are closed. An error is indicated by a leading byte with a value of 1.

Try again.
A *fork* by the server failed.

/bin/sh: ...
The user's login shell could not be started.

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is not secure, but it is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

A more extensible protocol should be used.

NAME

route – manually manipulate the routing tables

SYNOPSIS

/etc/route [**–f**] [**–n**] [command args]

DESCRIPTION

route is a program used to manually manipulate the network routing tables. It normally is not needed, as the system routing table management daemon, *gated*(1M), should tend to this task.

route accepts two commands: **add**, to add a route, and **delete**, to delete a route.

All commands have the following syntax:

/etc/route *command* [**net** | **host**] *destination gateway* [*metric*]

where *destination* is the destination host or network, *gateway* is the next-hop gateway to which packets should be addressed, and *metric* is a count indicating the number of hops to the *destination*. The metric is required for **add** commands; it must be zero if the destination is on a directly attached network, and nonzero if the route utilizes one or more gateways. If adding a route with metric 0, the gateway given is the address of this host on the common network, indicating the interface to be used for transmission. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. The optional keywords **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, 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. If the route is to a destination connected via a gateway, the *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first as a host name using *gethostbyname* (see *ghostnamed*(3I)). If this lookup fails, *getnetbyname* (see *getnetent*(3I)) is then used to interpret the name as that of a network.

route uses a raw socket and the SIOCADDRT and SIOCDELRT *ioctl*s to do its work. As such, only the superuser may modify the routing tables.

If the **–f** option is specified, *route* will "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

The **–n** option prevents attempts to print host and network names symbolically when reporting actions.

SEE ALSO

gated(1M), getnetent(3I), ghostnamed(3I), intro(7N).

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* call. If the gateway address used was not the primary address of the gateway (the first

one returned by *gethostbyname*), 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 **−f** flag 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 that 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.

NAME

rsh − remote shell

SYNOPSIS

rsh host [**−l** username] [**−n**] command
host [**−l** username] [**−n**]

DESCRIPTION

rsh connects to the specified *host* and executes the specified *command*. *rsh* copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit, and terminate signals are propagated to the remote command; *rsh* normally terminates when the remote command does.

The remote user name used is the same as your local user name, unless you specify a different remote name with the **−l** option. This remote name must be equivalent (in the sense of *rlogin*(1C)) to the originating account; no provision is made for specifying a password with a command.

If you omit *command*, then instead of executing a single command, you will be logged in on the remote host using *rlogin*(1C).

Shell metacharacters that are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command:

rsh otherhost cat remotefile >> localfile

appends the remote file **remotefile** to the localfile **localfile**, while:

rsh otherhost cat remotefile ">>" otherremotefile

appends **remotefile** to **otherremotefile**.

Host names are given in the file **/etc/hosts**. Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames. The host names for local machines are also commands in the directory **/usr/hosts**; if you put this directory in your search path, then the **rsh** can be omitted.

rsh may be linked to the name *rshell* or *remsh* to avoid a name conflict with **/bin/rsh**. **/usr/ucb/rsh** must still exist.

FILES

/etc/hosts
/usr/hosts/*

SEE ALSO

rlogin(1C).

BUGS

If you are using *csh*(1) and put a *rsh*(1C) in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. If no input is desired, you should redirect the input of *rsh* to **/dev/null** using the **−n** option, which redirects the input of rsh to **/dev/null**. This can prevent unwanted interaction with the shell used to invoke the *rsh* command.

You cannot run an interactive command such as *vi*(1); if you must run interactive commands, use *rlogin*(1C) instead.

NAME

rshd – remote shell server

SYNOPSIS

/etc/rshd

DESCRIPTION

rshd is the server for the *rcmd*(3I) routine and, consequently, for the *rsh*(1C) program. The server provides remote execution facilities with authentication based on privileged port numbers from trusted hosts.

rshd listens for service requests at the port indicated in the "cmd" service specification; see *services*(5). When a service request is received the following protocol is initiated:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection.

2) The server reads characters from the socket up to a null (**\0**) byte. The resultant string is interpreted as an ASCII number, base 10.

3) If the number received in step 1 is nonzero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 0-1023.

4) The server checks the client's source address and requests the corresponding host name (see *gethostbyaddr* (*ghostnamed*(3I)), *hosts*(5), and *named*(1M)). If the hostname cannot be determined, the dot-notation representation of the host address is used.

5) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the **client**'s machine.

6) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the **server**'s machine.

7) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

8) *rshd* then validates the user according to the following steps. The local (server-end) user name is looked up in the password file and a *chdir* is performed to the user's home directory. If lookup fails, the connection is terminated. If a *chdir* to the user's home directory fails, *chdir* to the root directory (/) will be performed and execution will continue. If the user is not the superuser (user-ID 0), the file **/etc/hosts.equiv** is consulted for a list of hosts considered "equivalent." If the client's host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the superuser, then the file **.rhosts** in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. If this lookup fails, the connection is terminated.

9) A null byte is returned on the initial socket and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rshd*.

SEE ALSO

named(1M), rsh(1C), ghostnamed(3I), rcmd(3I), hosts(5), services(5).

DIAGNOSTICS

Except for the last one listed below, all diagnostic messages are returned on the initial socket, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the execution of the login shell).

locuser too long
The name of the user on the client's machine is longer than 16 characters.

remuser too long
The name of the user on the remote machine is longer than 16 characters.

command too long
The command line passed exceeds the size of the argument list (as configured into the system).

Login incorrect.
No password file entry for the user name existed.

Permission denied.
The authentication procedure described above failed.

Can't make pipe.
The pipe needed for the **stderr** wasn't created.

Try again.
A *fork* by the server failed.

<shellname>: ...
The user's login shell could not be started. This message is returned on the connection associated with the **stderr**, and is not preceded by a flag byte.

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is not secure, but it is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

A more extensible protocol should be used.

NAME

ruptime – show host status of local machines

SYNOPSIS

ruptime [**–a**] [**–r**] [**–l**] [**–t**] [**–u**]

DESCRIPTION

ruptime gives a status line like *uptime* for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 11 minutes are shown as being down.

Users idle an hour or more are not counted unless the **–a** flag is given.

Normally, the listing is sorted by host name. The **–l**, **–t**, and **–u** flags specify sorting by load average, uptime, and number of users, respectively. The **–r** flag reverses the sort order.

FILES

/usr/spool/rwho/whod.* data files

SEE ALSO

rwho(1C).

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [–a]

DESCRIPTION

The *rwho* command produces output similar to *who*(1), but for all machines on the local network. If no report has been received from a machine for 11 minutes then *rwho* assumes the machine is down, and does not report users last known to be logged into that machine.

If a users hasn't typed to the system for 1 minute or more, then *rwho* reports this idle time. If a user hasn't typed to the system for an hour or more, then the user will be omitted from the output of *rwho* unless the **–a** flag is given.

FILES

/usr/spool/rwho/whod.* information about other machines

SEE ALSO

ruptime(1C), rwhod(1M).

BUGS

This is unwieldy when the number of machines on the local net is large.

NAME

rwhod – system status server

SYNOPSIS

/etc/rwhod

DESCRIPTION

rwhod is the server that maintains the database used by the *rwho*(1C) and *ruptime*(1C) programs. Its operation is predicated on the ability to broadcast messages on a network.

rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages that are broadcast on a network. As a consumer of information, it listens for other *rwhod* servers' status messages, validating them, then recording them in a collection of files located in the directory **/usr/spool/rwho**.

The server transmits and receives messages at the port indicated in the *rwho* service specification; see *services*(5). The messages sent and received, are of the form:

```
struct  outmp {
    char    out_line[8];        /* tty name */
    char    out_name[8];        /* user-ID*/
    long    out_time;           /* time on */
};

struct  whod {
    char    wd_vers;
    char    wd_type;
    char    wd_pad[2];
    int     wd_sendtime;
    int     wd_recvtime;
    char    wd_hostname[32];
    int     wd_loadav[3];
    int     wd_boottime;
    struct  whoent {
        struct  outmp we_utmp;
        int     we_idle;
    } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission; they are multiplied by 100 for representation in an integer. The host name included is that returned by *gethostname*(3I), with any trailing domain name omitted. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the *utmp*(4) entry for each non-idle terminal line and a value indicating the time in seconds since a character was last received on the terminal line.

Messages received by the *rwho* server are discarded unless they originated at an *rwho* server's port. In addition, if the host name, as specified in the message, contains any unprintable ASCII characters,

the message is discarded. Valid messages received by *rwhod* are placed in files named **whod.***hostname* in the directory **/usr/spool/rwho**. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 3 minutes. *rwhod* performs an *nlist*(3) on **/unix** every 30 minutes to guard against the possibility that this file is not the system image currently operating.

SEE ALSO

ruptime(1C), rwho(1C), services(5).

BUGS

There should be a way to relay status information between networks. Status information should be sent only upon request, rather than continuously. People often interpret the server dying or network communication failures as a machine going down.

NAME

sendmail – send mail over the internet

SYNOPSIS

/usr/lib/sendmail [flags] [address ...]

newaliases

mailq [**–v**]

DESCRIPTION

sendmail sends a message to one or more *recipients*, routing the message over whatever networks are necessary. *sendmail* does internetwork forwarding as necessary to deliver the message to the correct place.

sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; *sendmail* is used only to deliver preformatted messages.

With no flags, *sendmail* reads its standard input up to an end-of-file or a line consisting only of a single dot and sends a copy of the message found there to all of the addresses listed. It determines the network(s) to use based on the syntax and contents of the addresses.

Local addresses are looked up in a file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in any alias expansions, e.g., if **john** sends to **group**, and **group** includes **john** in the expansion, then the letter will not be delivered to **john**.

Flags are:

–ba Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the **From:** and **Sender:** fields are examined for the name of the sender.

–bd Run as a daemon. This requires Berkeley IPC. *sendmail* will fork and run in background listening on socket 25 for incoming SMTP connections. This is normally run from **/etc/rc**.

–bi Initialize the alias database.

–bm Deliver mail in the usual way (default).

–bp Print a listing of the queue.

–bs Use the SMTP protocol as described in RFC821 on standard input and output. This flag implies all the operations of the **–ba** flag that are compatible with SMTP.

–bt Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.

–bv Verify names only – do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.

−bz Create the configuration freeze file.

−C*file* Use alternate configuration file. *sendmail* refuses to run as **root** if an alternate configuration file is specified. The frozen configuration file is bypassed.

−d*X* Set debugging value to *X*.

−F*fullname* Set the full name of the sender.

−f*name* Sets the name of the **From:** person, i.e., the sender of the mail. **−f** can only be used by "trusted" users (normally **root**, **daemon**, and **network**) or if the person you are trying to become is the same as the person you are.

−h*N* Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop. If not specified, **Received:** lines in the message are counted.

−n Do not do aliasing.

−o*x value* Set option *x* to the specified *value*. Options are described below.

−q[*time***]** Processed saved messages in the queue at given intervals. If *time* is omitted, process the queue once. *time* is given as a tagged number, with *s* being seconds, *m* being minutes, *h* being hours, *d* being days, and *w* being weeks. For example, **−q1h30m** or **−q90m** would both set the timeout to one hour thirty minutes. If *time* is specified, *sendmail* will run in background. This option can be used safely with **−bd**.

−r*name* An alternate and obsolete form of the **−f** flag.

−t Read message for recipients. **To:**, **Cc:**, and **Bcc:** lines will be scanned for recipient addresses. The **Bcc:** line will be deleted before transmission. Any addresses in the argument list will be suppressed, that is, they will *not* receive copies even if listed in the message header.

−v Go into verbose mode. Alias expansions will be announced, etc.

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **−o** flag or in the configuration file. These are described in detail in the "Sendmail Installation and Operation Guide." The options are:

A*file* Use alternate alias file.

c	On mailers that are considered "expensive" to connect to, do not initiate immediate connection. This requires queuing.
d*x*	Set the delivery mode to *x*. Delivery modes are **i** for interactive (synchronous) delivery, **b** for background (asynchronous) delivery, and **q** for queue only, i.e., actual delivery is done the next time the queue is run.
D	Try to automatically rebuild the alias database if necessary.
e*x*	Set error processing to mode *x*. Valid modes are **m** to mail back the error message, **w** to "write" back the error message (or mail it back if the sender is not logged in), **p** to print the errors on the terminal (default), **q** to throw away error messages (only exit status is returned), and **e** to do special processing for the BerkNet. If the text of the message is not mailed back by modes **m** or **w** and if the sender is local to this machine, a copy of the message is appended to the file **dead.letter** in the sender's home directory.
E	Enable EXPN command.
F*mode*	The mode to use when creating temporary files.
f	Save UNIX System-style **From:** lines at the front of messages.
g*N*	The default group-ID to use when calling mailers.
H*file*	The SMTP help file.
i	Do not take dots on a line by themselves as a message terminator.
L*n*	The log level.
m	Send to "me" (the sender) also if I am in an alias expansion.
o	If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers, i.e., commas instead of spaces between addresses. If set, an adaptive algorithm is used that will correctly determine the header format in most cases.
Q*queuedir*	Select the directory in which to queue messages.
r*timeout*	The timeout on reads; if none is set, *sendmail* will wait forever for a mailer. This option violates the word (if not the intent) of the SMTP specification, show the timeout should probably be fairly large.
S*file*	Save statistics in the named file.

s	Always instantiate the queue file, even under circumstances where it is not strictly necessary. This provides safety against system crashes during delivery.
T*time*	Set the timeout on undelivered messages in the queue to the specified time. After delivery has failed (e.g., because of a host being down) for this amount of time, failed messages will be returned to the sender. The default is 3 days.
t*stz,dtz*	Set the name of the time zone.
u*N*	Set the default user-ID for mailers.

In aliases, the first character of a name may be a vertical bar to cause interpretation of the rest of the name as a command to pipe the mail to. It may be necessary to quote the name to keep *sendmail* from suppressing the blanks from between arguments. For example, a common alias is:

msgs: "|/usr/ucb/msgs −s"

Aliases may also have the syntax **:include: filename** to ask *sendmail* to read the named file for a list of recipients. For example, an alias such as:

poets: ":include:/usr/local/lib/poets.list"

would read **/usr/local/lib/poets.list** for the list of addresses making up the group.

sendmail returns an exit status describing what it did. The codes are defined in **<sysexits.h>**:

EX_OK	Successful completion on all addresses.
EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as cannot fork.
EX_NOHOST	Host name not recognized.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

If invoked as *newaliases*, *sendmail* will rebuild the alias database. If invoked as *mailq*, *sendmail* will print the contents of the mail queue.

FILES

Except for **/usr/lib/sendmail.cf**, these path names are all specified in **/usr/lib/sendmail.cf**. Thus, these values are only approximations.

/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.pag	
/usr/lib/aliases.dir	database of alias names
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	help file

/usr/lib/sendmail.st collected statistics
/usr/spool/mqueue/* temp files

SEE ALSO

syslog(3I).
mail(1), mailx(1), rmail(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.
aliases(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

DARPA Internet Request For Comments RFC819, RFC821, RFC822.

"Sendmail – An Internetwork Mail Router" and "Sendmail Installation and Operation Guide" in the *INTERACTIVE UNIX Operating System Guide*.

NAME

slattach – attach serial lines as network interfaces

SYNOPSIS

/usr/ucb/slattach ttyname [baudrate]

DESCRIPTION

The command *slattach* is used to assign a terminal line to a network interface and to define the network source and destination addresses. The *ttyname* parameter is a string of the form **tty***xx* or **/dev/tty***xx*. The optional *baudrate* parameter is used to set the speed of the connection. If not specified, it assumes a default value of 9600.

Only **root** (the superuser) may attach a network interface.

To detach the interface, use:

ifconfig <interface> down

after killing the *slattach* process. The name <*interface*> is the one shown by *netstat*(1).

Examples

```
/usr/ucb/slattach       ttyh8
/usr/ucb/slattach       /dev/tty01  4800
```

SEE ALSO

ifconfig(1M), netstat(1).

DIAGNOSTICS

Diagnostics include 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.

NAME

sldialup – call another UNIX System and connect via SLIP

SYNOPSIS

/usr/ucb/sldialup ttyname [baudrate]

DESCRIPTION

The command *sldialup* interactively calls another system running SLIP and connects a network interface to it. The *ttyname* parameter is a string of the form **tty***xx* or **/dev/tty***xx*. The optional *baudrate* parameter is used to specify the speed of the modem connected to the terminal line. If not specified, it defaults to a value of 9600 baud.

When you first invoke *sldialup*, you will be talking directly to the modem connected to the port. You must give the commands to the modem to dial the system to which you want to connect and then log in to an account that has *sllogin*(1M) as its shell. When connected, press the **ESC** key to connect the port to the SLIP network interface. Type **CTRL c** to hang up the line and abort the call.

To detach the interface, kill the forked *sllogin* process (this will also hang up the modem). Then type:

ifconfig <interface> down

where <*interface*> is the name shown by the *netstat* **–i** command.

Example

To connect to **/dev/tty00**, type:

sldialup tty00 2400

Give the commands to the modem to dial the target host (press **ESC** when connected or **CTRL c** to abort):

ATDT 5551212
CONNECT 2400

Login: slip
Password:

UNIX System V/386 Release 2.0 80386
UNIX
Copyright (c) 1987 AT&T
All rights reserved

Connecting remote to SLIP interface.
Give command to connect your local host.
<Escape>

SEE ALSO

slattach(1M), sllogin(1M), sl(7).

NAME

sllogin – SLIP login shell

SYNOPSIS

/usr/ucb/sllogin

DESCRIPTION

The utility *sllogin* enables incoming Serial Line Internet Protocol (SLIP) network connections via a modem. This utility is generally used with *sldialup*(1M). To create a dialin SLIP account, add an account in the usual manner. Then edit the **/etc/passwd** file and replace the normal shell with **/usr/ucb/sllogin**.

This account should be used only on normal serial ports and *not* on the console.

SEE ALSO

slattach(1M), sldialup(1M), sl(7).

NAME

syslogd – log system messages

SYNOPSIS

etc/syslogd [–fconfigfile **] [–m**markinterval **] [–d]**

DESCRIPTION

syslogd reads and logs messages into a set of files described by the configuration file, **/etc/syslog.conf**. Each message is one line. A message can contain a priority code, marked by a number in angle braces at the beginning of the line. Priorities are defined in **<sys/syslog.h>**. *syslogd* reads from an Internet domain socket specified in **/etc/services**, and from the special device **/dev/log** (to read kernel messages).

syslogd configures when it starts up and whenever it receives a hang up signal. Lines in the configuration file have a *selector*, to determine the message priorities to which the line applies, and an *action*. The *action* fields are separated from the selector by one or more tabs.

Selectors are semicolon-separated lists of priority specifiers. Each priority has a *facility* describing the part of the system that generated the message, a dot, and a *level* indicating the severity of the message. Symbolic names may be used. An asterisk selects all facilities. All messages of the specified level or higher (greater severity) are selected. More than one facility may be selected, using commas to separate them. For example:

*.emerg;mail,daemon.crit

selects all facilities at the *emerg* level and the *mail* and *daemon* facilities at the *crit* level.

Known facilities and levels recognized by *syslogd* are those listed in *syslog*(3) without the leading "LOG_". The additional facility "mark" has a message at priority LOG_INFO sent to it every 20 minutes (this may be changed with the **–m** flag). The "mark" facility is not enabled by a facility field containing an asterisk. The level "none" may be used to disable a particular facility. For example:

*.debug;mail.none

sends all messages *except* mail messages to the selected file.

The second part of each line describes where the message is to be logged if this line is selected. There are four forms:

- A file name (beginning with a leading slash). The file will be opened in append mode.
- A hostname preceded by an "at" sign (@). Selected messages are forwarded to the *syslogd* on the named host.
- A comma-separated list of users. Selected messages are written to those users if they are logged in.
- An asterisk. Selected messages are written to all logged-in users.

Blank lines and lines beginning with '#' are ignored.

For example, the configuration file:

```
kern,mark.debug          /dev/console
*.notice;mail.info       /usr/spool/adm/syslog
*.crit                   /usr/adm/critical
kern.err                 @ucbarpa
*.emerg                  *
*.alert                  eric,kridle
*.alert;auth.warning     ralph
```

logs all kernel messages and 20-minute marks onto the system console, all notice (or higher-level) messages and all mail system messages except debug messages into the file **/usr/spool/adm/syslog**, and all critical messages into **/usr/adm/critical**; kernel messages of error severity or higher are forwarded to **ucbarpa**. All users will be informed of any emergency messages, the users "eric" and "kridle" will be informed of any alert messages, and the user "ralph" will be informed of any alert message, or any warning (or higher-level) message from the authorization system.

The flags are:

−f Specify an alternate configuration file.

−m Select the number of minutes between mark messages.

−d Turn on debugging.

syslogd creates the file **/etc/syslog.pid**, if possible, which contains a single line with its process ID. This can be used to *kill* or reconfigure *syslogd*.

To bring *syslogd* down, it should be sent a terminate signal, e.g., *kill* `*cat* **/etc/syslog.pid**`.

FILES

/etc/syslog.conf	the configuration file
/etc/syslog.pid	the process ID
/dev/klog	the kernel log device

SEE ALSO

logger(1C), syslog(3I).

NAME

talk – talk to another user

SYNOPSIS

talk person [ttyname]

DESCRIPTION

talk is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on your own machine, then *person* is simply the person's login name. If you wish to talk to a user on another host, then *person* takes one of the following forms:

host!user
host.user
host:user
user@host

though *user@host* is perhaps preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, *talk* sends the message:

> Message from TalkDaemon@his_machine...
> talk: connection requested by your_name@your_machine.
> talk: respond with: talk your_name@your_machine

It doesn't matter from which machine the recipient replies, as long as his login name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing **CTRL l** causes the screen to be reprinted, while the "erase," "kill," and "word kill" characters will operate normally. To exit, simply type the "interrupt" character; *talk* then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the *mesg* command. At the outset, talking is allowed. Certain commands, in particular *nroff* and *pr*(1), disallow messages in order to prevent messy output.

FILES

/etc/hosts to find the recipient's machine
/etc/utmp to find the recipient's tty

SEE ALSO

mail(1), mesg(1), who(1), write(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.

BUGS

The version of *talk*(1) released with 4.3BSD uses a protocol that is incompatible with the protocol used in the version released with 4.2BSD.

NAME

talkd – remote user communication server

SYNOPSIS

/etc/talkd

DESCRIPTION

talkd is the server that notifies a user that another user wants to initiate a conversation. It acts as a repository for invitations and responds to requests by client programs wishing to rendezvous by sending a CTL_MSG to the server of type LOOK_UP (see **<protocols/talkd.h>**). This causes the server to search its invitation tables to check if an invitation currently exists for the caller to speak to the respondent specified in the message. If the lookup fails, the caller then sends an ANNOUNCE message, causing the server to broadcast an announcement on the respondent's login ports requesting contact. When the respondent responds, the local server uses the recorded invitation to respond with the appropriate rendezvous address and the caller and respondent client programs establish a STREAMS connection through which the conversation takes place.

SEE ALSO

talk(1C).

write(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.

NAME

tdate – print and set the date

SYNOPSIS

date [+ format]
date [mmddhhmm [[yy] [ccyy]]]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set (only by the superuser). The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24-hour system); the second *mm* is the minute number; *cc* is the century minus one and is optional; *yy* is the last two digits of the year number and is optional. For example:

date 10080045

sets the date to October 8, 12:45 a.m. The current year is the default if no year is mentioned. The system operates in Universal Coordinated Time (UTC). *tdate*(1) takes care of the conversion to and from local standard and daylight saving time. Only the superuser may change the date.

If the argument begins with +, the output of *tdate* is under the control of the user. All output fields are of fixed size (zero-padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character. If the argument contains embedded blanks, it must be quoted (see the **EXAMPLE** section).

Specifications of native language translations of month and weekday names are supported. The language used depends on the value of the environment variable LANGUAGE (see *environ*(5)). The month and weekday names used for a language are taken from strings in the file for that language in the **/lib/cftime** directory (see *cftime*(4)).

After successfully setting the date and time, *tdate* will display the new date according to the format defined in the environment variable CFTIME (see *environ*(5)).

Field Descriptors (must be preceded by a %):

- **a** abbreviated weekday name
- **A** full weekday name
- **b** abbreviated month name
- **B** full month name
- **d** day of month – 01 to 31
- **D** date as mm/dd/yy
- **e** day of month – 1 to 31 (single digits are preceded by a blank)
- **h** abbreviated month name (alias for %b)
- **H** hour – 00 to 23
- **I** hour – 01 to 12
- **j** day of year – 001 to 366
- **m** month of year – 01 to 12

M	minute – 00 to 59
n	insert a newline character
p	string containing ante-meridiem or post-meridiem indicator (by default, a.m. or p.m.)
r	time as *hh:mm:ss pp* where *pp* is the ante-meridiem or post-meridiem indicator (by default, a.m. or p.m.)
R	time as hh:mm
S	second – 00 to 59
t	insert a tab character
T	time as *hh:mm:ss*
U	week number of year (Sunday as the first day of the week) – 01 to 52
w	day of week – Sunday = 0
W	week number of year (Monday as the first day of the week) – 01 to 52
x	country-specific date format
X	country-specific time format
y	year within century – 00 to 99
Y	year as *ccyy* (4 digits)
Z	timezone name

If *timed*(1) is running to synchronize the clocks of machines in a local area network, *tdate* sets the time globally on all those machines.

EXAMPLE

```
date '+DATE: %m/%d/%y%nTIME: %H;%M:%S'
```

would have generated as output:

```
DATE: 08/01/76  TIME: 14:45:05
```

Note

Administrators should note the following: if you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight savings time is starting or ending), and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

FILES

/dev/kmem

SEE ALSO

cftime(4), environ(5) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

DIAGNOSTICS

No permission	if you are not the superuser and you try to change the date
bad conversion	if the date set is syntactically incorrect
bad format character	if the field descriptor is not recognizable

NAME

telnet – user interface to the TELNET protocol

SYNOPSIS

telnet [host [port]]

DESCRIPTION

telnet is used to communicate with another host using the TELNET protocol. If *telnet* is invoked without arguments, it enters command mode, which is indicated by its prompt (**telnet>**). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an *open* command (see below) with those arguments.

Once a connection has been opened, *telnet* enters an input mode. The input mode entered will be either character at a time or line by line, depending on what the remote system supports.

In character at a time mode, most text typed is immediately sent to the remote host for processing.

In line by line mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The local echo character (initially **CTRL e**) may be used to turn off and on the local echo (this would generally be used to enter passwords without the passwords being echoed).

In either mode, if the *localchars* toggle is TRUE (the default in line mode; see below), the user's *quit*, *intr*, and *flush* characters are trapped locally and sent as TELNET protocol sequences to the remote side. There are options (see **toggle** *autoflush* and **toggle** *autosynch* below) that cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of *quit* and *intr*).

While connected to a remote host, *telnet* command mode may be entered by typing the *telnet* escape character (initially **CTRL]**). When in command mode, the normal terminal editing conventions are available.

Commands

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the **mode**, **set**, **toggle**, and **display** commands).

open *host* [*port*]
: Open a connection to the named host. If no port number is specified, *telnet* will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see *hosts*(5)) or an Internet address specified in the dot notation (see *inet*(3I)).

close
: Close a TELNET session and return to command mode.

quit
: Close any open TELNET session and exit *telnet*. An end-of-file (in command mode) will also close a session and exit.

z
: Suspend *telnet*. This command only works when the user is running *csh*(1).

mode *type*
type is either *line* (for line by line mode) or *character* (for character at a time mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

status Show the current status of *telnet*. This includes the peer one is connected to, as well as the current mode.

display [*argument...*]
Displays all, or some, of the **set** and **toggle** values (see below).

? [*command*]
Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information for just that command.

send *arguments*
Sends one or more special character sequences to the remote host. The following arguments may be specified (more than one argument may be specified at a time):

escape Sends the current *telnet* escape character (initially ^]).

synch
Sends the TELNET SYNCH sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system — if it does not work, a lowercase **r** may be echoed on the terminal).

brk Sends the TELNET BRK (Break) sequence, which may have significance to the remote system.

ip Sends the TELNET IP (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

ao Sends the TELNET AO (Abort Output) sequence, which should cause the remote system to flush all output *from* the remote system *to* the user's terminal.

ayt Sends the TELNET AYT (Are You There) sequence, to which the remote system may or may not choose to respond.

ec Sends the TELNET EC (Erase Character) sequence, which should cause the remote system to erase the last character entered.

el Sends the TELNET EL (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

ga Sends the TELNET GA (Go Ahead) sequence, which likely has no significance to the remote system.

nop Sends the TELNET NOP (No OPeration) sequence.

? Prints out help information for the **send** command.

set *argument value*

Set any one of a number of *telnet* variables to a specific value. The special value off turns off the function associated with the variable. The values of variables may be interrogated with the **display** command. The variables which may be specified are:

echo This is the value (initially ^**E**) which, when in line by line mode, toggles between doing local echoing of entered characters (for normal processing) and suppressing echoing of entered characters (e.g., for entering a password).

escape

This is the *telnet* escape character (initially ^[) which causes entry into *telnet* command mode (when connected to a remote system).

interrupt

If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *interrupt* character is typed, a TELNET IP sequence (see **send** *ip* above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's **intr** character.

quit If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *quit* character is typed, a TELNET BRK sequence (see **send** *brk* above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's **quit** character.

flushoutput

If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *flushoutput* character is typed, a TELNET AO sequence (see **send** *ao* above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's **flush** character.

erase If *telnet* is in *localchars* mode (see **toggle** *localchars* below), *and* if *telnet* is operating in character at a time mode, then when this character is typed, a TELNET EC sequence (see **send** *ec* above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's **erase** character.

kill If *telnet* is in *localchars* mode (see **toggle** *localchars* below), *and* if *telnet* is operating in character at a time mode, then when this character is typed, a TELNET EL sequence (see **send** *el* above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's **kill** character.

EOF If *telnet* is operating in line by line mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the EOF character is taken to be the terminal's EOF character.

toggle *arguments...*

Toggle (between TRUE and FALSE) various flags that control how *telnet* responds to events. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are:

localchars

If this is TRUE, then the *flush*, *interrupt*, *quit*, *erase*, and *kill* characters (see **set** above) are recognized locally, and transformed into (hopefully) appropriate TELNET control sequences (respectively *ao*, *ip*, *brk*, *ec*, and *el*; see **send** above). The initial value for this toggle is TRUE in line by line mode, and FALSE in character at a time mode.

autoflush

If *autoflush* and *localchars* are both TRUE, then when the *ao*, *intr*, or *quit* characters are recognized (and transformed into TELNET sequences; see **set** above for details), *telnet* refuses to display any data on the user's terminal until the remote system acknowledges (via a TELNET *Timing Mark* option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user has not done an "stty noflsh," otherwise FALSE (see *stty*(1)).

autosynch

If *autosynch* and *localchars* are both TRUE, then when either the *intr* or *quit* characters is typed (see **set** above for descriptions of the *intr* and *quit* characters), the resulting TELNET sequence sent is followed by the TELNET SYNCH sequence. This procedure *should* cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

crmod Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is FALSE.

debug Toggles socket level debugging (useful only to user **root**). The initial value for this toggle is FALSE.

flowcontrol

Toggles whether flow control (XON/XOFF processing) is negotiated or only done remotely. Negotiation is done using the TELNET TOGGLEFLOW control sequence. The initial value is TRUE, meaning negotiated.

options
Toggles the display of some internal *telnet* protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.

netdata
Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

? Displays the legal **toggle** commands.

SEE ALSO

inet(3I), hosts(5).
csh(1), stty(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.

BUGS

There is no adequate way for dealing with flow control.

On some remote systems, echo has to be turned off manually when in line by line mode.

There is enough settable state to justify a **.telnetrc** file. However, no capability for a **.telnetrc** file is provided.

In line by line mode, the terminal's EOF character is only recognized (and sent to the remote system) when it is the first character on a line.

NAME

telnetd – DARPA TELNET protocol server

SYNOPSIS

/etc/telnetd [–Xdfsw]

DESCRIPTION

The *telnetd* server supports the DARPA standard TELNET virtual terminal protocol. The *telnetd* server is invoked by the internet server (see *inetd*(1M)), normally for requests to connect to the TELNET port as indicated by the **/etc/services** file (see *services*(5)).

The *telnetd* server operates by allocating a pseudo-terminal device (see *pty*(7)) for a client, then creating a login process which has the slave side of the pseudo-terminal as *stdin*, *stdout*, and *stderr*. *telnetd* manipulates the master side of the pseudo-terminal, implementing the TELNET protocol and passing characters between the remote client and the login process.

When a TELNET session is started up, *telnetd* sends TELNET options to the client side indicating that it is ready to do remote echo of characters, to suppress go-ahead, and to receive terminal type information from the remote client. If the remote client is willing, the remote terminal type is propagated in the environment of the created login process. The pseudo-terminal allocated to the client is configured to operate in canonical mode and with TAB3, CS7, OCRNL, and ONLCR enabled (see *termio*(7)).

telnetd is willing to do echo, binary, suppress go-ahead, negotiate flow control, and timing mark. *telnetd* is willing to have the remote client do binary, terminal type, X display location, speed, window size, and suppress go-ahead.

If any of the options *negotiate flow control*, *X display location*, and *speed* cause problems with clients, the server can be told to not volunteer the options with the **–X** option followed by any of the letters *f*, *d*, *s*, or *w*, which represent the options flow control, display location, terminal speed, and window size, respectively.

SEE ALSO

inetd(1M), telnet(1C), services(5), pty(7).
termio(7) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.

BUGS

Some TELNET commands are only partially implemented.

The TELNET protocol allows for the exchange of the number of lines and columns on the user's terminal, but *telnetd* does not make use of them.

Because of bugs in the original 4.2BSD *telnet*(1C), *telnetd* performs some dubious protocol exchanges to try to discover if the remote client is, in fact, a 4.2BSD *telnet*(1C).

Binary mode has no common interpretation except between similar operating systems (the UNIX System in this case).

The terminal type name received from the remote client is converted to lowercase letters.

The *telnetd* server never sends TELNET go-ahead commands.

NAME

tftp – Trivial File Transfer Protocol program

SYNOPSIS

tftp [host]

DESCRIPTION

tftp is the user interface to the Internet Trivial File Transfer Protocol (TFTP), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case *tftp* uses *host* as the default host for future transfers (see the **connect** command below).

Commands

Once *tftp* is running, it issues the prompt **tftp>** and recognizes the following commands:

connect *host-name* [*port*]
: Set the *host* (and optionally *port*) for transfers. Note that TFTP, unlike FTP, does not maintain connections between transfers; thus, the **connect** command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the **connect** command; the remote host can be specified as part of the **get** or **put** commands.

mode *transfer-mode*
: Set the mode for transfers; *transfer-mode* may be one of **ascii** or **binary**. The default is **ascii**.

put *file*

put *localfile remotefile*
put *file1 file2 ... fileN remote-directory*
: Put a file or set of files to the specified remote file or directory. The destination can be in one of two forms: a file name on the remote host, if the host has already been specified, or a string of the form *host*:*filename* to specify both a host and file name at the same time. If the latter form is used, the host name specified becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be a UNIX System machine.

get *filename*
get *remotename localname*
get *file1 file2 ... fileN*
: Get a file or set of files from the specified *sources*. *Source* can be in one of two forms: a file name on the remote host, if the host has already been specified, or a string of the form *host*:*filename* to specify both a host and file name at the same time. If the latter form is used, the last host name specified becomes the default for future transfers.

quit Exit *tftp*. An end-of-file also exits.

verbose Toggle verbose mode.

trace Toggle packet tracing.

status Show current status.

rexmt *retransmission-timeout*
Set the per-packet retransmission timeout, in seconds.

timeout *total-transmission-timeout*
Set the total transmission timeout, in seconds.

ascii Shorthand for **mode ascii**.

binary Shorthand for **mode binary**.

? [*command-name* ...]
Print help information.

BUGS

Because there is no user-login or validation within TFTP, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site and therefore difficult to document here.

NAME

tftpd – DARPA Trivial File Transfer Protocol server

SYNOPSIS

/etc/tftpd [**–D** [path]] [**–r**] [**–R**] [**–U** [uid]]

DESCRIPTION

The *tftpd* server supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the *tftp* service description; see *services*(5). The server is normally started by *inetd*(1M).

The use of *tftpd* does not require an account or password on the remote system. Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. Files may be written only if they already exist and are publicly writable. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling *tftp* service. The server should have the user ID with the lowest possible privilege.

The available options are:

–D [*path*]
: Specify directory to *chdir* so that relative path names may be used. If *path* is not specified, the default path name is **/usr/spool/uucppublic**.

–r
: Enable restricted access. With this option enabled, file paths are restricted to the directory specified in **–D** or to the default **/usr/spool/uucppublic** if a directory is not specifed.

–R
: Enable relative paths. Relative paths are relative to the directory specified with **–D** or **/usr/spool/uucppublic** if **–D** is not specified.

–U [*uid*]
: Specify the restricted access UID. This provides an additional level of security because *tftpd* is not running as **root**. While *inetd*(1M) could be set up to start *tftpd* as a non-**root** user, the *chroot*(2) call necessary to restrict access can only be run by **root**. If *uid* is not provided, the default is BIN_ID.

Notes

tftpd is not enabled in the default *inetd* configuration because of potential security problems.

SEE ALSO

inetd(1M), tftp(1C), services(5).

NAME

timed – time server daemon

SYNOPSIS

/etc/timed [–t] [–M] [–X] [–n network **] [–i** network **]**

DESCRIPTION

timed is the time server daemon and is normally invoked at boot time from the **/etc/rc3.d/S05hbtcp** file. It synchronizes the host's time with the time of other machines in a local area network running *timed*(1M). These time servers will slow down the clocks of some machines and speed up the clocks of others to bring them to the average network time. The average network time is computed from measurements of clock differences using the ICMP timestamp request message.

The service provided by *timed* is based on a master slave scheme. When *timed* is started on a machine, it asks the master for the network time and sets the host's clock to that time. After that, it accepts synchronization messages periodically sent by the master and calls *adjtime*(3I) to perform the needed corrections on the host's clock.

It also communicates with *tdate*(1) in order to set the date globally, and with *timedc*(1M), a timed control program. If the machine running the master crashes, then the slaves will elect a new master from among slaves running with the **–M** flag. A *timed* running without the **–M** flag will remain a slave. The **–t** flag enables *timed* to trace the messages it receives in the file **/usr/adm/timed.log**. Tracing can be turned on or off by the program *timedc*(1M). *timed* normally checks for a master time server on each network to which it is connected, except as modified by the options described below. It will request synchronization service from the first master server located. If permitted by the **–M** flag, it will provide synchronization service on any attached networks on which no current master server was detected. Such a server propagates the time computed by the top-level master. The **–n** flag followed by the name of a network which the host is connected to (see *networks*(5)) overrides the default choice of the network addresses made by the program. Each time the **–n** flag appears, that network name is added to a list of valid networks. All other networks are ignored. The **–i** flag followed by the name of a network to which the host is connected (see *networks*(5)) overrides the default choice of the network addresses made by the program. Each time the **–i** flag appears, that network name is added to a list of networks to ignore. All other networks are used by the time daemon. The **–n** and **–i** flags are meaningless if used together.

The **–X** flag indicates to the daemon that it should never attempt to adjust the local system time. This is useful when the local system is using another time protocol such as *ntp* for time synchronization.

FILES

/usr/adm/timed.log	tracing file for timed
/usr/adm/timed.masterlog	log file for master timed

SEE ALSO

tdate(1), timedc(1M), adjtime(3I), gettimeofday(3I), networks(5), icmp(7).

WARNING

timed(1M) will only work on INTERACTIVE UNIX System Version 3.0 or later.

NAME

timedc – timed control program

SYNOPSIS

/etc/timedc [command [argument ...]]

DESCRIPTION

timedc is used to control the operation of the *timed*(1M) program. It may be used to measure the differences between machines' clocks, find the location where the master time server is running, enable or disable tracing of messages received by *timed*, and perform various debugging actions.

Without any arguments, *timedc*(1M) will prompt for commands from the standard input. If arguments are supplied, *timedc* interprets the first argument as a command and the remaining arguments as parameters to the command. The standard input may be redirected causing *timedc* to read commands from a file. Commands may be abbreviated; recognized commands are:

? [command ...]
help [command ...]
: Print a short description of each command specified in the argument list or, if no arguments are given, a list of the recognized commands.

clockdiff host ...
: Compute the differences between the clock of the host machine and the clocks of the machines given as arguments.

trace { on | off }
: Enable or disable the tracing of incoming messages to *timed* in the file **/usr/adm/timed.log**.

quit : Exit from *timedc*.

Other commands may be included for use in testing and debugging *timed*; the help command and the program source may be consulted for details.

FILES

/usr/adm/timed.log	tracing file for timed
/usr/adm/timed.masterlog	log file for master timed

SEE ALSO

tdate(1), timed(1M), adjtime(3I), icmp(7).

DIAGNOSTICS

?Ambiguous command	abbreviation matches more than one command
?Invalid command	no match found
?Privileged command	command can be executed by **root** only

WARNING

timedc(1M) will only work on INTERACTIVE UNIX System Version 3.0 or later.

NAME

trpt – transliterate protocol trace

SYNOPSIS

trpt [–a] [–s] [–t] [–f] [–j] [–p hex-address]
[system [core]]

DESCRIPTION

trpt interrogates the buffer of TCP trace records created when a socket is marked for debugging (see *getsockopt*(3I)) and prints a readable description of these records. When no options are supplied, *trpt* prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB). The following options may be used to alter this behavior:

–a in addition to the normal output, print the values of the source and destination addresses for each packet recorded.

–s in addition to the normal output, print a detailed description of the packet sequencing information.

–t in addition to the normal output, print the values for all timers at each point in the trace.

–f follow the trace as it occurs, waiting a short time for additional records each time the end of the log is reached.

–j just give a list of the protocol control block addresses for which there are trace records.

–p show only trace records associated with the protocol control block, the address of which follows.

The recommended use of *trpt* is as follows. Isolate the problem and enable debugging on the socket(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the **–A** option to *netstat*(1). Then run *trpt* with the **–p** option, supplying the associated protocol control block addresses. The **–f** option can be used to follow the trace log once the trace is located. If there are many sockets using the debugging option, the **–j** option may be useful in checking to see if any trace records are present for the socket in question.

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

FILES

/unix
/dev/kmem

SEE ALSO

netstat(1), getsockopt(3I).

DIAGNOSTICS

no namelist
when the system image doesn't contain the proper symbols to find the trace buffer; others which should be self-explanatory.

BUGS

Should also print the data for each input or output, but this is not saved in the trace record.
The output format is inscrutable and should be described here.

NAME

_insque, _remque – insert/remove element from a queue

SYNOPSIS

```
struct qelem {
struct qelem *q_forw;
struct qelem *q_back;
struct char q_data[];
};

_insque(elem, pred)
struct qelem *elem, *pred;

_remque(elem)
struct qelem *elem;
```

DESCRIPTION

_insque and *_remque* manipulate queues built from doubly linked lists. Each element in the queue must be in the form of a **struct** *qelem*. *_insque* inserts *elem* in a queue immediately after *pred;* *_remqueue* removes an entry *elem* from a queue.

NAME

accept – accept a connection on a socket

SYNOPSIS

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

ns = accept(s, addr, addrlen)
int ns, s;
struct sockaddr *addr;
int *addrlen;
```

DESCRIPTION

The argument *s* is a socket that has been created with *socket*(3I), bound to an address with *bind*(3I), and is listening for connections after a *listen*(3I). *accept* extracts the first connection on the queue of pending connections, creates a new socket with the same properties of *s* and allocates a new file descriptor, *ns*, for the socket. If no pending connections are present on the queue, and the socket is not marked as non-blocking, *accept* blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, *accept* returns an error as described below. The accepted socket, *ns*, may not be used to accept more connections. The original socket *s* remains open.

The argument *addr* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the *addr* parameter is determined by the domain in which the communication is occurring. The *addrlen* is a value-result parameter; it should initially contain the amount of space pointed to by *addr*; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types, currently with SOCK_STREAM.

It is possible to *select*(3I) or *poll*(2) a socket for the purposes of doing an *accept* by selecting it for read.

Errors

The *accept* will fail if:

[EBADF]	The descriptor is invalid.
[ENOSTR]	The descriptor references a file, not a stream.
[EOPNOTSUPP]	The referenced socket is not of type SOCK_STREAM.
[EFAULT]	The *addr* parameter is not in a writable part of the user address space.
[EAGAIN]	The socket is marked non-blocking and no connections are present to be accepted.
[EPROTO]	The commands required prior to this one were not run or did not succeed.

SEE ALSO

bind(3I), connect(3I), listen(3I), select(3I), socket(3I).
poll(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

RETURN VALUES

The call returns −1 on error. If it succeeds, it returns a non-negative integer that is a descriptor for the accepted socket.

NAME

adjtime – correct the time to allow synchronization of the system clock

SYNOPSIS

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

adjtime(delta, olddelta)
struct timeval *delta;
struct timeval *olddelta;
```

DESCRIPTION

adjtime makes small adjustments to the system time, as returned by *gettimeofday*(3I), advancing or retarding it by the time specified by the *timeval delta*. If *delta* is negative, the clock is slowed down by incrementing it more slowly than normal until the correction is complete. If *delta* is positive, a larger increment than normal is used. The skew used to perform the correction is generally a fraction of 1 percent. Thus, the time is always a monotonically increasing function. A time correction from an earlier call to *adjtime* may not be finished when *adjtime* is called again. If *olddelta* is nonzero, the structure pointed to will contain, upon return, the number of microseconds still to be corrected from the earlier call.

This call may be used by time servers that synchronize the clocks of computers in a local area network. Such time servers would slow down the clocks of some machines and speed up the clocks of others to bring them to the average network time.

The call *adjtime*(3I) is restricted to the superuser.

Errors

The following error codes may be set in *errno*:

[EFAULT]	An argument points outside the process's allocated address space.
[EPERM]	The process's effective user ID is not that of the superuser.

SEE ALSO

timed(1M), timedc(1M), gettimeofday(3I).

RETURN VALUES

A return value of 0 indicates that the call succeeded. A return value of −1 indicates that an error occurred, and in this case an error code is stored in the global variable *errno*.

WARNING

adjtime(3I) will only work on INTERACTIVE UNIX System Version 3.0 or later.

NAME

bind – bind a name to a socket

SYNOPSIS

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

bind(s, name, namelen)
int s;
struct sockaddr *name;
int namelen;
```

DESCRIPTION

bind assigns a name to an unnamed socket. When a socket is created with *socket*(3I) it exists in a name space (address family) but has no name assigned. *bind* requests that *name* be assigned to the socket.

Notes

The rules used in name binding vary between communication domains. Consult the manual entries in Section 7 for detailed information.

Errors

The *bind* call will fail if:

[EBADF]	*S* is not a valid descriptor.
[ENOSTR]	*S* is not a stream.
[EADDRNOTAVAIL]	The specified address is not available from the local machine.
[EADDRINUSE]	The specified address is already in use.
[EINVAL]	The socket is already bound to an address.
[EACCES]	The requested address is protected, and the current user has inadequate permission to access it.
[EFAULT]	The *name* parameter is not in a valid part of the user address space.
[ERANGE]	The *namelen* parameter is too large.
[EPROTO]	The commands required prior to this one were not run or did not succeed.

SEE ALSO

connect(3I), getsockname(3I), listen(3I), socket(3I).

RETURN VALUES

If the bind is successful, a 0 value is returned. A return value of −1 indicates an error, which is further specified in the global *errno*.

NAME

bcopy, bcmp, bzero – byte string operations

SYNOPSIS

```
bcopy(src, dst, length)
char *src, *dst;
int length;

bcmp(b1, b2, length)
char *b1, *b2;
int length;

bzero(b, length)
char *b;
int length;
```

DESCRIPTION

The functions *bcopy, bcmp,* and *bzero* operate on variable length strings of bytes. They do not check for null bytes as the routines in *string*(3) do.

bcopy copies *length* bytes from string *src* to the string *dst.*

bcmp compares byte string *b1* against byte string *b2,* returning zero if they are identical, nonzero otherwise. Both strings are assumed to be *length* bytes long.

bzero places *length* zero bytes in the string *b1.*

BUGS

The *bcopy* routine takes parameters backwards from *strcpy*(3).

NAME

htonl, htons, ntohl, ntohs – convert values between host and network byte order

SYNOPSIS

```
#include <sys/bsdtypes.h>
#include <netinet/in.h>

netlong = htonl(hostlong);
u_long netlong, hostlong;

netshort = htons(hostshort);
u_short netshort, hostshort;

hostlong = ntohl(netlong);
u_long hostlong, netlong;

hostshort = ntohs(netshort);
u_short hostshort, netshort;
```

DESCRIPTION

These routines convert 16- and 32-bit quantities between network byte order and host byte order. On machines such as the SUN these routines are defined as null macros in the *include* file **<netinet/in.h>**.

These routines are most often used in conjunction with Internet addresses and ports as returned by *gethostbyname* (see *ghostnamed*(3I)) and *getservent*(3I).

Note

Intel processors handle bytes in a reverse order from the way most other companies' processors handle bytes. This is not expected to be altered in the near future.

SEE ALSO

getservent(3I), ghostnamed(3I).

NAME

connect – initiate a connection on a socket

SYNOPSIS

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

connect(s, name, namelen)
int s;
struct sockaddr *name;
int namelen;
```

DESCRIPTION

The parameter *s* is a socket. If it is of type SOCK_DGRAM, then this call specifies the peer with which the socket is to be associated; this address is that to which datagrams are to be sent, and the only address from which datagrams are to be received. If the socket is of type SOCK_STREAM, then this call attempts to make a connection to another socket. The other socket is specified by *name*, which is an address in the communications space of the socket. Each communications space interprets the *name* parameter in its own way. Generally, stream sockets may successfully *connect* only once; datagram sockets may use *connect* multiple times to change their association. Datagram sockets may dissolve the association by connecting to an invalid address, such as a null address.

Errors

The call fails if:

[EBADF]	*S* is not a valid descriptor.
[ENOSTR]	*S* is a descriptor for a file, not a stream.
[EADDRNOTAVAIL]	The specified address is not available on this machine.
[EAFNOSUPPORT]	Addresses in the specified address family cannot be used with this socket.
[EISCONN]	The socket is already connected.
[ETIMEDOUT]	Connection establishment timed out without establishing a connection.
[ECONNREFUSED]	The attempt to connect was forcefully rejected.
[ENETUNREACH]	The network is not reachable from this host.
[EADDRINUSE]	The address is already in use.
[EFAULT]	The *name* parameter specifies an area outside the process address space.
[EALREADY]	The socket is non-blocking and a previous connection attempt has not yet been completed.
[ERANGE]	The *namelen* parameter is too large.
[EPROTO]	The commands required prior to this one were not run or did not succeed.

SEE ALSO

accept(3I), getsockname(3I), socket(3I).

RETURN VALUES

If the connection or binding succeeds, then 0 is returned. Otherwise a −1 is returned, and a more specific error code is stored in *errno*.

NAME

gethostid, sethostid – get/set unique identifier of current host

SYNOPSIS

```
hostid = gethostid()
long hostid;

sethostid(hostid)
long hostid;
```

DESCRIPTION

sethostid establishes a 32-bit identifier for the current processor that is intended to be unique among all UNIX Systems in existence. This is normally a DARPA Internet address for the local machine. This call is allowed only to the superuser. The system administrator can set the *hostid* in a script added to the **/etc/rc3.d** directory by calling *hostid* with the host name as a parameter.

gethostid returns the 32-bit identifier for the current processor.

SEE ALSO

gethostname(3I).

BUGS

A value of 32 bits for the identifier is too small.

NAME

gethostname – get name of current host

SYNOPSIS

```
gethostname(name, namelen)
char *name;
int namelen;
```

DESCRIPTION

gethostname returns the standard host name for the current processor. The parameter *namelen* specifies the size of the *name* array. The returned name is null-terminated unless insufficient space is provided.

SEE ALSO

gethostid(3I).

RETURN VALUES

If the call succeeds, a value of 0 is returned. If the call fails, then a value of −1 is returned, and an error code is placed in the global location *errno*.

NAME

getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent – get network entry

SYNOPSIS

```
#include <netdb.h>

struct netent *getnetent()

struct netent *getnetbyname(name)
char *name;

struct netent *getnetbyaddr(net, type)
long net;
int type;

setnetent(stayopen)
int stayopen;

endnetent()
```

DESCRIPTION

getnetent, *getnetbyname*, and *getnetbyaddr* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network database, **/etc/networks**.

```
struct   netent {
         char          *n_name;    /* official name of net */
         char          **n_aliases; /* alias list */
         int           n_addrtype; /* net number type */
         unsigned long n_net;      /* net number */
};
```

The members of this structure are:

n_name The official name of the network.

n_aliases A zero-terminated list of alternate names for the network.

n_addrtype The type of the network number returned; currently only AF_INET.

n_net The network number. Network numbers are returned in machine byte order.

getnetent reads the next line of the file, opening the file if necessary.

setnetent opens and rewinds the file. If the *stayopen* flag is nonzero, the net database will not be closed after each call to *getnetbyname* or *getnetbyaddr*.

endnetent closes the file.

getnetbyname and *getnetbyaddr* sequentially search from the beginning of the file until a matching net name or net address and type is found, or until EOF is encountered. Network numbers are supplied in host order.

FILES

/etc/networks

SEE ALSO

networks(5).

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only Internet network numbers are currently understood. Expecting network numbers to fit in no more than 32 bits is probably naive.

NAME

getpeername – get name of connected peer

SYNOPSIS

getpeername(s, name, namelen)
int s;
struct sockaddr *name;
int *namelen;

DESCRIPTION

getpeername returns the name of the peer connected to socket *s*. The *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes). The name is truncated if the buffer provided is too small.

Errors

The call succeeds unless:

[EBADF]	The argument *s* is not a valid descriptor.
[ENOSTR]	The argument *s* is a file, not a stream.
[ENOTCONN]	The socket is not connected.
[EAGAIN]	Insufficient resources were available in the system to perform the operation.
[EFAULT]	The *name* parameter points to memory not in a valid part of the process address space.
[EPROTO]	The commands required prior to this one were not run or did not succeed.

SEE ALSO

accept(3I), bind(3I), getsockname(3I), socket(3I).

RETURN VALUES

A 0 is returned if the call succeeds, –1 if it fails.

NAME

getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent – get protocol entry

SYNOPSIS

```
#include <netdb.h>

struct protoent *getprotoent()

struct protoent *getprotobyname(name)
char *name;

struct protoent *getprotobynumber(proto)
int proto;

setprotoent(stayopen)
int stayopen

endprotoent()
```

DESCRIPTION

getprotoent, *getprotobyname*, and *getprotobynumber* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network protocol database, **/etc/protocols**.

```
struct  protoent    {
        char        *p_name;        /* official name of protocol */
        char        **p_aliases;    /* alias list */
        int         p_proto;        /* protocol number */
};
```

The members of this structure are:

p_name The official name of the protocol.

p_aliases A zero terminated list of alternate names for the protocol.

p_proto The protocol number.

getprotoent reads the next line of the file, opening the file if necessary.

setprotoent opens and rewinds the file. If the *stayopen* flag is nonzero, the net database will not be closed after each call to *getprotobyname* or *getprotobynumber*.

endprotoent closes the file.

getprotobyname and *getprotobynumber* sequentially search from the beginning of the file until a matching protocol name or protocol number is found, or until EOF is encountered.

FILES

/etc/protocols

SEE ALSO

protocols(5).

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet protocols are currently understood.

NAME

getservent, getservbyport, getservbyname, setservent, endservent – get service entry

SYNOPSIS

```
#include <netdb.h>

struct servent *getservent()

struct servent *getservbyname(name, proto)
char *name, *proto;

struct servent *getservbyport(port, proto)
int port; char *proto;

setservent(stayopen)
int stayopen

endservent()
```

DESCRIPTION

getservent, *getservbyname*, and *getservbyport* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network services database, **/etc/services**.

```
struct servent {
        char    *s_name;        /* official name of service */
        char    **s_aliases;    /* alias list */
        int     s_port;         /* port service resides at */
        char    *s_proto;       /* protocol to use */
};
```

The members of this structure are:

s_name The official name of the service.

s_aliases A zero terminated list of alternate names for the service.

s_port The port number at which the service resides. Port numbers are returned in network byte order.

s_proto The name of the protocol to use when contacting the service.

getservent reads the next line of the file, opening the file if necessary.

setservent opens and rewinds the file. If the *stayopen* flag is nonzero, the net database will not be closed after each call to *getservbyname* or *getservbyport*. *endservent* closes the file.

getservbyname and *getservbyport* sequentially search from the beginning of the file until a matching protocol name or port number is found, or until EOF is encountered. If a protocol name is also supplied (non-NULL), searches must also match the protocol.

FILES

/etc/services

SEE ALSO

getprotoent(3I), services(5).

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Expecting port numbers to fit in a 32-bit quantity is probably naive.

NAME

getsockname – get socket name

SYNOPSIS

```
getsockname(s, name, namelen)
int s;
struct sockaddr *name;
int *namelen;
```

DESCRIPTION

getsockname returns the current *name* for the specified socket. The *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes).

Errors

The call succeeds unless:

[EBADF]	The argument *s* is not a valid descriptor.
[ENOSTR]	The argument *s* is a file, not a stream.
[EAGAIN]	Insufficient resources were available in the system to perform the operation.
[EFAULT]	The *name* parameter points to memory not in a valid part of the process address space.
[EPROTO]	The commands required prior to this one were not run or did not succeed.

SEE ALSO

bind(3I), socket(3I).

RETURN VALUES

A 0 is returned if the call succeeds, –1 if it fails.

NAME

getsockopt, setsockopt – get and set options on sockets

SYNOPSIS

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

getsockopt(s, level, optname, optval, optlen)
int s, level, optname;
char *optval;
int *optlen;

setsockopt(s, level, optname, optval, optlen)
int s, level, optname;
char *optval;
int optlen;
```

DESCRIPTION

getsockopt and *setsockopt* manipulate *options* associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost "socket" level.

When manipulating socket options the level at which the option resides and the name of the option must be specified. To manipulate options at the "socket" level, *level* is specified as SOL_SOCKET. To manipulate options at any other level the protocol number of the appropriate protocol controlling the option is supplied. For example, to indicate that an option is to be interpreted by the TCP protocol, *level* should be set to the protocol number of TCP; see *getprotoent*(3I).

The parameters *optval* and *optlen* are used to access option values for *setsockopt*. For *getsockopt* they identify a buffer in which the value for the requested option(s) are to be returned. For *getsockopt*, *optlen* is a value-result parameter, initially containing the size of the buffer pointed to by *optval*, and modified on return to indicate the actual size of the value returned. If no option value is to be supplied or returned, *optval* may be supplied as 0.

optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file **<sys/socket.h>** contains definitions for "socket" level options, described below. Options at other protocol levels vary in format and name; consult the appropriate entries in Section 7.

Most socket-level options take an *int* parameter for *optval*. For *setsockopt*, the parameter should nonzero to enable a boolean option, or zero if the option is to be disabled.

The following options are recognized at the socket level. Except as noted, each may be examined with *getsockopt* and set with *setsockopt*.

SO_DEBUG	toggle recording of debugging information
SO_REUSEADDR	toggle local address reuse
SO_KEEPALIVE	toggle keep connections alive
SO_DONTROUTE	toggle routing bypass for outgoing messages

SO_BROADCAST	toggle permission to transmit broadcast messages
SO_OOBINLINE	toggle reception of out-of-band data in band
SO_SNDBUF	set buffer size for output
SO_RCVBUF	set buffer size for input
SO_TYPE	get the type of the socket (get only)
SO_ERROR	get and clear error on the socket (get only)

SO_DEBUG enables debugging in the underlying protocol modules. SO_REUSEADDR indicates that the rules used in validating addresses supplied in a *bind*(3I) call should allow reuse of local addresses. SO_KEEPALIVE enables the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified via a SIGPIPE signal. SO_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

The option SO_BROADCAST requests permission to send broadcast datagrams on the socket. Broadcast was a privileged operation in earlier versions of the system. With protocols that support out-of-band data, the SO_OOBINLINE option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with *recv* or *read* calls without the MSG_OOB flag. SO_SNDBUF and SO_RCVBUF are options to adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections, or may be decreased to limit the possible backlog of incoming data. The system places an absolute limit on these values. Finally, SO_TYPE and SO_ERROR are options used only with *setsockopt*. SO_TYPE returns the type of the socket, such as SOCK_STREAM; it is useful for servers that inherit sockets on startup. SO_ERROR returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

Errors

The call succeeds unless:

[EBADF]	The argument *s* is not a valid descriptor.
[ENOSTR]	The argument *s* is a file, not a stream.
[ENOPROTOOPT]	The option is unknown at the level indicated.
[EFAULT]	The address pointed to by *optval* is not in a valid part of the process address space. For *getsockopt*, this error may also be returned if *optlen* is not in a valid part of the process address space.
[EPROTO]	The commands required prior to this one were not run or did not succeed.

SEE ALSO

getprotoent(3I), socket(3I).
ioctl(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

RETURN VALUES

A 0 is returned if the call succeeds, −1 if it fails.

BUGS

Several of the socket options are handled at lower levels of the system.

NAME

gettimeofday – get date and time

SYNOPSIS

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

gettimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;
```

DESCRIPTION

The system's notion of the current Greenwich time and the current time zone is obtained with the *gettimeofday* call. The time is expressed in seconds and microseconds since midnight (0 hour), January 1, 1970. The resolution of the system clock is hardware dependent, and the time may be updated continuously or in "ticks."

The structures pointed to by *tp* and *tzp* are defined in **<sys/time.h>** as:

```
struct timeval {
    long    tv_sec;         /* seconds since Jan. 1, 1970 */
    long    tv_usec;        /* and microseconds */
};

struct timezone {
    int     tz_minuteswest; /* of Greenwich */
    int     tz_dsttime;     /* type of dst correction to apply */
};
```

The *timezone* parameter is ignored.

Errors

The following error code may be set in *errno*:

[EFAULT] An argument address referenced invalid memory.

RETURN VALUES

A 0 return value indicates that the call succeeded. A −1 return value indicates an error occurred, and in this case an error code is stored into the global variable *errno*.

WARNING

The microsecond portion of the *timeval* structure (*tv_usec*) is more accurate under the INTERACTIVE UNIX Operating System Version 3.0 or later.

NAME

getusershell, setusershell, endusershell – get legal user shells

SYNOPSIS

char *getusershell()

setusershell()

endusershell()

DESCRIPTION

getusershell returns a pointer to a legal user shell as defined by the system in the file **/etc/shells.** If **/etc/shells** does not exist, the two standard system shells **/bin/sh** and **/bin/csh** are returned.

getusershell reads the next line (opening the file if necessary); *setusershell* rewinds the file; *endusershell* closes it.

FILES

/etc/shells

DIAGNOSTICS

The routine *getusershell* returns a null pointer (0) on EOF or error.

BUGS

All information is contained in a static area, so it must be copied if it is to be saved.

NAME

gethostbyname, gethostbyaddr, gethostent, sethostent, endhostent – get network host entry

SYNOPSIS

```
#include <netdb.h>

extern int h_errno;

struct hostent *gethostbyname(name)
char *name;

struct hostent *gethostbyaddr(addr, len, type)
char *addr; int len, type;

struct hostent *gethostent()

sethostent(stayopen)
int stayopen;

endhostent()
```

DESCRIPTION

gethostbyname and *gethostbyaddr* each return a pointer to an object with the following structure. This structure contains either the information obtained from the name server, *named*(1M), or broken-out fields from a line in **/etc/hosts**. If the local name server is not running these routines do a lookup in **/etc/hosts**.

```
struct    hostent  {
          char     *h_name;        /* official name of host */
          char     **h_aliases;    /* alias list */
          int      h_addrtype;     /* host address type */
          int      h_length;       /* length of address */
          char     **h_addr_list;  /* list of addresses from name server */
};
#define h_addr h_addr_list[0]  /* address, for backward compatibility */
```

The members of this structure are:

h_name Official name of the host.

h_aliases A zero terminated array of alternate names for the host.

h_addrtype The type of address being returned; currently always AF_INET.

h_length The length, in bytes, of the address.

h_addr_list A zero terminated array of network addresses for the host. Host addresses are returned in network byte order.

h_addr The first address in *h_addr_list*; this is for backward compatibility.

sethostent allows a request for the use of a connected socket using TCP for queries. If the *stayopen* flag is nonzero, this sets the option to send all queries to the name server using TCP and to retain the connection after each call to *gethostbyname* or *gethostbyaddr*.

endhostent closes the TCP connection.

FILES

/etc/hosts

SEE ALSO

named(1M), resolver(3I), hosts(5).

DIAGNOSTICS

Error return status from *gethostbyname* and *gethostbyaddr* is indicated by return of a null pointer. The external integer *h_errno* may then be checked to see whether this is a temporary failure or an invalid or unknown host.

h_errno can have the following values:

HOST_NOT_FOUND	No such host is known.
TRY_AGAIN	This is usually a temporary error and means that the local server did not receive a response from an authoritative server. A retry at some later time may succeed.
NO_RECOVERY	This is a non-recoverable error.
NO_ADDRESS	The requested name is valid but does not have an IP address; this is not a temporary error. This means another type of request to the name server will result in an answer.

WARNINGS

gethostent is defined, and *sethostent* and *endhostent* are redefined, when *libc* is built to use only the routines to lookup in **/etc/hosts** and not the name server.

gethostent reads the next line of **/etc/hosts**, opening the file if necessary.

sethostent is redefined to open and rewind the file. If the *stayopen* argument is nonzero, the hosts database will not be closed after each call to *gethostbyname* or *gethostbyaddr*. *endhostent* is redefined to close the file.

BUGS

All information is contained in a static area, so it must be copied if it is to be saved. Only the Internet address format is currently understood.

NAME

inet_addr, inet_network, inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof – Internet address manipulation routines

SYNOPSIS

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

unsigned long inet_addr(cp)
char *cp;

unsigned long inet_network(cp)
char *cp;

char *inet_ntoa(in)
struct in_addr in;

struct in_addr inet_makeaddr(net, lna)
int net, lna;

int inet_lnaof(in)
struct in_addr in;

int inet_netof(in)
struct in_addr in;
```

DESCRIPTION

The routines *inet_addr* and *inet_network* each interpret character strings representing numbers expressed in the Internet standard dot notation, returning numbers suitable for use as Internet addresses and Internet network numbers, respectively. The routine *inet_ntoa* takes an Internet address and returns an ASCII string representing the address in dot notation. The routine *inet_makeaddr* takes an Internet network number and a local network address and constructs an Internet address from it. The routines *inet_netof* and *inet_lnaof* break apart Internet host addresses, returning the network number and local network address part, respectively.

All Internet address are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

Internet Addresses

Values specified using the dot notation take one of the following forms:

a.b.c.d
a.b.c
a.b
a

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on the 386 the bytes referred to above appear as *d.c.b.a*. That is, 386 bytes are ordered from right to left.

When a three-part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as **128.net.host**.

When a two-part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as **net.host**.

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied as parts in a . notation may be decimal, octal, or hexadecimal, as specified in the C language (i.e., a leading **0x** or **0X** implies hexadecimal; otherwise, a leading **0** implies octal; otherwise, the number is interpreted as decimal).

SEE ALSO

getnetent(3I), ghostnamed(3I), hosts(5), networks(5).

DIAGNOSTICS

The value −1 is returned by *inet_addr* and *inet_network* for malformed requests.

BUGS

The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed.

The string returned by *inet_ntoa* resides in a static memory area.

inet_addr should return a *struct in_addr*.

NAME

listen – listen for connections on a socket

SYNOPSIS

listen(s, backlog)
int s, backlog;

DESCRIPTION

To accept connections, a socket is first created with *socket*(3I), a willingness to accept incoming connections and a queue limit for incoming connections are specified with *listen*(3I), and then the connections are accepted with *accept*(3I). The *listen* call applies only to sockets of type SOCK_STREAM.

The *backlog* parameter defines the maximum length the queue of pending connections may grow to. If a connection request arrives with the queue full the client may receive an error with an indication of ECONNREFUSED, or, if the underlying protocol supports retransmission, the request may be ignored so that retries may succeed.

Errors

The call fails if:

[EBADF]	The argument *s* is not a valid descriptor.
[ENOSTR]	The argument *s* is not a stream.
[EOPNOTSUPP]	The socket is not of a type that supports the operation *listen*.
[EPROTO]	The commands required prior to this one were not run or did not succeed.

SEE ALSO

accept(3I), connect(3I), socket(3I).

RETURN VALUES

A 0 return value indicates success; –1 indicates an error.

BUGS

The *backlog* is currently limited (silently) to 5.

NAME

rcmd, rresvport, ruserok – routines for returning a stream to a remote command

SYNOPSIS

```
rem = rcmd(ahost, inport, locuser, remuser, cmd, fd2p);
char **ahost;
int inport;
char *locuser, *remuser, *cmd;
int *fd2p;

s = rresvport(port);
int *port;

ruserok(rhost, superuser, ruser, luser);
char *rhost;
int superuser;
char *ruser, *luser;
```

DESCRIPTION

rcmd is a routine used by the superuser to execute a command on a remote machine using an authentication scheme based on reserved port numbers. *rresvport* is a routine that returns a descriptor to a socket with an address in the privileged port space. *ruserok* is a routine used by servers to authenticate clients requesting service with *rcmd*. All three functions are present in the same file and are used by the *rshd*(1M) server (among others).

rcmd looks up the host **ahost* using *gethostbyname,* (see *ghostnamed*(3I)) returning −1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port *inport*.

If the connection succeeds, a socket in the Internet domain of type SOCK_STREAM is returned to the caller, and given to the remote command as *stdin* and *stdout*. If *fd2p* is nonzero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as being UNIX System signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the *stderr* (unit 2 of the remote command) will be made the same as the *stdout* and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

The protocol is described in detail in *rshd*(1M).

The *rresvport* routine is used to obtain a socket with a privileged address bound to it. This socket is suitable for use by *rcmd* and several other routines. Privileged Internet ports are those in the range 0 to 1023. Only the superuser is allowed to bind an address of this sort to a socket.

ruserok takes a remote host's name, as returned by a *gethostbyaddr* routine (see *ghostnamed*(3I)), two user names and a flag indicating whether the local user's name is that of the superuser. It then checks the files **/etc/hosts.equiv** and, possibly, **.rhosts** in the current working

directory (normally the local user's home directory) to see if the request for service is allowed. A 0 is returned if the machine name is listed in the **hosts.equiv** file, or the host and remote user name are found in the **.rhosts** file; otherwise *ruserok* returns −1. If the *superuser* flag is 1, the checking of the **host.equiv** file is bypassed. If the local domain (as obtained from *gethostname*(3I)) is the same as the remote domain, only the machine name need be specified.

SEE ALSO

rlogin(1C), rsh(1C), rexecd(1M), rlogind(1M), rshd(1M), ghostnamed(3I) rexec(3I).
intro(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

DIAGNOSTICS

rcmd returns a valid socket descriptor on success. It returns −1 on error and prints a diagnostic message on the standard error.

rresvport returns a valid, bound socket descriptor on success. It returns −1 on error with the global value *errno* set according to the reason for failure. The error code EAGAIN is overloaded to mean "All network ports in use."

NAME

recv, recvfrom, recvmsg – receive a message from a socket

SYNOPSIS

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

cc = recv(s, buf, len, flags)
int cc, s;
char *buf;
int len, flags;

cc = recvfrom(s, buf, len, flags, from, fromlen)
int cc, s;
char *buf;
int len, flags;
struct sockaddr *from;
int *fromlen;

cc = recvmsg(s, msg, flags)
int cc, s;
struct msghdr msg[];
int flags;
```

DESCRIPTION

recv, *recvfrom*, and *recvmsg* are used to receive messages from a socket.

The *recv* call is normally used only on a *connected* socket (see *connect*(3I)), while *recvfrom* and *recvmsg* may be used to receive data on a socket whether it is in a connected state.

If *from* is nonzero, the source address of the message is filled in. *fromlen* is a value-result parameter, initialized to the size of the buffer associated with *from*, and modified on return to indicate the actual size of the address stored there. The length of the message is returned in *cc*. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see *socket*(3I)).

If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is nonblocking (see *ioctl*(2)) in which case a *cc* of −1 is returned with the external variable *errno* set to EAGAIN.

The *select*(3I) or *poll*(2) call may be used to determine when more data arrives.

The *flags* argument to a recv call is formed by **or**ing one or more of the values:

```
#define MSG_OOB     0x1     /* process out-of-band data */
#define MSG_PEEK    0x2     /* peek at incoming message */
```

The *recvmsg* call uses a *msghdr* structure to minimize the number of directly supplied parameters. This structure has the following form, as defined in **<sys/socket.h>**:

```
struct msghdr {
        caddr_t   msg_name;        /* optional address */
        int       msg_namelen;     /* size of address */
        struct    iovec *msg_iov;  /* scatter/gather array */
        int       msg_iovlen;      /* # elements in msg_iov */
        caddr_t   msg_accrights;   /* access rights sent/received */
        int       msg_accrightslen;
};
```

Here *msg_name* and *msg_namelen* specify the destination address if the socket is unconnected; *msg_name* may be given as a null pointer if no names are desired or required. The *msg_iov* and *msg_iovlen* describe the scatter gather locations, as described in *read*(2). A buffer to receive any access rights sent along with the message is specified in *msg_accrights*, which has length *msg_accrightslen*. Access rights are currently limited to file descriptors, which each occupy the size of an **int**.

Errors

The calls fail if:

[EBADF]	The argument *s* is an invalid descriptor.
[ENOSTR]	The argument *s* is not a stream.
[EAGAIN]	The socket is marked non-blocking and the receive operation would block.
[EINTR]	The receive was interrupted by delivery of a signal before any data was available for the receive.
[EFAULT]	The data was specified to be received into a non-existent or protected part of the process address space.
[EPROTO]	The commands required prior to this one were not run or did not succeed.

SEE ALSO

getsockopt(3I), select(3I), send(3I), socket(3I).
fcntl(2), poll(2), read(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

RETURN VALUES

These calls return the number of bytes received, or −1 if an error occurred.

NAME

rename – change the name of a file

SYNOPSIS

rename(from, to)
char *from, *to;

DESCRIPTION

rename causes the link named *from* to be renamed as *to*. If *to* exists, then it is first removed. Both *from* and *to* must be of the same type (that is, both directories or both non-directories), and must reside on the same file system.

rename guarantees that an instance of *to* will always exist, even if the system should crash in the middle of the operation.

If the final component of *from* is a symbolic link, the symbolic link is renamed, not the file or directory to which it points.

Errors

rename will fail and neither of the argument files will be affected if any of the following are true:

[ENOENT]	A component of the *from* path does not exist, or a path prefix of *to* does not exist.
[EACCESS]	Search permission is denied for a component of either argument's path prefix, or the requested link requires writing in a directory with a mode that denies *write* permission.
[EPERM]	The file named by *from* is a directory and the effective user ID is not superuser.
[ENOTDIR]	A component of either argument's path prefix is not a directory.
[EISDIR]	*to* is a directory, but *from* is not a directory.
[EXDEV]	The link named by *to* and the file name by *from* are on different logical devices (file systems).
[EIO]	An I/O error has occurred while accessing the file system.
[EFAULT]	*from* or *to* point to outside the allocated address space of the process.
[EROFS]	The path prefix of *to* resides on a *read*-only file system.
[EINVAL]	*from* is a parent directory of *to*, or an attempt is made to rename dot (.) or dot-dot (..).

SEE ALSO

open(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

RETURN VALUES

A 0 value is returned if the operation succeeds; otherwise *rename* returns −1, and the global variable *errno* indicates the reason for the failure.

NAME

res_mkquery, res_send, res_init, dn_comp, dn_expand – resolver routines

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

res_mkquery(op, dname, class, type, data, datalen, newrr, buf, buflen)
int op;
char *dname;
int class, type;
char *data;
int datalen;
struct rrec *newrr;
char *buf;
int buflen;

res_send(msg, msglen, answer, anslen)
char *msg;
int msglen;
char *answer;
int anslen;

res_init()

dn_comp(exp_dn, comp_dn, length, dnptrs, lastdnptr)
char *exp_dn, *comp_dn;
int length;
char **dnptrs, **lastdnptr;

dn_expand(msg, eomorig, comp_dn, exp_dn, length)
char *msg, *eomorig, *comp_dn, exp_dn;
int length;
```

DESCRIPTION

These routines are used for making, sending and interpreting packets to Internet domain name servers. Global information that is used by the *resolver* routines is kept in the variable *_res*. Most of the values have reasonable defaults and can be ignored. Options stored in *_res.options* are defined in **resolv.h** and are as follows. Options are a simple bit mask and are *or*'ed in to enable.

RES_INIT
: True if the initial name server address and default domain name are initialized, i.e., *res_init* has been called.

RES_DEBUG
: Print debugging messages.

RES_AAONLY
: Accept authoritative answers only. *res_send* will continue until it finds an authoritative answer or finds an error. Currently this is not implemented.

RES_USEVC
: Use TCP connections for queries instead of UDP.

RES_STAYOPEN
: Used with RES_USEVC to keep the TCP connection open between queries. This is useful only in programs that regularly have many queries. UDP should be the normal mode used.

RES_IGNTC
: Unused currently (ignore truncation errors; i.e., don't retry with TCP).

RES_RECURSE
: Set the "recursion desired" bit in queries. This is the default. (*res_send* does not do iterative queries and expects the name server to handle recursion.)

RES_DEFNAMES
: Append the default domain name to single label queries. This is the default.

RES_DONTUSE
: Disable *resolver* use. If this flag is set, *res_send* will return an error immediately, rather than calling the name server.

res_init reads the initialization file to get the default domain name and the Internet address of the initial hosts running the name server. If this line does not exist, the host running the *resolver* is tried. *res_mkquery* makes a standard query message and places it in *buf*. *res_mkquery* will return the size of the query or −1 if the query is larger than *buflen*. *op* is usually QUERY but can be any of the query types defined in **nameser.h**. *dname* is the domain name. If *dname* consists of a single label and the RES_DEFNAMES flag is enabled (the default), *dname* will be appended with the current domain name. The current domain name is defined in a system file and can be overridden by the environment variable LOCALDOMAIN. *newrr* is currently unused but is intended for making update messages.

res_send sends a query to name servers and returns an answer. It will call *res_init* if RES_INIT is not set, send the query to the local name server, and handle timeouts and retries. The length of the message is returned or −1 if there were errors. If RES_DONTUSE is set, *res_send* will return −1 immediately rather than calling the name server. This allows *gethostbyname* and *gethostbyaddr* (see *ghostnamed*(3I)) to use the **/etc/host** file even though the resolver routines are compiled in.

dn_expand expands the compressed domain name *comp_dn* to a full domain name. Expanded names are converted to uppercase. *msg* is a pointer to the beginning of the message, *exp_dn* is a pointer to a buffer of size *length* for the result. The size of compressed name is returned or −1 if there was an error.

dn_comp compresses the domain name *exp_dn* and stores it in *comp_dn*. The size of the compressed name is returned or −1 if there were errors. *length* is the size of the array pointed to by *comp_dn*. *dnptrs* is a list of pointers to previously compressed names in the current message. The first pointer points to to the beginning of the message and the list ends with NULL. *lastdnptr* is a pointer to the end of the array pointed to by *dnptrs*. A side effect is to update the list of

pointers for labels inserted into the message by *dn_comp* as the name is compressed. If *dnptr* is NULL, no attempt is made to compress names. If *lastdnptr* is NULL, the list is not updated.

FILES

/etc/resolv.conf
see resolver(5)

SEE ALSO

named(1M), ghostnamed(3I), resolver(5).
"Name Server Operations Guide for BIND."
RFC882, RFC883, RFC973, RFC974.

NAME

rexec – return stream to a remote command

SYNOPSIS

```
rem = rexec(ahost, inport, user, passwd, cmd, fd2p);
char **ahost;
int inport;
char *user, *passwd, *cmd;
int *fd2p;
```

DESCRIPTION

rexec looks up the host **ahost* using *gethostbyname* (see *ghostnamed*(3I)), returning −1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host. If a user name and password are both specified, then these are used to authenticate to the foreign host; otherwise the environment and then the user's **.netrc** file in his home directory are searched for appropriate information. If all this fails, the user is prompted for the information.

The port *inport* specifies which well-known DARPA Internet port to use for the connection; the call **getservbyname("exec", tcp**) (see *getservent*(3I)) will return a pointer to a structure, which contains the necessary port. The protocol for connection is described in detail in *rexecd*(1M).

If the connection succeeds, a socket in the Internet domain of type SOCK_STREAM is returned to the caller, and given to the remote command as *stdin* and *stdout*. If *fd2p* is nonzero, then an auxiliary channel to a control process will be setup, and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as being UNIX System signal numbers, to be forwarded to the process group of the command. The diagnostic information returned does not include remote authorization failure, as the secondary connection is set up after authorization has been verified. If *fd2p* is 0, then the *stderr* (unit 2 of the remote command) will be made the same as the *stdout* and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

SEE ALSO

rexecd(1M), getservent(3I), ghostnamed(3I), rcmd(3I).

NAME

select − synchronous I/O multiplexing

SYNOPSIS

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

nfound = select(nfds, readfds, writefds, exceptfds, timeout)
int nfound, nfds;
fd_set *readfds, *writefds, *exceptfds;
struct timeval *timeout;

FD_SET(fd, &fdset)
FD_CLR(fd, &fdset)
FD_ISSET(fd, &fdset)
FD_ZERO(&fdset)
int fd;
fd_set fdset;
```

DESCRIPTION

select examines the I/O descriptor sets whose addresses are passed in *readfds*, *writefds*, and *exceptfds* to see if some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. The first *nfds* descriptors are checked in each set; i.e., the descriptors from 0 through *nfds*−1 in the descriptor sets are examined. On return, *select* replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. The total number of ready descriptors in all the sets is returned in *nfound*.

The descriptor sets are stored as bit fields in arrays of integers. The following macros are provided for manipulating such descriptor sets: *FD_ZERO(&fdset)* initializes a descriptor set *fdset* to the null set. *FD_SET(fd, &fdset)* includes a particular descriptor *fd* in *fdset*. *FD_CLR(fd, &fdset)* removes *fd* from *fdset*. *FD_ISSET(fd, &fdset)* is nonzero if *fd* is a member of *fdset*, zero otherwise. The behavior of these macros is undefined if a descriptor value is less than zero or greater than or equal to *FD_SETSIZE*, which is normally at least equal to the maximum number of descriptors supported by the system.

If *timeout* is a nonzero pointer, it specifies a maximum interval to wait for the selection to complete. If *timeout* is a zero pointer, the select blocks indefinitely. To affect a poll, the *timeout* argument should be nonzero, pointing to a zero-valued timeval structure.

Any of *readfds*, *writefds*, and *exceptfds* may be given as zero pointers if no descriptors are of interest.

NOTES

select(3I) is implemented as a library routine that reformats its parameters and calls *poll*(2). It is more efficient to call *poll*(2) directly, instead of *select*(3I).

Errors

An error return from *select* indicates:

[EBADF] One of the descriptor sets specified an invalid descriptor.

[EINTR] A signal was delivered before the time limit expired and before any of the selected events occurred.

[EINVAL] The specified time limit is invalid. One of its components is negative or too large.

SEE ALSO

accept(3I), connect(3I), recv(3I), send(3I).
poll(2), read(2), write(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

RETURN VALUES

select returns the number of ready descriptors that are contained in the descriptor sets, or −1 if an error occurred. If the time limit expires, then *select* returns 0. If *select* returns with an error, including one due to an interrupted call, the descriptor sets will be unmodified.

BUGS

FD_SETSIZE is limited to 256 unless *select*(3I) is rebuilt. The number of file descriptors that can be selected in one call is limited to 32.

select should probably return the time remaining from the original timeout, if any, by modifying the time value in place. This may be implemented in future versions of the system. Thus, it is unwise to assume that the timeout value will be unmodified by the *select* call.

NAME

send, sendto, sendmsg – send a message from a socket

SYNOPSIS

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

cc = send(s, msg, len, flags)
int cc, s;
char *msg;
int len, flags;

cc = sendto(s, msg, len, flags, to, tolen)
int cc, s;
char *msg;
int len, flags;
struct sockaddr *to;
int tolen;

cc = sendmsg(s, msg, flags)
int cc, s;
struct msghdr msg[];
int flags;
```

DESCRIPTION

send, *sendto*, and *sendmsg* are used to transmit a message to another socket. *send* may be used only when the socket is in a *connected* state, whereas *sendto* and *sendmsg* may be used at any time.

The address of the target is given by *to* with *tolen* specifying its size. The length of the message is given by *len*. If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned and the message is not transmitted.

No indication of failure to deliver is implicit in a *send*. A return value of −1 indicates some locally detected errors.

If no message space is available at the socket to hold the message to be transmitted, then *send* normally blocks, unless the socket has been placed in non-blocking I/O mode. The *select*(3I) or *poll*(2) calls may be used to determine when it is possible to send more data.

The *flags* parameter may include one or more of the following:

```
#define MSG_OOB          0x1    /* process out-of-band data */
#define MSG_DONTROUTE    0x4    /* bypass routing, use direct
#                               interface */
```

The flag MSG_OOB is used to send out-of-band data on sockets that support this notion (e.g., SOCK_STREAM); the underlying protocol must also support out-of-band data. MSG_DONTROUTE is usually used only by diagnostic or routing programs.

See *recv*(3I) for a description of the *msghdr* structure.

Errors

[EBADF]	An invalid descriptor was specified.
[ENOSTR]	The argument *s* is not a stream.
[EFAULT]	An invalid user space address was specified for a parameter.

[EMSGSIZE] The socket requires that message be sent atomically, and the size of the message to be sent made this impossible.

[EAGAIN] The socket is marked non-blocking and the requested operation would block.

[EAGAIN] The system was unable to allocate an internal buffer. The operation may succeed when buffers become available.

[EPROTO] The commands required prior to this one were not run or did not succeed.

[EDESTADDRREQ] A destination address has not been specified.

sendmsg may also set *errno* to:

[ENOBUFS] The system was unable to allocate an internal buffer. The operation may succeed when buffers become available.

SEE ALSO

getsockopt(3I), recv(3I), select(3I), socket(3I).
fcntl(2), poll(2), write(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

RETURN VALUES

The call returns the number of characters sent, or −1 if an error occurred.

WARNINGS

As with all TCP/IP programs, you must define INET with the **−DINET** flag to *cc*(1).

NAME

seteuid, setegid – set the effective user and group IDs

SYNOPSIS

seteuid(euid)
int euid;

setegid(egid)
int egid;

DESCRIPTION

seteuid (setegid) sets the effective user ID (group ID) of the current process.

These calls are only permitted to the superuser or if the argument is the real or effective ID.

SEE ALSO

setreuid(3I).

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned.

NAME

setreuid, setregid − set the real and effective user and group IDs

SYNOPSIS

setreuid(ruid, euid)
int ruid, euid;

setregid(rgid, egid)
int rgid, egid;

DESCRIPTION

The real and effective user (group) IDs of the current process are set according to the arguments. If *ruid* (*rgid*) is −1, the current uid (gid) is filled in by the system. Unprivileged users may change the real user (group) ID to the effective user (group) ID and vice-versa; only the superuser may make other changes.

Errors

[EPERM] The current process is not the superuser and a change other than changing the effective user (group) ID to the real user (group) ID was specified.

SEE ALSO

seteuid(3I).

RETURN VALUES

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

NAME

settimeofday – set date and time

SYNOPSIS

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

settimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;
```

DESCRIPTION

The system's notion of the current Universal Coordinated Time (UTC) is set with the *settimeofday* call. The time is expressed in seconds and microseconds since midnight (0 hour), January 1, 1970. The resolution of the system clock is hardware dependent, and the time may be updated continuously or in "ticks." The *tzp* argument is ignored.

The structures pointed to by *tp* and *tzp* are defined in **<sys/time.h>** as:

```
struct timeval {
    long    tv_sec;         /* seconds since */
                            /* Jan. 1, 1970 */
    long    tv_usec;        /* and microseconds */
};

    struct timezone {
        int     tz_minuteswest;     /* of Greenwich */
        int     tz_dsttime;         /* type of dst */
                                    /* correction */
                                    /* to apply */
    };
```

Only the superuser may set the time of day or time zone.

Errors

The following error codes may be set in *errno*:

[EFAULT] An argument address referenced invalid memory.

[EPERM] A user other than the superuser attempted to set the time.

SEE ALSO

timed(1M), adjtime(3I).
stime(2), ctime(3C) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

RETURN VALUES

A 0 return value indicates that the call succeeded. A −1 return value indicates that an error occurred, and in this case an error code is stored into the global variable *errno.*

WARNING

The microsecond portion of the value is ignored for INTERACTIVE UNIX System releases prior to Version 3.0.

NAME

shutdown − shut down part of a full-duplex connection

SYNOPSIS

shutdown(s, how)
int s, how;

DESCRIPTION

The *shutdown* call causes all or part of a full-duplex connection on the socket associated with *s* to be shut down. If *how* is 0, then further receives will be disallowed. If *how* is 1, then further sends will be disallowed. If *how* is 2, then further sends and receives will be disallowed.

Errors

The call succeeds unless:

[EBADF]	*S* is not a valid descriptor.
[EPROTO]	The commands required prior to this one were not run or did not succeed.
[ENOSTR]	*S* is a file, not a stream.
[ENOTCONN]	The specified socket is not connected.

SEE ALSO

connect(3I), socket(3I).

RETURN VALUES

A 0 is returned if the call succeeds, −1 if it fails.

NAME

socket – create an endpoint for communication

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>

s = socket(domain, type, protocol)
int s, domain, type, protocol;

DESCRIPTION

socket creates an endpoint for communication and returns a descriptor.

The *domain* parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file **<sys/socket.h>**. The only currently understood format is:

PF_INET (ARPA Internet protocols)

The socket has the indicated *type*, which specifies the semantics of communication. Currently defined types are:

SOCK_STREAM
SOCK_DGRAM
SOCK_RAW

A SOCK_STREAM type provides sequenced, reliable, two-way connection based byte streams. An out-of-band data transmission mechanism may be supported. A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). SOCK_RAW sockets provide access to internal network protocols and interfaces. The type SOCK_RAW, which is available only to the superuser, is not described here.

The *protocol* specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the "communication domain" in which communication is to take place; see *protocols*(5).

Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. A stream socket must be in a *connected* state before any data may be sent or received on it. A connection to another socket is created with a *connect*(3I) call. Once connected, data may be transferred using *read*(2) and *write*(2) calls or some variant of the *send*(3I) and *recv*(3I) calls. When a session has been completed a *close*(2) may be performed. Out-of-band data may also be transmitted as described in *send*(3I) and received as described in *recv*(3I).

The communications protocols used to implement a SOCK_STREAM insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with −1 returns and with ETIMEDOUT as the specific code in the global variable *errno*. The

protocols optionally keep sockets "warm" by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for a extended period (e.g., 5 minutes). A SIGPIPE signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

SOCK_DGRAM and SOCK_RAW sockets allow sending of datagrams to correspondents named in *send*(3I) calls. Datagrams are generally received with *recvfrom*(3I), which returns the next datagram with its return address.

An *ioctl*(2) call can be used to specify a signal to send when out-of-band data arrives. An *fcntl*(2) call can be used to enable non-blocking I/O and asynchronous notification of I/O events via SIGIO.

The operation of sockets is controlled by socket level *options*. These options are defined in the file **<sys/socket.h>**. *setsockopt*(3I) and *getsockopt*(3I) are used to set and get options, respectively.

Errors

The *socket* call fails if:

[EPROTONOSUPPORT]	The protocol type or the specified protocol is not supported within this domain.
[EMFILE]	The per-process descriptor table is full.
[ENFILE]	The system file table is full.
[EACCES]	Permission to create a socket of the specified type and/or protocol is denied.
[ENOSR]	Insufficient stream resources are available. The socket cannot be created until sufficient resources are freed.
[ENXIO]	Streams open failed.
[EPROTO]	The commands required prior to this one were not run or did not succeed.

SEE ALSO

accept(3I), bind(3I), connect(3I), getsockname(3I), getsockopt(3I), listen(3I), recv(3I), send(3I), shutdown(3I), protocols(5).
ioctl(2), read(2), write(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

An Introductory 4.3BSD Interprocess Communication Tutorial.
An Advanced 4.3BSD Interprocess Communication Tutorial.

RETURN VALUES

A −1 is returned if an error occurs; otherwise the return value is a descriptor referencing the socket.

NAME

spipe – create a streams pipe

SYNOPSIS

spipe(fildes)
int fildes[2];

DESCRIPTION

The call creates an I/O mechanism called a STREAMS pipe. The file descriptors returned can be used in *read* and *write* operations. When the pipe is written using the descriptor *fildes*[1], a *read* using the descriptor *fildes*[0] will pick up the data.

It is assumed that after the STREAMS pipe has been set up, two (or more) cooperating processes (created by subsequent *fork* calls) will pass data through the STREAMS pipe with *read*(2) and *write*(2) calls.

read calls on an empty STREAMS pipe (no buffered data) with only one end (all *write* file descriptors closed) return an end-of-file.

STREAMS pipes are really a special case of the *open*(2) call and, in fact, are implemented as such in the system.

SEE ALSO

sh(1) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual*.
fork(2), read(2), open(2), write(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual*.

DIAGNOSTICS

The *spipe* call will fail if:

[EMFILE]	Too many descriptors are active.
[ENFILE]	The system file table is full.

RETURN VALUES

The function value zero is returned if the *spipe* was created; −1 if an error occurred.

WARNINGS

The INTERACTIVE Network Connection Facilities must be installed for *spipe* to function properly.

NAME

syslog, openlog, closelog, setlogmask – control system log

SYNOPSIS

#include <syslog.h>

openlog(ident, logopt, facility)
char *ident;

syslog(priority, message, parameters ...)
char *message;

closelog()

setlogmask(maskpri)

DESCRIPTION

syslog arranges to write *message* onto the system log maintained by *syslogd*(1M). The message is tagged with *priority*. The message looks like a *printf*(3) string except that **%m** is replaced by the current error message (collected from *errno*). A trailing newline is added if needed. This message will be read by *syslogd*(1M) and written to the system console, log files, or forwarded to *syslogd* on another host, as appropriate.

Priorities are encoded as a *facility* and a *level*. The facility describes the part of the system generating the message. The level is selected from an ordered list:

LOG_EMERG	A panic condition. This is normally broadcast to all users.
LOG_ALERT	A condition, such as a corrupted system database, that should be corrected immediately.
LOG_CRIT	Critical conditions, e.g., hard device errors.
LOG_ERR	Errors.
LOG_WARNING	Warning messages.
LOG_NOTICE	Conditions that are not error conditions but should possibly be handled specially.
LOG_INFO	Informational messages.
LOG_DEBUG	Messages that contain information normally of use only when debugging a program.

If *syslog* cannot pass the message to *syslogd*, it will attempt to write the message on **/dev/console** if the LOG_CONS option is set (see below).

If special processing is needed, *openlog* can be called to initialize the log file. The parameter *ident* is a string that is prepended to every message. *logopt* is a bit field indicating logging options. Current values for *logopt* are:

LOG_PID	log the process ID with each message: useful for identifying instantiations of daemons.
LOG_CONS	Force writing messages to the console if unable to send it to *syslogd*. This option is safe to use in daemon processes that have no controlling

terminal since *syslog* will fork before opening the console.

LOG_NDELAY — Open the connection to *syslogd* immediately. Normally, the *open* is delayed until the first message is logged. Useful for programs that need to manage the order in which file descriptors are allocated.

LOG_NOWAIT — Don't wait for children forked to log messages on the console. This option should be used by processes that enable notification of child termination via SIGCHLD, as *syslog* may otherwise block waiting for a child whose exit status has already been collected.

The *facility* parameter encodes a default facility to be assigned to all messages that do not have an explicit facility encoded:

LOG_KERN — Messages generated by the kernel. These cannot be generated by any user processes.

LOG_USER — Messages generated by random user processes. This is the default facility identifier if none is specified.

LOG_MAIL — The mail system.

LOG_DAEMON — System daemons, such as *ftpd*(1M), *gated*(1M), etc.

LOG_AUTH — The authorization system: *login*(1), *su*(1), *getty*(1M), etc.

LOG_LOCAL0 — Reserved for local use (similarly for LOG_LOCAL1 through LOG_LOCAL7).

closelog can be used to close the log file.

setlogmask sets the log priority mask to *maskpri* and returns the previous mask. Calls to *syslog* with a priority not set in *maskpri* are rejected. The mask for an individual priority *pri* is calculated by the macro LOG_MASK(*pri*); the mask for all priorities up to and including *toppri* is given by the macro LOG_UPTO(*toppri*). The default allows all priorities to be logged.

Examples

```
syslog(LOG_ALERT, "who: internal error 23");

openlog("ftpd", LOG_PID, LOG_DAEMON);
setlogmask(LOG_UPTO(LOG_ERR));
syslog(LOG_INFO, "Connection from host %d", CallingHost);

syslog(LOG_INFO|LOG_LOCAL2, "foobar error: %m");
```

SEE ALSO

logger(1C), syslogd(1M).

NAME

hosts – host name database

DESCRIPTION

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

official host name
Internet address
aliases

Items are separated by any number of blanks and/or tab characters. A **#** indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file.

When using the name server *named*(1M), this file provides a backup when the name server is not running. For the name server, it is suggested that only a few addresses be included in this file. These include address for the local interfaces that *ifconfig*(1M) needs at boot time and a few machines on the local network.

This file may be created from the official host database maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts. As the database maintained at NIC is incomplete, use of the name server is recommended for sites on the DARPA Internet.

Network addresses are specified in the conventional . notation using the *inet_addr*() routine from the Internet address manipulation library, *inet*(3I). Host names may contain any printable character other than a field delimiter, new-line character, comment character, or uppercase letters.

FILES

/etc/hosts

SEE ALSO

ifconfig(1M), named(1M), ghostnamed(3I).

"Name Server Operations Guide for BIND."

NAME

networks – network name database

DESCRIPTION

The **networks** file contains information regarding the known networks which comprise the DARPA Internet. For each network a single line should be present with the following information:

official network name
network number
aliases

Items are separated by any number of blanks and/or tab characters. A **#** indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file. This file is normally created from the official network database maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network numbers may be specified in the conventional . notation using the *inet_network()* routine from the Internet address manipulation library, *inet*(3I). Network names may contain any printable character other than a field delimiter, new-line character, or comment character.

FILES

/etc/networks

SEE ALSO

getnetent(3I).

BUGS

A name server should be used instead of a static file.

NAME

printcap – printer capability database

SYNOPSIS

/etc/printcap

DESCRIPTION

printcap is a simplified version of the *termcap* database used to describe line printers. The spooling system accesses the *printcap* file every time it is used, allowing dynamic addition and deletion of printers. Each entry in the database is used to describe one printer.

The default printer is normally **lp**, though the environment variable PRINTER may be used to override this. Each spooling utility supports an option, **–P***printer*, to allow explicit naming of a destination printer.

Refer to the "Line Printer Spooler Manual" for a complete discussion on how to set up the database for a given printer.

Capabilities

Each entry in *printcap* describes a single printer and consists of a number of fields separated by colons (:). The first entry for each printer specifies the names that are known for the printer, separated by pipe symbols (|). The first name is typically a shorthand name to be used by users, while the last name given should be a long name fully identifying the printer; any other names are understood as synonyms for the printer name. All names but the last should be in lowercase letters and contain no blanks; the last name may contain uppercase letters and blanks for readability.

All *printcap* capabilities are specified using two-character codes and may be of three basic types:

Boolean

Capabilities that indicate that the printer has some particular feature. Boolean capabilities are simply written between the two colons, i.e., ":sh:" would indicate that short headers should be used for this printer.

Numeric

Capabilities that supply some sort of numeric value, such as the width or number of lines per page. They are expressed as the name, followed by the "#" sign, followed by the numeric value. An example would be ":pl#66:" to indicate that the printer prints 66 lines per page.

String

Capabilities that specify items such as file names or character sequences to be passed to the printer. These are specified as the name, followed by the "=" sign, then the string value. An example is ":sd=/dev/lp0:" to specify that a printer is connected to **/dev/lp0**.

An example:

```
lp|Panasonic 4450 laser printer:\
    :lp=/dev/tty04:\
    :tt=B19200+CS8+IXON+IXOFF+CREAD+CLOCAL\
    :lf=/usr/adm/lpd-errs:\
    :sh:sd=/usr/spool/lpd:
```

This example is for the system default printer (**lp**), with a longer description indicating that it is a Panasonic 4450 laser printer. The **lp=** capability shows it is connected to the device **/dev/tty04**, the **tt=** field specifies the communication parameters to use, and the **lf=** entry causes any errors to be logged to the file **/usr/adm/lpd-errs**. Additionally, the *sh* boolean capability indicates that only short headers should be printed, and all jobs sent to this printer will be spooled to the directory **/usr/spool/lpd**, as specified by **sd=**.

This is a complete list of all printcap capabilities used by *lpd*:

Name	Type	Default	Description
af	str	NULL	name of accounting file
br	num	none	if lp is a tty, set the baud rate (ioctl call)
cc	num	0	if lp is a tty, bits to clear in c_cflag
cf	str	NULL	cifplot data filter
cs	num	0	if lp is a tty, bits to set in c_cflag
df	str	NULL	tex data filter (DVI format)
ff	str	\f	string to send for a form feed
fo	bool	false	print a form feed when device is opened
gf	str	NULL	graph data filter (plot(3X) format)
hl	bool	false	print the burst header page last
ic	num	0	if lp is a tty, bits to clear in c_iflag
if	str	NULL	name of text filter that does accounting
is	num	0	if lp is a tty, bits to set in c_iflag
lf	str	/dev/console	error logging file name
lo	str	lock	name of lock file
lp	str	/dev/lp	device name to open for output
mx	num	1000	maximum file size (in BUFSIZ blocks), zero = unlimited
nd	str	NULL	next directory for list of queues (unimplemented)
nf	str	NULL	ditroff data filter (device-independent troff)
oc	num	0	if lp is a tty, bits to clear in c_oflag
of	str	NULL	name of output filtering program
os	num	0	if lp is a tty, bits to set in c_oflag
pc	num	200	price per foot or page in hundredths of cents
pl	num	66	page length (in lines)
pw	num	132	page width (in characters)
px	num	0	page width in pixels (horizontal)
py	num	0	page length in pixels (vertical)
rf	str	NULL	filter for printing FORTRAN-style text files

rg	str	NULL	restricted group; only members of group allowed access
rm	str	NULL	machine name for remote printer
rp	str	lp	remote printer name argument
rs	bool	false	restrict remote users to those with local accounts
rw	bool	false	open the printer device for reading and writing
sb	bool	false	short banner (one line only)
sc	bool	false	suppress multiple copies
sd	str	/usr/spool/lpd	spool directory
sf	bool	false	suppress form feeds
sh	bool	false	suppress printing of burst page header
st	str	status	status file name
tf	str	NULL	troff data filter (C/A/T phototypesetter)
tr	str	NULL	trailer string to print when queue empties
tt	str	NULL	getydefs-like string specifying values to set and clear in termio structure
vf	str	NULL	raster image filter

Filters

The *lpd*(1M) daemon creates a pipeline of filters to process files for various printer types. The filters selected depend on the flags passed to *lpr*(1). The pipeline set up is:

−p	pr | if	regular text + pr(1)
none	if	regular text
−c	cf	cifplot
−d	df	DVI (tex)
−g	gf	plot(3X)
−n	nf	ditroff
−f	rf	Fortran
−t	tf	troff
−v	vf	raster image

The **if** filter is invoked with arguments:

if [**−c**] **−w**width **−l**length **−i**indent **−n** login **−h** host acct-file

The **−c** flag is passed only if the **−l** flag (pass control characters literally) is specified to *lpr*. *width* and *length* specify the page width and length (from **pw** and **pl** respectively) in characters. The **−n** and **−h** parameters specify the login name and host name of the owner of the job respectively. *acct-file* is passed from the **af** *printcap* entry.

If no **if** is specified, **of** is used instead, with the distinction that **of** is opened only once, while **if** is opened for every individual job. Thus, **if** is better suited to performing accounting. The **of** is only given the *width* and *length* flags.

All other filters are called as:

filter **−x**width **−y**length **−n** login **−h** host acct-file

where *width* and *length* are represented in pixels, specified by the **px** and **py** entries, respectively.

All filters take **stdin** as the file, **stdout** as the printer, may log to **stderr** or use *syslog*(3I), and must not ignore SIGINT.

Logging

Error messages generated by the line printer programs themselves (that is, the *lp** programs) are logged by *syslog*(3I) using the *lpr* facility. Messages printed on **stderr** associated with one of the filters are sent to the corresponding **lf** file. The filters may, of course, use *syslog* themselves.

Error messages sent to the console have a carriage return and a line-feed character appended to them, rather than just a line-feed character.

SEE ALSO

lpc(1M), lpd(1M), lpr(1), lpq(1), lprm(1), syslog(3I).
tty(7) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*
plot(3X) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*
"Line Printer Spooler Manual."

NAME

protocols – protocol name database

DESCRIPTION

The **protocols** file contains information regarding the known protocols used in the DARPA Internet. For each protocol a single line should be present with the following information:

official protocol name
protocol number
aliases

Items are separated by any number of blanks and/or tab characters. A **#** indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file.

Protocol names may contain any printable character other than a field delimiter, new-line character, or comment character.

FILES

/etc/protocols

SEE ALSO

getprotoent(3I).

BUGS

A name server should be used instead of a static file.

NAME

resolver – resolver configuration file

SYNOPSIS

/etc/resolv.conf

DESCRIPTION

The *resolver* configuration file contains information that is read by the *resolver* routines the first time they are invoked by a process. The file contains a list of name-value pairs that provide various types of resolver information.

On a normally configured system this file should not be necessary. The only name server to be queried will be on the local machine and the domain name is retrieved from the system.

The different configuration options are:

nameserver
: followed by the Internet address (in dot notation) of a name server that the resolver should query. At least one name server should be listed. Up to MAXNS (currently three) name servers may be listed; the resolver library queries them in the order listed. If no **nameserver** entries are present, the default is to use the name server on the local machine. (The algorithm used is to try a name server, and if the query times out, try the next, until out of name servers. Repeat, trying all the name servers until a maximum number of retries are made).

domain followed by a domain name that is the default domain to append to names that do not have a dot in them. If no **domain** entries are present, the **root domain** is assumed.

nonameserver
: with or without a value. This stops the *resolver* routines from contacting any name server and forces *gethostbyname* and *gethostbyaddr* (see *ghostnamed*(3I)) to use the **/etc/hosts** file. This allows use of the host table without recompiling network applications or waiting for the *resolver* routines to time out.

The name value pair must appear on a single line, and the keyword, e.g., **nameserver**, must start the line. The value follows the keyword, separated by white space.

FILES

/etc/resolv.conf

SEE ALSO

named(1M), ghostnamed(3I), resolver(3I).

"Name Server Operations Guide for BIND."

NAME

services – service name database

DESCRIPTION

The **services** file contains information regarding the known services available in the DARPA Internet. For each service a single line should be present with the following information:

official service name
port number
protocol name
aliases

Items are separated by any number of blanks and/or tab characters. The port number and protocol name are considered a single *item*; a / is used to separate the port and protocol, e.g., *512/tcp*. A **#** indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file.

Service names may contain any printable character other than a field delimiter, new-line character, or comment character.

FILES

/etc/services

SEE ALSO

getservent(3I).

BUGS

A name server should be used instead of a static file.

NAME

socket – socket configuration file

SYNOPSIS

/etc/socket.conf

DESCRIPTION

The socket configuration file contains information that is read by the socket library routine each time it is invoked by a process. The file is designed to be human readable and contains a list of streams devices that are to be opened and streams modules that are to be pushed on top of the opened devices for different socket domains, types, and protocols.

On a normally configured system this file should not need to be changed. It is installed with the information required for the particular network configuration that has been installed with it.

The file consists of lines with fields separated by white space (space or tab characters). Each line constitutes one entry. The first three fields on each line are numerical and contain the **socket domain, socket type,** and **protocol** corresponding to the respective parameters to the *socket*(3I) call. The fourth field is the complete name of the device to open for that particular combination of **domain, type**, and **protocol.** Any additional fields are names of streams modules that should be pushed on the stream above the opened driver (see *streamio*(7)).

Here is the entry for TCP:

```
#  domain      type                protocol    device      modules
#     AF_INET    SOCK_STREAM
   2           1                   0           /dev/tcp    socket
```

If a line begins with a number sign (#), that line is treated as a comment and is skipped. A zero in any of the first three fields will match any value of that parameter. The first line that matches the calling parameters is used. A list of the possible values for the first two fields are contained in the include file **/usr/include/sys/socket.h.** The **domain** field is called the **address family** in that file. The **protocol** field is interpreted according to the value of the **domain** and **type** fields. It can usually be left zero. If no module names are listed, the *socket*(7) module is pushed.

FILES

/etc/socket.conf
/usr/include/sys/socket.h

SEE ALSO

socket(3I), socket(7).
streamio(7) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*

NAME

networking – introduction to networking facilities

SYNOPSIS

#include <sys/socket.h>
#include <net/route.h>
#include <net/if.h>

DESCRIPTION

This section briefly describes the networking facilities available in the system. Documentation in this part of Section 7 is divided into three areas: protocol families (domains), protocols, and network interfaces.

All network protocols are associated with a specific protocol family. A protocol family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. A protocol family may support multiple methods of addressing, though the current protocol implementations do not. A protocol family is normally comprised of a number of protocols, one per *socket*(3I) type. It is not required that a protocol family support all socket types. A protocol family may contain multiple protocols supporting the same socket abstraction.

A protocol supports one of the socket abstractions detailed in *socket*(3I). A specific protocol may be accessed either by creating a socket of the appropriate type and protocol family or by requesting the protocol explicitly when creating a socket. Protocols normally accept only one type of address format, usually determined by the addressing structure inherent in the design of the protocol family/network architecture. Certain semantics of the basic socket abstractions are protocol specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide non-standard facilities or extensions to a mechanism. For example, a protocol supporting the SOCK_STREAM abstraction may allow more than one byte of out-of-band data to be transmitted per out-of-band message.

A network interface is similar to a device interface. Network interfaces comprise the lowest layer of the networking subsystem, interacting with the actual transport hardware. An interface may support one or more protocol families and/or address formats. The **SYNOPSIS** section of each network interface entry gives a sample specification of the related drivers. The **DIAGNOSTICS** section lists messages which may appear on the console and/or in the system error log, **/usr/adm/messages** (see *syslogd*(1M)), due to errors in device operation.

All network-related *ioctl*s are in the stream I_STR format (see *streamio*(7)).

Protocols

The system currently supports the DARPA Internet protocols. Raw socket interfaces are provided to the IP protocol layer of the DARPA Internet. Consult the appropriate manual entries in this section for more information regarding the support for each protocol family.

Addressing

Associated with each protocol family is an address format. The following address formats are used by the system (and additional formats are defined for possible future implementation):

```
#define   AF_INET      2        /* internetwork: UDP, TCP, etc. */
```

Routing

The network facilities provide limited packet routing. A simple set of data structures comprise a "routing table" used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. A user process, the routing daemon, maintains this database with the aid of two socket-specific *ioctl*(2) commands, SIOCADDRT and SIOCDELRT. These commands allow the addition and deletion, respectively, of a single routing table entry. Routing table manipulations may only be carried out by superuser.

A routing table entry has the following form, as defined in **<net/route.h>**;

```
struct rtentry {
        u_long          rt_hash;
        struct          sockaddr rt_dst;
        struct          sockaddr rt_gateway;
        short           rt_flags;
        short           rt_refcnt;
        u_long          rt_use;
        struct          ifnet *rt_ifp;
};
```

with *rt_flags* defined from,

```
#define   RTF_UP          0x1    /* route usable */
#define   RTF_GATEWAY     0x2    /*destination is a gateway */
#define   RTF_HOST        0x4    /* host entry (net otherwise) */
#define   RTF_DYNAMIC     0x10   /* created dynamically */
                                 /* (by redirect) */
```

There are three types of routing table entries: for a specific host, for all hosts on a specific network, and for any destination not matched by entries of the first two types (a "wildcard" route). When the system is booted and addresses are assigned to the network interfaces, each protocol family installs a routing table entry for each interface when it is ready for traffic. Normally the protocol specifies the route through each interface as a "direct" connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface is requested to address the packet to the gateway listed in the routing entry, i.e., the packet is forwarded.

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (*rt_refcnt* is nonzero), the routing entry will be marked down and removed from the routing table, but the resources associated with it will not be reclaimed until all references to it are released. The routing code

returns EEXIST if requested to duplicate an existing entry, ESRCH if requested to delete a non-existent entry, or ENOBUFS if insufficient resources were available to install a new route. User processes read the routing tables through the **/dev/kmem** device. The *rt_use* field contains the number of packets sent along the route.

When routing a packet, the kernel will first attempt to find a route to the destination host. Failing that, a search is made for a route to the network of the destination. Finally, any route to a default (wildcard) gateway is chosen. If multiple routes are present in the table, the first route found will be used. If no entry is found, the destination is declared to be unreachable.

A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

Interfaces

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, *lo*(7), do not.

The following *ioctl* calls may be used to manipulate network interfaces. The *ioctl* is made on a socket (typically of type SOCK_DGRAM) in the desired domain. Unless specified otherwise, the request takes an *ifreq* structure as its parameter. This structure has the following form:

```
struct  ifreq   {
#define IFNAMSIZ 16
        char ifr_name[IFNAMSIZ];  /* if name, e.g. "ec0" */
        union   {
                struct      sockaddr ifru_addr;
                struct      sockaddr ifru_dstaddr;
                struct      sockaddr ifru_broadaddr;
                short       ifru_flags;
                int         ifru_metric;
                int         ifru_lindex;
                caddr_t     ifru_data;
                struct      ifstats ifru_stats;
                struct      hwaddr ifru_hwaddr;
        }       ifr_ifru;

#define ifr_addr        ifr_ifru.ifru_addr      /* address */
#define ifr_dstaddr     ifr_ifru.ifru_dstaddr   /* other end of */
                                                /* p-to-p link */
#define ifr_broadaddr   ifr_ifru.ifru_broadaddr /* broadcast */
                                                /* address */
#define ifr_flags       ifr_ifru.ifru_flags     /* flags */
#define ifr_metric      ifr_ifru.ifru_metric    /* routing metric */
#define ifr_lindex      ifr_ifru.ifru_lindex    /* link index for if */
                                                /* name set */
```

```
#define ifr_data        ifr_ifru.ifru_data      /* for use by */
                                                /* interface */
#define ifr_stats       ifr_ifru.ifru_stats     /* interface */
                                                /* statistics */
#define ifr_hwaddr      ifr_ifru.ifru_hwaddr    /* interface physical */
                                                /* address */
};
```

SIOCSIFADDR
: Set interface address for protocol family. Following the address assignment, the "initialization" routine for the interface is called.

SIOCGIFADDR
: Get interface address for protocol family.

SIOCSIFDSTADDR
: Set point to point address for protocol family and interface.

SIOCGIFDSTADDR
: Get point to point address for protocol family and interface.

SIOCSIFBRDADDR
: Set broadcast address for protocol family and interface.

SIOCGIFBRDADDR
: Get broadcast address for protocol family and interface.

SIOCGPHYSADDR
: Retrieve an interface physical address.

SIOCSIFFLAGS
: Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified; some interfaces may be reset so that incoming packets are no longer received. When marked up again, the interface is reinitialized.

SIOCGIFFLAGS
: Get interface flags.

SIOCSIFMETRIC
: Set interface routing metric. The metric is used only by user-level routers.

SIOCGIFMETRIC
: Get interface metric.

SIOCGIFCONF
: Get interface configuration list. This request takes an *ifconf* structure (see below) as a value-result parameter. The *ifc_len* field should be initially set to the size of the buffer pointed to by *ifc_buf*. On return, it will contain the length, in bytes, of the configuration list.

  This *ioctl* is not 100 percent compatible with the BSD implementation. In all cases, *ioctl* calls to the TCP/IP modules must conform to the I_STR format described in *streamio*(7).

```
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all accessible networks).
 */
struct  ifconf      {
        int         ifc_len;            /* size of associated buffer */
        union       {
                    struct  ifreq ifcu_req[1];
        }ifc_ifcu;
#define ifc_buf     ifc_ifcu.ifcu_buf   /* buffer address */
#define ifc_req     ifc_ifcu.ifcu_req   /* array of structures */
                                        /* returned */
};
```

Rather than using a pointer to a buffer, the *ifconf* structure must be allocated with sufficient space to hold all of the data desired. The *ifc_len* member must contain the amount of space allocated for the data. The *ifcu_req* member is an array that indexes all of the data.

SEE ALSO

route(1M), syslog(1M), socket(3I).
streamio(7) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*
ioctl(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

NAME

arp – Address Resolution Protocol

SYNOPSIS

#include <net/i_arp.h>
s = open("/dev/arp", O_RDWR);

DESCRIPTION

arp is a protocol used to dynamically map between 32-bit Internet addresses and 48-bit Ethernet addresses. It is implemented as a multiplexing STREAMS driver and is linked between network layer drivers and link layer drivers by *netd*(1M). *arp* communicates with both upper and lower STREAMS modules through the USL Link Provider Interface. It is not specific to Internet protocols or to 10 MB/s Ethernet, but this implementation currently supports only that combination.

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

arp has been extended to implement transparent subnet gateways by the "Proxy *arp*" or "*arp* hack" technique. This interface is enabled through *ifconfig*(1M).

To facilitate communications with systems which do not use *arp*, *ioctl*s are provided to enter and delete entries in the Internet-to-Ethernet tables. The structures used by these *ioctl*s are defined in the include file **<net/if_arp.h>**. The *ioctl*s may be applied to any socket descriptor *s*, but *arp* table entries may be set only by the superuser. Usage:

```
#include <sys/types.h>
#include <sys/sioctl.h>
#include <sys/socket.h>
#include <net/if_arp.h>
struct strioctl strioctl;
struct arpreq arpreq;
struct arpent arpent;
struct arptimtab arptimtab;

strioctl.ic_cmd = SIOCGARP;
strioctl.ic_timout = 0;
strioctl.ic_len = sizeof (struct arpreq);
strioctl.ic_dp = (char *) &arpreq;
ioctl (s, I_STR, (caddr_t)&strioctl);
```

The *ioctl*s SIOCSARP, SIOCGARP, and SIOCDARP use the *arpreq* structure as an argument. SIOCSARP sets an *arp* table entry, SIOCGARP gets an *arp* table entry, and SIOCDARP deletes an *arp* table entry. The SIOCSARP *ioctl* may set the following flag bits:

```
#define ATF_PERM  0x04  /* permanent entry */
#define ATF_PUBL  0x08  /* publish (respond for other host) */
```

ATF_PERM causes the entry to be permanent if the *ioctl* call succeeds. The peculiar nature of the *arp* tables may cause the *ioctl* to fail if more than eight (permanent) Internet host addresses hash to the same slot. ATF_PUBL specifies that the *arp* code should respond to *arp* requests for the indicated host coming from other machines. This allows a host to act as an *arp* server, which may be useful in convincing an *arp*-only machine to talk to a non-*arp* machine. The address family for the *arp_pa* sockaddr must be AF_INET; for the *arp_ha* sockaddr it must be AF_UNSPEC.

The SIOCGENT *ioctl* uses the *arpent* structure as an argument and gets an *arp* table entry by index from the kernel. The return value in the *arpent* structure's *ae_index* field is the number of entries in the *arp* table. This is convenient for dumping an *arp* table of unknown length.

The *ioctl*s SIOCGTIM and SIOCSTIM use the *arptimtab* structure as an argument. SIOCGTIM gets the *arp* timeout parameters, and SIOCSTIM sets the timeout parameters. These parameters are:

```
int arpt_interval;      /* interval between timer runs */
int arpt_killcom;       /* timeout to kill a complete entry */
int arpt_killinc;       /* timeout to kill an incomplete entry */
```

arpt_interval is the amount of time that the *arp* timer sleeps between running. The default value is 60 seconds. *arpt_killcom* is the amount of time before the timer kills a table entry. The default value is 20 minutes and the timer is reset to zero every time a packet is sent to or received from that host. *arpt_killinc* is the amount of time before the timer kills a table entry for a host that never responded to an *arp* request. The default value is 3 minutes.

arp sends out a request for the local host's address when it starts, and watches passively for hosts impersonating the local host, i.e., a host that responds to an *arp* mapping request for the local host's address.

Configuration

The **space.c** file provides a mechanism for changing operating parameters of the *arp* driver. The number of upper and lower streams, timeout parameters, and *arp* table size can be changed.

SEE ALSO

arp(1M), ifconfig(1M), netd(1M), ec(7), inet(7), ip(7), llc(7), wd(7).

"An Ethernet Address Resolution Protocol," RFC826, Dave Plummer, Network Information Center, SRI.

"Using ARP to Implement Transparent Subnet Gateways," RFC1027, Carl-Mitchell Smoot and John S. Quarterman, Texas Internet Consulting.

DIAGNOSTICS

duplicate IP address!! sent from ethernet address: %x:%x:%x:%x:%x:%x.
arp has discovered another host on the local network which responds to mapping requests for its own Internet address.

BUGS

arp packets on the Ethernet need only 42 bytes of data but are padded to the minimum required by the lower stream protocol layer. The smallest legal Ethernet packet is 60 bytes (not including CRC).

NAME

ec – 3COM IE Ethernet Controller interface

DESCRIPTION

The *ec* interface provides STREAMS access to the 3COM IE family of 10 MB Ethernet controllers.

The driver is provided with *IEEE 802.2 LLC Class 1* and *Ethernet* encapsulation of data by the *llc*(7) driver. All communication with the *ec* driver is done via function calls between *ec* and *llc*.

The *ec* driver supports promiscuous mode, as configured by *llc*. It receives and understands "trailer" packets but does not generate them. Loopback is handled between the *ec* and *llc* drivers, even though the 3COM board is capable of loopback.

Configuration

The **space.c** file provides a mechanism for specifying the IE board operating parameters to the driver. If a different I/O port base address or interrupt vector are necessary, the **space.c** file should be updated accordingly.

SEE ALSO

arp(7), ip(7), llc(7).

DIAGNOSTICS

ecinit: CSR read failed from 3COM board: %x.
The driver was unable to read the expected data from the control/status register on the board.

ecinit: XMT read failed from 3COM board: %x.
The driver was unable to read the expected data from the transmit register on the board.

ecinit: RCV read failed from 3COM board: %x.
The driver was unable to read the expected data from the receive register on the board.

ecinit: 3COM board returns bad ethernet address.
The driver was unable to read a legal 3COM Ethernet address from the prom on the board.

3COM board not present or in error.
The driver refuses to initialize the board because one or more of the previous errors suggests that the board is not working.

WARNING

The COM 3C501 board and *ec* driver are not capable of high performance operation. This is due to a limitation in the hardware and not a bug in the driver. Use with INTERACTIVE NFS is discouraged.

NAME

el – 3COM 3C503 Etherlink II Ethernet Controller interface

DESCRIPTION

The *el* interface provides STREAMS access to the 3COM 3C503 10 MB Ethernet controller.

The driver is provided with *IEEE 802.2 LLC Class 1* and *Ethernet* encapsulation of data by the *llc*(7) driver. All communication with the *el* driver is done via function calls between *el* and *llc*.

The *el* driver supports promiscuous mode, as configured by *llc*. It receives and understands "trailer" packets, but does not generate them. Loopback is handled between the *el* and *llc* drivers, even though the 3C503 board is capable of loopback.

Configuration

The **space.c** file provides a mechanism for specifying the 3C503 board operating parameters to the driver. If a different I/O port base address, interrupt vector or shared memory base address are necessary, the **space.c** file should be updated accordingly by means of the *sysadm* command.

SEE ALSO

arp(7), ip(7), llc(7).

DIAGNOSTICS

elinit: 3C503 board returned bad ethernet address.
The driver was unable to read a legal 3COM Ethernet address from the prom on the board. This suggests that the board is not present or configured incorrectly.

elinit: Bad IRQ for 3C503 board: %d.
The Interrupt Request Vector chosen for the board during setup is not a legal value for this board. A legal value must be put in the **space.c** file with the **sysadm** command.

elintr: irq wrong: %x.
The driver was interrupted by a board that is not configured into the kernel.

3C503 board not present or in error.
The driver refuses to initialize the board because previous errors suggest that the board is not working.

NAME

icmp – Internet Control Message Protocol

SYNOPSIS

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

s = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
```

or

```
#include <sys/tiuser.h>
s = t_open("/dev/ip", O_RDWR, NULL);
```

DESCRIPTION

The Internet Control Message Protocol (ICMP) is the error and control message protocol used by IP and the Internet protocol family. It may be accessed through a raw socket for network monitoring and diagnostic functions. Access via TLI is also possible by using **/dev/ip** as the STREAM device. The *proto* parameter to the socket call to create an ICMP socket is obtained from *getprotobyname* (see *getprotoent*(3I)). ICMP sockets are connectionless and are normally used with the *sendto* and *recvfrom* calls, though the *connect*(2) call may also be used to fix the destination for future packets (in which case the *read*(2) or *recv*(3I) and *write*(2) or *send*(3I) system calls may be used).

With TLI, the stream must be bound to IPPROTO_ICMP. The address is a single byte in length. The *t_sndudata*(3N) and *t_rcvudata*(3N) calls may then be used.

Outgoing packets currently must have an IP header prefixed to them (based on the destination address). Incoming packets are received with the IP header and options intact.

SEE ALSO

recv(3I), send(3I), intro(7N), inet(7), ip(7).
t_bind(3N), t_open(3N), t_rcvudata(3N), t_sndudata(3N) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

DIAGNOSTICS

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

[EISCONN] when trying to establish a connection on a socket that already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;

[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket has not been connected;

[ENOBUFS] when the system runs out of memory for an internal data structure;

[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for which no network interface exists.

NAME

inet – Internet protocol family

SYNOPSIS

```
#include <sys/types.h>
#include <sys/bsdtypes.h>
#include <netinet/in.h>
```

DESCRIPTION

The Internet protocol family is a collection of protocols layered atop the Internet Protocol (IP) transport layer and utilizing the Internet address format. The Internet family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types as well as the T_COTS and T_CLTS types of transport; the SOCK_RAW interface provides access to the IP protocol.

Addressing

Internet addresses are four-byte quantities, stored in network standard format (on the 386 these are word and byte reversed). The include file **<netinet/in.h>** defines this address as a discriminated union.

Sockets and TLI streams bound to the Internet protocol family utilize the following addressing structure:

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

Sockets may be created with the local address INADDR_ANY to effect wildcard matching on incoming messages. The address in a *connect*(3I), *sendto*(3I), *t_connect*(3N), or *t_sndudata*(3N) call may be given as INADDR_ANY to mean "this host." The distinguished address INADDR_BROADCAST is allowed as a shorthand for the broadcast address on the primary network if the first network configured supports broadcast.

Protocols

The Internet protocol family is comprised of the IP transport protocol, Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). TCP is used to support the SOCK_STREAM and T_COTS abstractions, while UDP is used to support the SOCK_DGRAM and T_CLTS abstractions. A raw interface to IP is available by creating an Internet socket of type SOCK_RAW in the socket environment or directly opening **/dev/ip** in the TLI environment. The ICMP message protocol is accessible from a raw socket or from opening **/dev/ip**.

The 32-bit Internet address contains both network and host parts. It is frequency-encoded; the most-significant bit is clear in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses use the high-order 16 bits as the network field, and Class C addresses have a 24-bit network part. Sites with a cluster of local networks and a connection to the DARPA Internet may choose to use a single network number for the cluster; this is done by using subnet addressing. The local (host) portion of the address is further

subdivided into subnet and host parts. Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following *ioctl*(2) commands on a datagram socket in the Internet domain; they have the same form as the SIOCIFADDR command (see *intro*(7N)).

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

SIOCGIFNETMASK Get interface network mask.

SEE ALSO

socket(3I), intro(7N), icmp(7), ip(7), tcp(7), udp(7).
ioctl(2) in the *INTERACTIVE SDS Guide and Programmer's Reference Manual.*

WARNINGS

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

NAME

ip – internet protocol

SYNOPSIS

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

s = socket(AF_INET, SOCK_RAW, protocol);
or **#include <sys/niuser.h>**
#include <sys/ip.h>
#include <netinet/in.h>

t = t_open("/dev/ip", O_RDWR);

DESCRIPTION

ip is a STREAMS multiplexing driver (both upper and lower), which implements the Department of Defense Internet Protocol and Internet Control Message Protocol. It utilizes the Network Interface protocol to communicate with an upstream user. *ip* only implements a *connectionless* protocol and thus uses the same model as the TLI T_CLTS interface. The Network Provider Interface is basically the same as the TLI connectionless model.

Each upper STREAM can be bound to a unique Network Service Access Point (NSAP), which is mapped into the *ip* protocol identifier. This NSAP is an eight-bit quantity with values defined for each higher layer protocol as defined in RFC1010 – Assigned Numbers. There is a wildcard value (IPPROTO_RAW), which allows a STREAM to get copies of all messages that aren't directed to a protocol bound to some other STREAM.

A number of data link level device STREAMS may be linked below the *ip* driver to provide network access. These STREAMS must understand the Link Interface protocol.

ip normally determines the name of the lower device by looking at the *qinfo* structure at I_LINK time. The name is used in later configuration specific ioctl calls. If the *arp* device is linked below the *ip* driver, a new name should be assigned with the SIOCSIFNAME *ioctl* in order to provide better identification for use with the configuration ioctl calls. *ip* also treats the case of the loopback driver, **/dev/lo**, as a special case.

In addition to the Network Interface protocol, there is provision for passing control information between *ip* and higher layer protocols which is implemented using the M_CTL message format. This is only of interest to other STREAMS modules/drivers and not to user-level processes.

Currently, the user must provide a valid *ip* header, but the critical fields are filled in by the *ip* driver. This may change in later releases.

SEE ALSO

netd(1M), arp(7), ec(7), llc(7), socket(7), tcp(7), ub(7), udp(7), wd(7).

NAME

llc – Logical Link Control/Ethernet Program Interface

DESCRIPTION

The IEEE 802.3/Ethernet class of network interfaces are supported under STREAMS via the Link Provider Interface (LPI) protocol. Several forms of data encapsulation are supported with this interface, of which two forms are supported for the IEEE 802.3/Ethernet classes of device: *IEEE 802.2 Logical Link Control Class 1* and *Ethernet*.

The *llc* Class 1 protocol is defined in the ANSI/IEEE standard, while the Ethernet protocol is as defined in the DEC-Intel-XEROX Ethernet Version 2 specification. The encapsulation is chosen by the Service Access Point (SAP) bound to. If the SAP value to be bound is a value larger than 255 (0xFF), then the type of encapsulation is taken to be the Ethernet style rather than *llc* Class 1. Rather than a SAP value, the value passed will be used as the *Ethernet type* field for the stream.

There are ten LPI primitives, four user initiated and six driver initiated. The user-initiated primitives are DL_INFO_REQ, DL_BIND_REQ, DL_UNBIND_REQ, and DL_UNITDATA_REQ. The driver-initiated primitives are DL_INFO_ACK, DL_BIND_ACK, DL_ERROR_ACK, DL_OK_ACK, DL_UNITDATA_IND, and DL_UDERR_IND.

The DL_BIND_REQ request uses the DL_bind_req structure defined in **<sys/lihdr.h>** and has the LLC_sap field filled in with the SAP desired. For DL_ETHER-style networks, the SAP value is the Ethernet type value desired. For DL_CSMACD-style networks, the SAP value is the IEEE 802.2 LSAP value desired as the *source* LSAP. Destination LSAPs are specified in the address used with the DL_UNITDATA_REQ primitive. The SubNetwork Access Point (SNAP) is understood and uses the values of the two GROWTH_field locations of DL_bind_req as the "organization" and "organization-specific" fields, respectively. GROWTH_field[0] is treated as a three-octet string containing the organization identifier in network order, and GROWTH_field[1] is the two-octet organization-specific identifier in local system byte ordering. By specifying an LSAP of **−3**, the Novell form of 802.3 usage is enabled. Use of this mode is discouraged since it violates the 802.3 standard usage.

The DL_UNITDATA_REQ primitive requires that a *Remote Address* be specified as part of the control message. The offset of the remote address from the beginning of the control portion of the message is specified in the RA_offset field of the DL_unitdata_req structure. The length of the remote address is specified in the RA_length field. For DL_CSMACD STREAMS, the address is the concatenation of the physical address and the destination LSAP. Data to be sent is in the data portion of the message. The header for the message is generated by the physical interface driver associated with the *llc* driver.

The *llc* driver is linked closely through function calls with one of several link layer drivers, currently all Ethernet drivers. It provides a generic hardware-independent interface to upper protocol layers to avoid duplicate code in each driver.

A number of *ioctl* operations are supported. If device-dependent actions are required, these are communicated to the device driver through a function call. Currently supported operations are:

DLGADDR	Get the physical address of the interface. The argument to the *ioctl* is a six-byte buffer for the physical address.
DLSADDR	Set the physical address of the interface. The argument to the *ioctl* is a six-byte buffer for the physical address.
DLGSTAT	Get the link driver statistics values.
DLGCLRSTAT	Get the link driver statistics and zero the driver values.
DLGBROAD	Get the broadcast address of the interface.
DLPROM	Enable or disable promiscuous mode (accepting all packets).
DLRESET	Reset driver and controller to power up condition.
DLSMULT	Define a multicast address and enable reception of the multicast address in the hardware. The argument is the six-octet value of the multicast address to set.
DLDMULT	Delete a multicast address from the multicast table. The argument is the six-octet multicast address to delete from the table.
DLGMULT	Get the multicast table. With no buffer provided, it returns the number of entries in the multicast table. If a buffer argument is provided, the buffer will be filled with as many of the multicast table entries as will fit in the space provided.

All *llc ioctl* functions are implemented using the I_STR form of the STREAMS *ioctl* as described in *streamio*(7).

SEE ALSO

ec(7), wd(7).
streamio(7) in the *INTERACTIVE UNIX System User's/System Administrator's Reference Manual.*
IEEE Standard 802.2 Logical Link Control.

NAME

lo – Data Link Level Loopback Driver

DESCRIPTION

The *lo* driver provides a generic loopback driver conforming to the DL (datalink) interface specification. Multiple loopback drivers may be configured and used for purposes other than the use with the Internet protocols.

Address format is not relevant and the address value specified in DL_* requests is ignored. DL_UNITDATA_IND messages will have the source address being a duplicate of the destination address provided on the DL_UNITDATA_IND request.

SEE ALSO

ip(7), llc(7).

NAME

pty – pseudo terminal driver

DESCRIPTION

The *pty* driver provides support for a device-pair termed a *pseudo terminal*. A pseudo terminal is a pair of character devices, a *master* device and a *slave* device. The slave device provides an interface identical to that described in *tty*(4). However, whereas all other devices that provide the interface described in *tty*(4) have a hardware device of some sort behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, anything written on the master device is given to the slave device as input and anything written on the slave device is presented as input on the master device.

In configuring, if an optional "count" is given in the specification, that number of pseudo terminal pairs are configured; the default count is 16.

The following *ioctl* calls apply only to pseudo terminals:

TIOCPKT

Enable/disable *packet* mode. Packet mode is enabled by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. When applied to the master side of a pseudo terminal, each subsequent *read* from the terminal will return data written on the slave part of the pseudo terminal preceded by a zero byte (symbolically defined as TIOCPKT_DATA), or a single byte reflecting control status information. In the latter case, the byte is an inclusive-or of zero or more of the bits:

TIOCPKT_FLUSHREAD
whenever the *read* queue for the terminal is flushed.

TIOCPKT_FLUSHWRITE
whenever the *write* queue for the terminal is flushed.

TIOCPKT_STOP
whenever output to the terminal is stopped with ^S.

TIOCPKT_START
whenever output to the terminal is restarted.

TIOCPKT_DOSTOP
whenever *t_stopc* is ^S and *t_startc* is ^Q.

TIOCPKT_NOSTOP
whenever the start and stop characters are not ^S/^Q.

While this mode is in use, the presence of control status information to be read from the master side may be detected by a *select* for exceptional conditions.

This mode is used by *rlogin*(1C) and *rlogind*(1M) to implement a remote-echoed, locally ^S/^Q flow-controlled remote login with proper back-flushing of output; it can be used by other similar programs.

TIOCREMOTE

A mode for the master half of a pseudo terminal, independent of TIOCPKT. This mode causes input to the pseudo terminal to be flow controlled and not input edited (regardless of the terminal mode). Each *write* to the control terminal produces a record boundary for the process reading the terminal. In normal usage, a *write* of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an end-of-file character. TIOCREMOTE can be used when doing remote line editing in a window manager, or whenever flow-controlled input is required.

TIOCPTYCTL

Enable/disable *ptyctl* mode. In *ptyctl* mode, each buffer of data read from or written to the pseudo terminal will start with a flag byte. The flag byte identifies the rest of the buffer. A flag byte of data means the rest of the buffer contains user data. A flag byte other than data means the data buffer contains control information for the pseudo terminal.

Flag bytes for writing include:

pty_ctl3_cmd_user_data
The rest of the buffer contains user data.

pty_ctl3_cmd_char_echo
Control echoing of input characters. The data byte following the command byte must be either 1 (TRUE) or 0 (FALSE). If the data byte is 1, input characters will be echoed if, and only if, the ECHO flag is set on the the slave side of the pseudo terminal. If the data flag is 0, input characters will never be echoed. Character echoing will be true by default when the service is started.

pty_ctl3_cmd_break
A break should be sent to the line discipline.

pty_ctl3_cmd_first_unused
The highest command literal defined.

Flag bytes for reading include:

pty_ctl3_info_user_data
The rest of the buffer contains user data.

pty_ctl3_info_flush_input
All input to the pseudo terminal should be flushed.

pty_ctl3_info_change_ioctl
The slave side program has changed the state of the line discipline's ioctl information.

pty_ctl3_info_want_data
The slave side program has done a read requesting more data.

pty_ctl3_info_output_suspend
Output from the pseudo terminal has been suspended.

pty_ctl3_info_output_resumed
Output from the pseudo terminal has been resumed.

pty_ctl3_info_break
A break has been sent to output.

pty_ctl3_info_first_unused
Specifies the first unused info literal.

FILES

/dev/pty[p–r][0–9a–f]
master pseudo terminals
/dev/tty[p–r][0–9a–f]
slave pseudo terminals

NAME

sl – Serial Line Internet Protocol

DESCRIPTION

The *sl* or Serial Line Internet Protocol (SLIP) line discipline sends IP packets with the normal serial line device drivers. Because the *sl* interface is a point-to-point link, it must have network addresses assigned for both the source and destination interfaces used. These addresses may be set or changed with the *ioctl* calls SIOCSIFADDR and SIOCSIFDSTADDR.

The most common procedure is to enable the *sl* network interface and then connect it to another system running SLIP using the *slattach*(1M) or *sllogin*(1M) commands.

SEE ALSO

slattach(1M), sllogin(1M), inet(7).

NAME

socket – Berkeley socket emulation under System V STREAMS

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
s = socket (AF_INET, type, protocol);
```

or

```
#include <sys/sohdr.h>
s = open (device, O_RDWR, 0);
ioctl (s, I_PUSH, "socket");
```

DESCRIPTION

Berkeley sockets are emulated by the *socket* STREAMS module and the user library **libinet.a**. The module and library communicate, using a *socket protocol* that is defined in **<sys/sohdr.h>**. All state information about the open *socket* is stored in the *socket* module, so extra system calls can be avoided. The state is unaffected by forks.

The Berkeley system calls emulated by the library are:

SOCKET
: Open the appropriate STREAMS device and push the *socket* module on the stream.

BIND
: Bind a name to the *socket*.

CONNECT
: Initiate a connection to a foreign address.

LISTEN
: Listen for connections on the *socket*.

ACCEPT
: Accept a connection on a listening *socket*.

SEND, SENDTO, SENDMSG
: Send a message to a foreign address.

RECV, RECVFROM, RECVMSG
: Receive a message from a foreign address.

SETSOCKOPT, GETSOCKOPT
: Set or retrieve *socket* option values.

GETSOCKNAME
: Get the name bound to the *socket*.

GETPEERNAME
: Get the peer name connected to the *socket*.

SHUTDOWN
: Shut down part of the *socket* connection.

SELECT
: Emulate the synchronous I/O multiplexing using the *poll* system call.

The *read* and *write* system calls are converted by the *socket* module into the Transport Layer Interface (TLI) required by lower-level protocols.

The direct interface to the *socket* module consists of the user-originated primitives SO_SOCKET, SO_BIND, SO_CONNECT, SO_LISTEN, SO_ACCEPT, SO_FDINSERT, SO_SEND, SO_SETOPT, SO_GETOPT, SO_GETNAME, SO_GETPEER and SO_SHUTDOWN and the module-originated primitives SO_ACCEPT_ACK, SO_UNITDATA_IND, SO_GETOPT_ACK, SO_GETNAME_ACK, SO_GETPEER_ACK, SO_ERROR_ACK, and SO_OK_ACK.

The current implementation of the *socket* module is limited to the AF_INET domain and the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW types. EPROTO will be returned, corresponding to the TLI error TOUTSTATE.

SEE ALSO

accept(3I), bind(3I), connect(3I), getpeername(3I), getsockname(3I), getsockopt(3I), listen(3I), recv(3I), select(3I), send(3I), shutdown(3I), socket(3I), ip(7), tcp(7), udp(7).

An Introductory 4.3BSD Interprocess Communication Tutorial, Stuart Sechrest, University of California, Berkeley.

An Advanced 4.3BSD Interprocess Communication Tutorial, Samuel J. Leffler, et.al., University of California, Berkeley.

"Transport Layer Interface," *Network Programmer's Guide*, UNIX System V Release 3.

DIAGNOSTICS

Most of the failures reported by the *socket* module will be specific to one of the Berkeley system calls emulated, so the appropriate manual page for the library function should be consulted. If library functions are called in the wrong order or if error indications are ignored and calls proceed after an error, the error EPROTO will be returned, corresponding to the TLI error TOUTSTATE.

NAME

tcp – Internet Transmission Control Protocol

SYNOPSIS

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

s = socket(AF_INET, SOCK_STREAM, 0);
or
#include <sys/tiuser.h>
#include <sys/tcp.h>
#include <netinet/in.h>

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

DESCRIPTION

The *tcp* protocol provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the SOCK_STREAM abstraction in the *socket* environment and the T_COT type of transport in the TLI environment. *tcp* uses the standard Internet address format and provides a per-host collection of port addresses. Thus, each address is composed of an Internet address specifying the host and network, with a specific *tcp* port on the host identifying the peer entity.

Transport endpoints utilizing the *tcp* protocol are either active or passive. Active endpoints initiate connections to passive endpoints. By default, *tcp* endpoints are created active; to create a passive socket in the socket emulation, the *listen*(3I) call must be used after binding the socket with the *bind*(3I) call; to create a passive endpoint with TLI, the *qlen* field in the *t_bind* structure of a *t_bind*(3N) call must be set to a nonzero value. Only passive endpoints may use the *accept*(3I) call of the socket emulation or the *t_accept*(3N) call of TLI to accept incoming connections. Only active sockets may use the *connect*(3I) call of the socket emulation to initiate connections. In the TLI interface, both passive and active STREAMS may initiate a connection with the *t_connect*(3N) call.

Passive endpoints may underspecify their location to match incoming connection requests from multiple networks. This technique, termed wildcard addressing, allows a single server to provide service to clients on multiple networks. To create an endpoint which listens on all networks, the Internet address INADDR_ANY must be bound. The TCP port may still be specified at this time; if the port is not specified the system will assign one. Once a connection has been established, the endpoint's address is fixed by the peer entity's location. The address assigned the endpoint is the address associated with the network interface through which packets are being transmitted and received. Normally this address corresponds to the peer entity's network.

tcp supports several options which are set with *setsockopt* (see *getsockopt*(3I)) in the socket emulation and *t_optmgmt*(3N) or *t_connect*(3N) in the option field for TLI, and tested with *getsockopt*(3I) for the socket emulation and *t_optmgmt*(3N) for TLI.

Under most circumstances, *tcp* sends data when it is presented. When outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an

acknowledgment is received. The option level for the *setsockopt* call is the protocol number for *tcp*, available from *getprotobyname* (see *getprotoent*(3I)). The same values are used with the TLI option management, but the options are specified as a sequence of options in the format specified in *struct inetopts* in:

#include <netinet/in.h>

The TLI options also support a number of the *socket* level options.

The socket-level options that apply to *tcp* are: SO_SNDBUF, SO_RCVBUF, SO_SNDLOWAT, SO_RCVLOWAT, SO_DEBUG, SO_REUSEADDR, SO_KEEPALIVE, SO_BROADCAST, and SO_OOBINLINE, which are described in *getsockopt*(3I).

For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays. Therefore, *tcp* provides a boolean option, TCP_NODELAY (from **<netinet/tcp.h>**), to defeat this algorithm.

tcp also has the option TCP_OOBMODE, which provides a mechanism for changing the interpretation of *tcp urgent* data. The parameter to TCP_OOBMODE is an integer bit mask. The bits determine the exact mode. Options are defined in **netinet/tcp_var.h**.

TCPOOB_RFCFORM urgent pointer is last octet of urgent data rather than first octet of non-urgent data.

TCPOOB_SEGMENT return all urgent data rather than last octet. The urgent segment is all data from the beginning of the *tcp* segment to the end of urgent data.

Options at the *ip* transport level may be used with *tcp*; see *ip*(7). Incoming connection requests that are source-routed are noted, and the reverse source route is used in responding.

DIAGNOSTICS

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

[EISCONN] when trying to establish a connection on a socket which already has one.

[ENOBUFS] when the system runs out of memory for an internal data structure.

[ETIMEDOUT] when a connection was dropped due to excessive retransmissions.

[ECONNRESET] when the remote peer forces the connection to be closed.

[ECONNREFUSED] when the remote peer actively refuses to establish a connection (usually because no process is listening to the port).

[EADDRINUSE] when an attempt is made to create a socket with a port that has already been allocated.

[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for which no network interface exists.

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

[TOUTSTATE]	when a function was used in the wrong sequence on the STREAM descriptor or the STREAM was not in the idle state.
[TACCES]	when the use did not have the proper permissions for the operation.
[TBADOPT]	when an incorrectly formatted option or an unimplemented option is specified.
[TBADDATA]	when the amount of data was not within the bounds of the transport provider.
[TBADSEQ]	when an invalid connection sequence number was specified.
[TLOOK]	when an asynchronous event occurred on the STREAM descriptor.
[TNOTSUPPORT]	when the underlying transport provider does not support a requested operation.
[TSYSERR]	when a system error occurs during operation.
[TNOADDR]	when the transport provider cannot assign the requested address.
[TBADADDR]	when the user attempts to use an illegal address or formats the address information incorrectly.
[TNODATA]	when O_NDELAY is set and an operation would normally have blocked.
[TBADFLAG]	when an invalid flag was provided in a *t_optmgmt* request.

SEE ALSO

getprotoent(3I), getsockopt(3I), socket(3I), intro(7N), inet(7), ip(7), socket(7).

Network Programmer's Guide.

NAME

ub – Ungermann-Bass PC NIC (Ethernet controller)

DESCRIPTION

The *ub* interface provides STREAMS access to the Ungermann-Bass PC NIC 10 MB Ethernet controller. The Data General PC Lan Controller also uses this driver.

The driver is provided with *IEEE 802.2 LLC Class 1* and *Ethernet* encapsulation of data by the *llc*(7) driver. All communication with the *ub* driver is via function calls between *ub* and *llc*.

The *ub* driver supports promiscuous mode, as configured by *llc*. It receives and understands "trailer" packets, but does not generate them. Loopback is handled between the *ub* and *llc* drivers.

Configuration

The **space.c** file provides a mechanism for specifying the PC NIC board operating parameters to the driver. If a different interrupt vector or shared memory base address is necessary, the **space.c** file should be updated accordingly by means of the *sysadm* command.

SEE ALSO

arp(7), ip(7), llc(7).

NAME

udp – Internet User Datagram Protocol

SYNOPSIS

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

s = socket(AF_INET, SOCK_DGRAM, 0);
```

or

```
#include <sys/tiuser.h>
#include <sys/udp.h>
#include <netinet/in.h>

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

DESCRIPTION

udp is a simple, unreliable datagram protocol that is used to support the SOCK_DGRAM abstraction in the *socket* environment and the T_CLTS type of transport in the TLI environment for the Internet protocol family. Transport endpoints utilizing the *udp* protocol are connectionless. *udp* sockets are normally used with the *sendto* and *recvfrom* calls, though the *connect*(3I) call may also be used to fix the destination for future packets (in which case the *recv*(3I) or *read*(2) and *send*(3I) or *write*(2) system calls may be used). *udp* TLI STREAMS are used with the *t_sndudata*(3N) and *t_rcvudata*(3N) calls.

udp address formats are identical to those used by TCP. In particular, *udp* provides a port identifier in addition to the normal Internet address format. Note that the *udp* port space is separate from the TCP port space, i.e., a *udp* port may not be connected to a TCP port. In addition, broadcast packets may be sent (assuming the underlying network supports this) by using a reserved broadcast address; this address is network interface dependent.

udp supports several options that are set with *setsockopt*(3I) in the socket emulation and *t_optmgmt*(3N) for TLI and tested with *getsockopt*(3I) for the socket emulation and *t_optmgmt*(3N) for TLI.

The option level for the *setsockopt* call is the protocol number for *udp*, available from *getprotobyname (see getprotoent*(3I)). The same values are used with the TLI option management, but the options are specified as a sequence of options in the format specified in *struct inetopts* in:

```
#include <netinet/in.h>
```

The TLI options also support a number of the *socket* level options.

The socket level options that apply to *udp* are SO_BROADCAST and SO_DONTROUTE, which are described in *getsockopt*(3I).

Options at the IP transport level may be used with *udp*; see *ip*(7).

SEE ALSO

getprotoent(3I), getsockopt(3I), socket(3I), intro(7N), inet(7), ip(7), socket(7).

Network Programmer's Guide.

DIAGNOSTICS

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

[EISCONN] when trying to establish a connection on a socket that already has one, or when trying to send a datagram with the destination address specified and the socket is already connected.

[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket has not been connected.

[ENOBUFS] when the system runs out of memory for an internal data structure.

[EADDRINUSE] when an attempt is made to create a socket with a port that has already been allocated.

[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for which no network interface exists.

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

[TOUTSTATE] when an operation is requested in the wrong sequence.

[TBADOPT] when an incorrectly formatted or unimplemented option is specified.

[TBADF] when the specified file descriptor does not refer to a transport endpoint.

[TACCESS] when the user does not have the proper permissions for the operation requested.

[TBADDATA] when the amount of data is not within the bounds of the transport provider.

[TNOTSUPPORT] when the underlying transport provider does not support the requested operation.

[TSYSERR] when a system error occurs during an operation.

[TNOADDR] when the transport provider cannot assign the requested address.

[TBADADDR] when the user attempts to use an illegal or incorrectly formatted address.

[TNODATA] when O_NDELAY is set, but no data units are currently available from the transport provider, and the operation would normally have blocked.

[TBADFLAG] when an invalid flag is specified in a *t_optmgmt* request.

BUGS

read(2) may not be used with SOCK_DGRAM sockets due to the way STREAMS works.

NAME

wd – Western Digital 8003 PC adapter board

DESCRIPTION

The *wd* driver provides STREAMS access to the Western Digital 8003 family of Local Area Network adapters. The board provides a pool of buffers for buffers received from the network and provides for all the management of the on-board buffers.

The driver provides for *IEEE 802.2 LLC Class 1* and *Ethernet* encapsulation of data. All communication with the driver is done via STREAMS messages in the *Link Interface* format specified in the file **/usr/include/sys/lihdr.h**.

The LLC1 protocol supports receipt of and response to XID and TEST but currently does not generate them. This may change with a later release of the driver. Group Service Access Points (SAPs) are not currently supported, although the broadcast SAP is.

A user may bind to any non-group SAP address that is not already bound to another STREAM. Attempts to bind to a non-legal SAP address are rejected.

If the SAP value to be bound is a value larger than 255 (0xFF), then the type of encapsulation is taken to be the older Ethernet style rather than IEEE 802.2. Rather than a SAP value, the value passed will be used as the *Ethernet type* field for the STREAM. If the SAP value is **−3**, then the Novell form of 802.3 encapsulation is used. This mode is provided to support Ported NetWare, but its use is discouraged since it violates the 802.3 standard usage.

By supporting both link level types, the driver is capable of supporting a wide variety of protocols.

For DL_ETHER-style networks, the SAP value is the Ethernet type value desired. For DL_CSMACD-style networks, the SAP value is the IEEE 802.2 LSAP value desired as the *source* LSAP. Destination LSAPs are specified in the address used with the DL_UNITDATA_REQ primitive. The SubNetwork Access Point (SNAP) is understood and uses the values of the two GROWTH_field locations of DL_bind_req as the "organization" and "organization-specific" fields, respectively. GROWTH_field[0] is treated as a three-octet string containing the organization identifier in network order, and GROWTH_field[1] is the two-octet organization-specific identifier in local system byte ordering. For 802.3 hardware, a third type of use is possible. By specifying an LSAP of **−3**, the Novell form of 802.3 usage is enabled. Use of this mode is discouraged since it violates the 802.3 standard usage.

A number of *ioctl* operations are supported. Some of the operations may be specific to the Western Digital ViaNet product.

DLGADDR	Get the MAC address of the interface.
DLSADDR	Set a new MAC address to be used by the interface.
DLGBROAD	Get the MAC broadcast address.
DLGSTAT	Get the current board statistics counters.

DLPROM	Enable promiscuous mode.
DLSMULT	Define a multicast address and enable reception of the multicast address in the hardware. The argument is the six-octet value of the multicast address to set.
DLDMULT	Delete a multicast address from the multicast table. The argument is the six-octet multicast address to delete from the table.
DLGMULT	Get the multicast table. With no buffer provided, it returns the number of entries in the multicast table. If a buffer argument is provided, the buffer will be filled with as many of the multicast table entries as will fit in the space provided.

Configuration

The **space.c** file provides a mechanism for specifying the WD8003 operating parameters to the driver. If a different shared memory address or I/O port address are necessary, the **space.c** file should be updated accordingly.

Western Digital WD8013 Adapter Board

When using a WD8013 board in a 16-bit slot, the driver will program the board into 16-bit access mode. The ISA bus specification requires that the bus be put into 16-bit mode for this type of operation. This causes the 128K region in which the WD8013's shared memory region resides to be usable only by 16-bit boards during this time. In order to coexist with 8-bit type boards (such as VGA) which may overlap this region, the *wd* driver only enters 16-bit mode during data transfer and leaves this mode when data transfer is complete. Interrupts are disabled during this time to prevent an 8-bit device from interrupting the data transfer while 16-bit mode is enabled. If it is known that the WD8013 has no conflicts with 8-bit boards in the system, it is possible to configure the driver to not disable interrupts during the data copy. This may be accomplished by setting the *wd_bflags* field of the *wdparams* structure in the **wd0/space.c** file to WDF_NODISABLE. The *wd_bflags* field is the structure member following the number of units field.

The WD8013 board may have its shared memory region at a location above 1 MB. This can eliminate the possible conflict with an 8-bit board.

WARNINGS

Future releases of this software will conform to USL's link level specification. Consequently, some of the interfaces may change.

SEE ALSO

ip(7), llc(7).

ANSI/IEEE Standard 802.2 – Logical Link Control.

SNMP Overview and Installation Instructions

CONTENTS

SNMP Overview and Installation Instructions

1. INTRODUCTION

Welcome to SNMP. This article provides a brief overview of INTERACTIVE's implementation of the Simple Network Management Protocol (SNMP) facilities.

The "SNMP Reference Manual," which contains manual entries that describe the programs and commands available with SNMP, is also supplied.

The SNMP protocol is used to inspect or alter management information remotely. It provides a simple, workable architecture and system for managing TCP/IP-based internets, in particular the Internet.

SNMP includes user programs and a development environment. The user applications consist of a daemon (the SNMP agent) and certain utilities used in network management. These utilities act as SNMP clients either of the local agent or of the remote SNMP agents. The development environment includes the header files and SNMP libraries required by an SNMP applications developer.

2. OVERVIEW

2.1 The SNMP Architecture

The SNMP architectural model is a collection of network management stations and network elements. The network management stations execute management applications to monitor and control network elements, which are devices such as hosts, gateways, terminal servers, etc., that have management agents responsible for performing the requested functions. SNMP is used to communicate management information between the stations and agents in the network elements.

2.2 Services Provided

SNMP provides a set of facilities used in queries for statistics. These statistics are called the Management Information Base (MIB). The information retrieval services currently provided are queries to the following MIB objects:

- System
- Interfaces
- Address Translation (AT)
- IP
- ICMP
- TCP
- UDP

This implementation of SNMP includes several application programs (commands) which are described in section 2.4, "Utilities," and in the SNMP manual entries. There are commands to retrieve information (MIB variables), commands to alter (set) information, commands to display the contents of network-related data structures, and a daemon (the SNMP agent) that handles the SNMP information. Commands to set statistics other than the interface state are not supported in this release.

The commands to retrieve MIB variables can query either the local SNMP agent or a remote SNMP agent.

2.3 The SNMP Agent

The SNMP agent is represented by a daemon program called `snmpd`. This program listens and responds to network management queries and commands from local or remote network management stations. The `snmpd` program uses three configuration files, which are described in the manual entry *snmpd*(1M).

2.4 Utilities

Several SNMP utility programs are provided. They will be installed in the directory `/usr/ucb`. The `getid`, `getmany`, `getnext`, `getone`, and `getroute` commands are used to retrieve MIBs from a local or remote SNMP agent.

`setany` is used to retrieve and set MIBs. The current SNMP agent implementation does not support other any other set commands.

`snmpstat` provides a general network status of such elements as the transport endpoint table, routing table, and address translation table.

The utilities `trap_send` and `trap_rece` are used to send and receive SNMP trap PDUs.

2.5 Headers and Libraries

The header files and library routines are provided for SNMP application developers. They perform the necessary details of constructing the full SNMP packets for the users. Those tasks include ASN.1 compilation, etc.

3. INSTALLING SNMP

3.1 Software Requirements

You must have INTERACTIVE TCP/IP Version 1.3 or later installed on your system before you install SNMP.

3.2 Installation Using the System Administration Menu

The System Administration menu (`sysadm`) can be used to perform the installation. You must use the console terminal to log in as `sysadm`, and you will be asked for the password assigned to `sysadm`. Experienced UNIX* System users can log in to the console terminal as `root` and perform these same tasks from the command line.

To install the SNMP software on an INTERACTIVE UNIX System:

1. Use the `sysadm` command or log in as `sysadm` and access the Main menu. Your screen will look similar to this:

```
                    SYSTEM ADMINISTRATION

  1 diskmgmt        disk management menu
  2 filemgmt        file management menu
  3 machinemgmt     machine management menu
  4 packagemgmt     package management menu
  5 softwaremgmt    software management menu
  6 syssetup        system setup menu
  7 ttymgmt         tty management menu
  8 usermgmt        user management menu

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  q to QUIT:
```

2. Type `q` at any time to quit the current operation. If a `?` appears as a choice, you can type `?` for help.

   Select option `5` to access the Software Management (`softwaremgmt`) menu. Your screen will look similar to this:

```
                    SOFTWARE MANAGEMENT

  1 installpkg      install new software package onto built-in disk
  2 listpkg         list packages already installed
  3 removepkg       remove previously installed package from built-in disk
  4 runpkg          run software package without installing it

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT: 1
```

3. Select option 1, `installpkg`.

   The system will then prompt you for the number of the drive you will use for the installation. Your screen will look similar to this:

   ```
   Enter the drive number you wish to install from ((default)0,1):
   ```

4. Type the number that corresponds to the drive you are using for the installation, or press ENTER to accept the default.

   The system will then prompt you for the density of the diskettes you will use for the installation. Your screen will look similar to this:

   ```
   Enter density of the diskettes you are installing from:
           1) 1.2 MB   ( 5 1/4" high density )
           2) 1.44 MB  ( 3 1/2" high density )
           3) 360 Kb   ( 5 1/4" low density )
           4) 720 Kb   ( 3 1/2" low density )

   Please enter #(1-4), default 1:
   ```

5. Type the number that corresponds to the type of diskettes you are using for the installation, or press ENTER to accept the default. The system will then ask you to insert the first diskette of the subsystem into the diskette drive. Your screen will look similar to this:

   ```
   Confirm

   Please insert the floppy disk.

   If the program installation requires more than one floppy
   disk, be sure to insert the disks in the proper order,
   starting with disk number 1.
   After the first floppy disk, instructions will be provided
   for inserting the remaining floppy disks.

   Strike ENTER when ready
   or ESC to stop.
   ```

6. If you wish to stop the installation process and return to the system prompt, press [ESC]. To proceed with the installation, insert the *SNMP* diskette into the drive and press [ENTER]. The following message will be displayed:

```
Installation is in progress -- do not remove the floppy disk.
Install Simple Network Management Protocol -
Version 1.1 (y/n)?
```

7. Type **y** to proceed with the installation of SNMP. (You can type **n** to stop the installation.) If you type **y**, your screen will look similar to this:

```
Copyright (c) 1988, 1990 Interactive Systems Corporation
All Rights Reserved

Derived from SNMP Research sources version 9.4.0.3
Copyright 1987, 1988, 1989 Jeffrey D. Case and Kenneth W. Key
(SNMP Research)

Installing Simple Network Management Protocol -
Version 1.1
The following files are being installed:
.
.
.
Creating Necessary System Files.
If these files currently exist, they will NOT be over-written.

The file /etc/snmp/snmpd.config has been created.
You should edit this file to your site specifications.

The file /etc/snmp/snmpd.commun has been created.
You should edit this file to your site specifications.

The file /etc/snmp/snmpd.trap_c has been created.
You should edit this file to your site specifications.
```

8. A list of the files being copied onto your system from the diskette will accompany the above message. When all the files have been copied, the following message will be displayed:

```
Installation of Simple Network Management
Protocol - Version 1.1 is complete.

Confirm

Please insert the floppy disk.

If the program installation requires more than one floppy
disk, be sure to insert the disks in the proper order,
starting with disk number 1.
After the first floppy disk, instructions will be provided
for inserting the remaining floppy disks.

Strike ENTER when ready
or ESC to stop.

Press the RETURN key to see the softwaremgmt menu [?, ^, q]:
```

4. FOR MORE INFORMATION

This article and the accompanying manual entries provide information on how to use the SNMP protocol, and supplement the networking information found in the *INTERACTIVE TCP/IP Guide*. For a complete listing of all INTERACTIVE UNIX System-related documentation, refer to the "Documentation Roadmap" included in the *INTERACTIVE UNIX Operating System Guide*.

For more information on the standard network management services, refer to RFC1157. For more information on the statistics that make up the MIB, refer to RFC1156. These RFCs also provide references to other RFCs that address network-related issues.

SNMP Reference Manual

CONTENTS

NAME

getid – retrieve the "system" Management Information Base (MIB) variables from an SNMP entity

SYNOPSIS

getid entity_addr "community_string"

DESCRIPTION

getid is a Simple Network Management Protocol (SNMP) application that retrieves the variables *sysDescr.0*, *sysObjectID.0*, and *sysUpTime.0* from an SNMP entity. The arguments are the entity's address and the community string needed for access to the SNMP entity. The primary purpose of this application is to illustrate the use of the SNMP library routines.

SEE ALSO

getmany(1).
RFC1065, RFC1066, RFC1098.

NAME

getmany – retrieve classes of variables from an SNMP entity

SYNOPSIS

getmany entity_addr "community_string" ["variable class name"]...

DESCRIPTION

getmany is an SNMP application that retrieves classes of variables from an SNMP entity. The arguments are the entity's address, the community string for access to the SNMP entity, and the variable class name(s) expressed as an object identifier in either dot notation or the mib-variable name as it appears in the MIB document. *getmany* retrieves the variable class by calling the SNMP entity with the variable class name to get the first variable in the class. It then calls the entity again using the variable name returned in the previous call to retrieve the next variable in the class, using the "get-next" aspect of the variable retrieval system. For example, running the following:

```
getmany suzzy public ipRouteDest
```

will traverse the network entity's *ipRouteDest* variable class (the next node travelled to in the route for the given net number, which makes up the rest of the variable name). The traversing of the variable space stops when all of the classes being polled return a variable of a class different than that which was requested. Note that a network entity's entire variable tree can be traversed with the call:

```
getmany suzzy public iso
```

SEE ALSO

RFC1065, RFC1066, RFC1098.

NAME

getnext – retrieve variables from an SNMP entity

SYNOPSIS

getnext entity_addr "community_string" ["variable name"]...

DESCRIPTION

getnext is an SNMP application that retrieves a set of individual variables from an SNMP entity using a "get-next" request. The arguments are the entity's address, the community string for access to the SNMP entity, and the variable name(s) as it appears in the MIB document or as expressed in dot notation. It should be noted that since the function uses the powerful *getnext* operator, the variable returned will be the lexicographically greater fully qualified object identifier for what was entered. For example:

getnext suzzy public system interfaces

would return the variables *sysDescr.0* and *ifNumber.0*.

SEE ALSO

getmany(1), getone(1).
RFC1065, RFC1066, RFC1098.

NAME

getone – retrieve variables from an SNMP entity

SYNOPSIS

getone entity_addr "community_string" ["variable name"]...

DESCRIPTION

getone is an SNMP application that retrieves a set of individual variables from an SNMP entity using a "get" request. The arguments are the entity's address, the community string for access to the SNMP entity, and the fully qualified variable name(s) as it appears in the MIB document or as expressed in dot notation. It should be noted that since the function is "get," as opposed to a "get-next," the variable *must* be fully qualified for the request to be successful. For example:

getone suzzy public sysDescr.0 ifNumber.0

would return the variables *sysDescr.0* and *ifNumber.0*, but the call:

getone suzzy public system

would return an error from the entity since it is not a fully qualified SNMP variable.

SEE ALSO

getmany(1), getnext(1).
RFC1065, RFC1066, RFC1098.

NAME

getroute – extract the routing information from an SNMP entity

SYNOPSIS

getroute entity_addr "community_string"

DESCRIPTION

getroute is an SNMP application that retrieves routing information from an entity by traversing the *ipRouteDest*, *ipRouteIfIndex*, *ipRouteMetric1*, *ipRouteNextHop*, *ipRouteType*, and *ipRouteProto* variable classes for each route found. It takes as arguments the address of the SNMP entity and a community string to provide access to that entity.

SEE ALSO

getmany(1).
RFC1065, RFC1066, RFC1098.

NAME

setany − retrieve and set variables in an SNMP entity

SYNOPSIS

setany entity_addr "community_name"
["variable name" < i/o/d/a/c/g/t > value]...

DESCRIPTION

setany does a "get" request to obtain the current values of the variables to be set, then performs a set request on the variables. The arguments are the entity name or address in Internet dot notation, the community name for access to the SNMP entity, and a triplet for each variable to be set consisting of: the variable name in dot notation, an *i*, *o*, *d*, *a*, *c*, *g*, or *t* to indicate if the variable's value is being given as an integer, an octet string (in hex notation), an object identifier (in dot notation), an IP address (in dot notation), a counter, a gauge, or time-ticks, followed by the value. For example:

setany 128.169.1.1 suranet0 "ifAdminStatus.2" −i 3

sets the administrative status of interface 2 to 3 (down).

During a set request, a get request is issued to obtain the variable name that will be used for the request.

SEE ALSO

RFC1065, RFC1066, RFC1098.

NAME

snmpd – an SNMP agent daemon

SYNOPSIS

snmpd [–v] [–C configuration_file **]**

DESCRIPTION

snmpd is an SNMP application that listens for and responds to network management queries and commands from logically remote network management stations. *snmpd* accesses the supported variables in the instrumentation of the protocol layers and the kernel. These parameters are made accessible via the SNMP. *snmpd* starts the daemon that handles SNMP requests.

The following options are available:

–v Enables verbose mode and is output to standard output. Debugging information is written to standard output.

–C configuration_file Allows override of default values; **snmpd.config** is the default name for this file.

Configuration Files

The *snmpd* program uses three configuration files. By default these files are read from the directory where *snmpd* is invoked. An option to *snmpd* is used to specify a different path name for the first configuration file, while the other two can be assigned dynamically in the first configuration file to any location in the file system.

snmpd.config

This file is used to override default values. It contains from zero to five lines in any combination or order. Two lines can be used to override the default values for the other configuration files. The other three can be used to override the default values for the three "system" variables. For example:

```
COMMUNITY=community_configuration_filename
TRAPCOMM=trap_community_configuration_filename
DESCR=system_description_string
OBJID=system_object_identifier_string
FORWARD=forward_value
```

A *forward_value* of 1 indicates that the machine running the SNMP agent is a gateway, which forwards IP datagrams.

A *forward_value* of 2 indicates that the machine running the SNMP agent is a host, which does not forward IP datagrams.

The keywords *descr*, *objid*, and *forward* are case-insensitive and will match on any case.

snmpd.commun

This file specifies the manager stations authorized to issue queries and commands to the agent. It also allows the system administrator to tailor the SNMP agent so that it only recognizes particular community names, Internet addresses, and enforce privileges for those recognized. A combination of community name, Internet address, and privilege constitutes a record, which is represented as a line in the file.

Each line consists of three fields. The first field is the community name. The second field is the IP address of the remote site; an Internet address of 0.0.0.0 means that any machine referring to the community name with the proper privilege will pass authentication to issue queries and commands to the SNMP agent. The third field specifies the privileges given that community name. These currently consist of *read* for read only, *write* for read/write, or *none* to lock out a community name. The format is:

community_name IP_address_in_dot_notation privileges

For example, this file could contain

public 0.0.0.0 read
private 128.212.48.29 read
private 128.212.48.30 read

The first line means that the SNMP agent recognized the community name *public*, which is readable by any machine. The second line means that the SNMP agent recognized the community name *private*, which is only readable by a machine with an Internet address of 128.212.48.29. The third line also means that the SNMP agent recognized the community name *private*, and it is only readable by a machine with an Internet address of 128.212.48.30.

A "**#**" in column 1 indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that read the file.

snmpd.trap_c
This file is used to register remote SNMP trap services; these services are registered in a combination of community name, remote Internet address, and remote UDP port number used. The SNMP agent will send trap PDUs to those machines listed in the file through their UDP ports at the specified time, such as at startup time. Each line consists of three fields. The first is the community name. The second is the IP address of the remote site. The third is the port number on the manager station. The format is:

community_name IP_address_in_dot_notation port_number

For example, this file could contain:

test2 128.212.48.29 162
test3 128.212.48.30 162

This means that at the proper time, the SNMP agent will send a trap PDU for community name test2 to a machine with Internet address 128.212.48.29 using UDP port 162 and to a machine with address 128.212.48.30, also using UDP port 162. Other UDP ports can be used if the remote machines choose to do so, but port number 162 is normally used for this purpose.

Limitations

Not all MIB variables as specified in RFC1066 can be supported without source code alterations to the kernel or device drivers.

FILES

/etc/snmp/snmpd
/etc/snmp/snmpd.config
/etc/snmp/snmpd.commun
/etc/snmp/snmpd.trap_c

SEE ALSO

RFC1065, RFC1066, RFC1098.

WARNING

snmpd(1M) must be used with INTERACTIVE TCP/IP Version 1.3 or later.

NAME

snmpstat – show network status using SNMP

SYNOPSIS

snmpstat [–trasin] [host **] [** session **]**

DESCRIPTION

The *snmpstat* command symbolically displays the contents of various network-related data structures. The options have the following meanings:

–t show the complete transport endpoint table

–r show the routing table

–a show the address translation table

–s show the variables comprising the *system* group of the MIB

–i show the status of active interfaces

–n display addresses and port numbers numerically instead of symbolically

The arguments *host* and *session* allow substitutes for the defaults **localhost** and **public**. *session* refers to the SNMP session or community in which to make the specified requests.

There are a number of display formats, depending on the information presented.

By default, active transport endpoints are displayed. The **–t** flag is used to display all transport endpoints, including servers. Active transport endpoints are those in which the local address portions have been set to a specific address. The protocol, local and remote address, and the internal state of the protocol (if applicable) are shown.

Address formats are of the form *host.port* or *network.port* if an endpoint's address specifies a network but no specific host address. If the **–n** flag is not used, the host and network addresses and port numbers are displayed symbolically according to the databases **/etc/hosts**, **/etc/networks**, and **/etc/services**, respectively. If a symbolic name for an address is unknown or if the **–n** flag has been used, the address is printed in Internet dot notation. Unspecified, or "wildcard," addresses and ports appear as "*".

The routing table display indicates the available routes and the status of each. Each route consists of a destination host or network and a gateway to use in forwarding packets. The *metric* field shows the metric associated with the route. The *type* field displays what kind of route this is, whether it is for a directly connected network or a remote network, etc. The *proto* field indicates the mechanism by which the route was learned. The *interface* shows the name of the interface with which this route is associated. The *type* and *proto* fields are displayed symbolically.

The address translation display indicates the current knowledge regarding address translations for remote hosts that have been communicated with recently. Entries in the address translation table consist of a host address, its physical address (typically an Ethernet

address), and the name of the interface for which this translation is valid.

The system display contains the description of the entity being managed, the object identifier describing the management subsystem of the entity, and the duration of time since the management subsystem was re-initialized.

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and queue lengths. The name, address, and "mtu" (maximum transmission unit) of the interface are also displayed.

SEE ALSO

getid(1), getroute(1), snmpd(1M), hosts(5), networks(5).
RFC1065, RFC1066, RFC1067.

NAME

trap_rece – receive traps from a remote trap generating entity

SYNOPSIS

trap_rece

DESCRIPTION

trap_rece is a program that receives traps from remote SNMP trap generating entities. It binds to the SNMP trap port (**udp/162**) to listen for the traps and thus must be run as **root**. It prints to standard output messages corresponding to the traps it has received. The primary purpose of this program is to demonstrate how traps are parsed using the SNMP library.

SEE ALSO

trap_send(1).
RFC1065, RFC1066, RFC1098.

NAME

trap_send – generate traps

SYNOPSIS

trap_send ip_address trap_type

DESCRIPTION

trap_send is a program that sends trap messages to trap monitoring stations. It takes as arguments the IP address of the monitoring station and the integer number that corresponds to the trap to be sent.

SEE ALSO

trap_rece(1).
RFC1065, RFC1066, RFC1098.

NAME

build_authentication – perform the ASN.1 compilation to build the full SNMP packet

SYNOPSIS

```
#include "snmp.h"
build_authentication(auth_ptr, pdu_ptr);
AuthHeader *auth_ptr;
PDU *pdu_ptr;
```

DESCRIPTION

This routine takes the PDU and the authentication information and builds the actual SNMP packet in the authentication data structure's ***packlet** octet string area. **auth_ptr->packlet->octet_ptr** points to the packet and **auth_ptr->packlet->length** points to the packet's length. Since this "packlet" is *free*d when *free_authentication*(3I) is called, it should be *bcopy()*ed to a holding area, or the authentication should not be *free*d until the packet is actually sent. Once this has been done, the authentication can be *free*d with a call to *free_authentication()*.

*AuthHeader *auth_ptr* is the pointer to the authentication header returned by *make_authentication*(3I). *PDU *pdu_ptr* is the pointer to the PDU structure returned by *make_pdu*(3I).

SEE ALSO

free_authentication(3I), make_authentication(3I), make_pdu(3I).

NAME

build_pdu – perform the ASN.1 compilation to make the PDU

SYNOPSIS

```
#include "snmp.h"
build_pdu(pdu_ptr);
PDU *pdu_ptr;
```

DESCRIPTION

This routine is called with the PDU pointer being used to create the PDU. It traces down the structure of *var_bind*s that has been added to it and builds the ASN.1 packet in the ***packlet** pointer of the PDU pointer's data structure. At this point, PDU processing is complete, and the structure is ready to be passed on to the authentication layers.

*PDU *pdu_ptr* is the pointer to the PDU structure returned by *make_pdu*(3I).

SEE ALSO

build_authentication(3I), link_varbind(3I), make_pdu(3I).

NAME

free_authentication – free a "trivial" authentication header

SYNOPSIS

```
#include "snmp.h"
free_authentication(auth_ptr);
AuthHeader *auth_ptr;
```

DESCRIPTION

This routine frees all memory associated with a "trivial" authentication header data structure, including the actual SNMP packet that *build_authentication*(3I) creates and the octet string associated with *make_authentication*(3I). The PDU structure pointed to by the *pdu_ptr* that is passed to *make-authentication()* is *not* touched.

SEE ALSO

build_authentication(3I), free_pdu(3I), make_authentication(3I).

NAME

free_pdu – free the PDU data structure

SYNOPSIS

```
#include "snmp.h"
free_pdu(pdu_ptr);
PDU *pdu_ptr;
```

DESCRIPTION

This routine frees all memory associated with the PDU header data structure, including the actual "packlet" and all *var_bind* structures that were linked to the PDU.

SEE ALSO

build_pdu(3I), free_authentication(3I), make_pdu(3I).

NAME

link_varbind – add a var_bind entry to a PDU

SYNOPSIS

```
#include "snmp.h"
link_varbind(pdu_ptr, varbind_ptr);
PDU *pdu_ptr;
VarBind *varbind_ptr;
```

DESCRIPTION

This routine adds the *var_bind* entry created by a call to *make_varbind*(3I) to a PDU previously started by a call to *make_pdu*(3I) to flesh out the PDU. This should be called exactly once for each *varbind_ptr* being associated with the PDU as it is this association that is used to reclaim memory when the PDU is *free*d.

*PDU *pdu_ptr* is the pointer to the PDU header allocated by *make_pdu()*. *VarBind *varbind_ptr* is the pointer to the *var_bind* created by *make_varbind()*.

SEE ALSO

build_pdu(3I), make_pdu(3I), make_varbind(3I).

NAME

make_authentication – create a "trivial" authentication header

SYNOPSIS

#include "snmp.h"
AuthHeader *make_authentication(community_ptr);
OctetString *community_ptr

DESCRIPTION

This routine is used to create a library format authentication header data structure for use by *build_authentication*(3I). This particular implementation of the library creates an authentication header based on the "trivial" authentication put forth by RFC1098, which calls for a community octet string usually based on text. The header and the octet string associated with the header are *free*d when *free_authentication*(3I) is called with the authentication pointer.

*OctetString *community_ptr* is usually made from *make_octet_from_text*(3I).

SEE ALSO

build_authentication(3I), free_authentication(3I), make_octet_from_text(3I).

NAME

make_dot_from_obj_id – create a dot notation character string from an object identifier

SYNOPSIS

```
#include "snmp.h"
#include "snmpuser.h"
short make_dot_from_obj_id(oid_ptr, buffer);
OID *oid_ptr;
char buffer[];
```

DESCRIPTION

This routine is called to create a dot notation character string from a library format object identifier (OID) data structure, usually for use in a human interface. The dot notation output is the usual form (1.2.3.4.1.2.1.1), with the MIB name substituted for the most possible sub-identifiers starting from the left (1.3.6.1.2.1.1.1.0 becomes *sysDescr.0*). The MIB names included in the library are found in the *mib_oid_table* in **snmp-mib.h**. This file is used in the compilation of **oid_lib.c**, part of the **libsnmpuser.a** library, and need not be included in applications.

*OID *oid_ptr* is the pointer to the OID construct to be converted. The *char_buffer*[] is a character buffer to which a string is written.

SEE ALSO

make_varbind(3I).

NAME

make_obj_id_from_dot – create an object identifier from a dot notation character string

SYNOPSIS

```
#include "snmp.h"
#include "snmpuser.h"
OID *make_obj_id_from_dot(Object_id_string);
char *object_id_string;
```

DESCRIPTION

This routine is called to create a library format OID from a character string. The string input is usually in the format "integer.integer.integer...." (i.e., "1.3.6.1.2.1.1.1.0"), but it can be shortened by using the names as they appear in RFC1066 (i.e., *sysDescr.0*). It returns a pointer to a *malloc*ed data structure containing the internal library representation for an object identifier. This identifier can then be used in calls to *make_varbind*(3I) and *make_pdu*(3I) (in the case of traps). This *malloc*ed data structure will be *free*d by calls to *free_pdu*(3I) after the pointer has been used.

*char *object_id_string* is the string representation of the OID.

SEE ALSO

free_pdu(3I), make_pdu(3I), make_varbind(3I).

NAME

make_octet_from_hex – create an octet string from a hex character string

SYNOPSIS

```
#include "snmp.h"
OctetString *make_octet_from_hex(hex_string)
char *hex_string;
```

DESCRIPTION

This routine is used to create a library format octet string data structure from hex text strings for use by calls to *make_varbind*(3I) and *make_authentication*(3I). The format of these strings is a hex value (one or more hex digits, uppercase or lowercase) followed by a space, with more hex values and spaces to finish out the string. For example, to create an octet string or length two with a hex 15 in the first octet and a hex 3D in the second octet, the text string could be "15 3d" or "0x15 0x3D". The octet string data structure returned by this call can be *free*d by a call to *free_OctetString*. This is usually unnecessary as the structure is most often passed to another library routine for inclusion in a larger ASN.1 data structure, and that library's free counterpart will take care of memory recovery.

The *char *hex_string* is the hex to be converted to an octet string.

SEE ALSO

make_authentication(3I), make_varbind(3I).

NAME

make_octet_from_text – create an octet string from character text

SYNOPSIS

```
#include "snmp.h"
#include "snmpuser.h"
OctetString *make_octet_from_text(text_string)
char *text_string;
```

DESCRIPTION

This routine is used to create a library format octet string data structure from text strings for use by calls to *make_varbind*(3I) and *make_authentication*(3I).

The *char *text_string* is the text to be converted to an octet string.

SEE ALSO

make_authentication(3I), make_varbind(3I).

NAME

make_octetstring – make an octet string data structure from a byte string and a length

SYNOPSIS

```
#include "snmp.h"
OctetString *make_octetstring(in_string, length)
char *in_string;
long length;
```

DESCRIPTION

This routine produces a library format octet string data structure from an input byte string and length. This structure is usually passed to other library calls, such as *make_varbind*(3I) or *make_authentication*(3I), and linked into a larger library data structure of an ASN.1 entity. *free_octetstring()* recovers all memory *malloc*ed by *make_octetstring*, but it should not be used if the octet string is passed to another library routine. The *free_*()* counterparts to those routines free all memory that has been linked to the higher level ASN.1 structure.

The *char *in_string* is the octet string's value – byte string. The *long length* is the length of the byte string.

SEE ALSO

make_authentication(3I), make_pdu(3I), make_varbind(3I).

NAME

make_oid – make an object identifier data structure from an unsigned long array of sub-identifiers and a length

SYNOPSIS

```
#include "snmp.h"
OID *make_oid(sid_array, length)
unsigned long sid_array[];
long length;
```

DESCRIPTION

This routine produces a library format OID data structure from a passed sub-identifier array and length. The sub-identifier array is an array of unsigned long integers, with each element of the array corresponding to the value of each integer at each position in the dot notation display of an object identifier. For example, the OID 1.3.6.1.2.1.1 would have a value 1 in *sid_array*[*0*], 3 in *sid_array*[*1*], etc. Length is the number of sub-identifier elements present in the array. The returned structure can be *free*d by passing the pointer to *free_oid()*. This is usually unnecessary as the structure is most often passed to another library routine for inclusion in a larger ASN.1 data structure, and that library routine's *free*ing counterpart will do the memory recovery.

The *unsigned long sid_array*[] is the sub-identifier of the OID. The *long length* is the number of array elements.

SEE ALSO

make_authentication(3I), make_pdu(3I), make_varbind(3I).

NAME

make_oid_from_hex – create an object identifier from a hex character string

SYNOPSIS

```
#include "snmp.h"
OID *make_oid_from_hex(hex_string)
char *hex_string;
```

DESCRIPTION

This routine is used to create a library format OID data structure from hex text strings for use by calls to *make_varbind*(3I) and *make_authentication*(3I). The format of these strings is a hex value (one or more hex digits, uppercase or lowercase) followed by a space, with more hex values and spaces to finish out the string. For example, to create an OID consisting of three sub-identifiers (1.21.51), the text string could be "1 15 3d" or "0x1 0x15 0x3D". The OID structure returned by this call can be *free*d by a call to *free_oid*. This is usually unnecessary as the structure is most often passed to another library routine for inclusion in a larger ASN.1 data structure, and that library's free counterpart will take care of memory recovery.

The *char *hex_string* is the hex to be converted to an OID.

SEE ALSO

make_authentication(3I), make_varbind(3I).

NAME

make_pdu − create a PDU data structure to start the SNMP packet

SYNOPSIS

```
#include "snmp.h"
PDU *make_pdu(type, request_id, error_status, error_index,
   enterprise, agent_addr, generic_trap, specific_trap, time_ticks)
short type;
long request_id;
long error_status;
long error_index;
OID *enterprise;
OctetString *agent_addr;
long generic_trap;
long specific_trap;
long time_ticks;
```

DESCRIPTION

This routine is called to create the initial header block for building the SNMP ASN.1 data structure, which upon completion is used to build the actual SNMP packet. It returns a pointer to a *malloc*ed data structure of type PDU:

```
typedef struct _Pdu {
  OctetString *packlet; /* compiled SNMP packet, filled by
                           build_pdu */
  short type;           /* PDU type */
   union {
     NormPdu normpdu;   /* holds values for non-TRAP_TYPE PDU's*/
     TrapPdu trappdu;   /* holds values for TRAP_TYPE PDU's */
   } u;
   VarBind *var_bind_list;    /* pointer to the linked list of var_binds */
   VarBind *var_bind_end_ptr; /* pointer to end of the linked list */
  } Pdu;
```

The type is one of *get_request_type*, *get_next_request_type*, *get_response_type*, *set_request_type*, or *trap_type*. The request ID is the ID number assigned to the particular packet by the application. Since the application is UDP-based, retry is solely controlled by the network management application. The *error_status* is set to other than 0 in GET_RESPONSE_TYPE only to indicate that this response is in reply to a bad request. The *error_index* is used only by GET_RESPONSE_TYPE and points to the *var_bind* entry in the PDU that offends the agent. The enterprise is used by *trap_type* PDUs and is an object identifier associated with the trap-generating entity. The *agent_addr* is used by the *trap_type* PDU and consists of an octet string containing the IP address of the trap-generating entity. The *generic_trap* and *specific_trap* are used by the *trap_type* PDU and consist of integers that indicate which type of trap this PDU represents. The *time_ticks* is the *trap_type*-emitting entity's sense of time since the agent has restarted.

This routine is called once for each packet to be generated. The PDU pointer is then passed to *link_varbind*(3I) repeatedly to string *var_bind*s into the packet. *build_pdu*(3I) is then called to perform the ASN.1 encoding of the PDU and to place that result in the PDU

pointer's ***packlet** field. After the packlet has been wrapped in an authentication envelope, it is *free*d by passing the pointer to *free_pdu*(3I).

The *short type* is the PDU type – GET_REQUEST_TYPE, SET_REQUEST_TYPE, etc. The *long request_id* is the SNMP request ID number. The *long error_status* is the SNMP packet error status to send in the PDU. The *long error_index* is the index to error in the PDU. The *OID *enterprise* is the enterprise OID for trap PDUs. The *OctetString *agent_addr* is the agent IP address for trap PDUs. The *long generic_trap* is a generic trap type; the *long specific_trap* is the vendor-specific trap type. The *long time_ticks* is an SNMP "time ticks" timestamp for trap.

SEE ALSO

build_pdu(3I), free_pdu(3I), link_varbind(3I).

NAME

make_varbind – make a var_bind entry data structure

SYNOPSIS

```
#include "snmp.h"
VarBind *make_varbind(oid_ptr, type, ul_value, sl_value,
        os_value, oid_value)
OID *oid_ptr
short type;
unsigned long ul_value;
long sl_value;
OctetString *os_value
OID *oid_value
```

DESCRIPTION

This routine is called to create a *var_bind* entry to be strung onto a PDU. It returns a pointer to a *malloc*ed data structure of the type *VarBind*. This pointer is usually then used in a call to *link_varbind*(3I) to associate this *var_bind* with a PDU. The structure is *free*d when the PDU is *free*d with a call to *free_pdu*(3I).

A *var_bind* is a variable-value binding. It binds the name, the type, and the value of a variable into one data structure. The following is a list of SNMP types and associated calling variables whose values must be set:

COUNTER_TYPE	uses ul_value
GAUGE_TYPE	uses ul_value
INTEGER_TYPE	uses sl_value
TIME_TICKS_TYPE	uses sl_value
OBJECT_ID_TYPE	uses oid_value
OCTET_PRIM_TYPE	uses os_value
OCTET_CONSTRUCT_TYPE	uses os_value
IP_ADDR_PRIM_TYPE	uses os_value
IP_ADDR_CONSTRUCT_TYPE	uses os_value
OPAQUE_PRIM_TYPE	uses os_value
OPAQUE_CONSTRUCT_TYPE	uses os_value
NULL_TYPE	no passed value needed

The *short type* is a variable type such as INTEGER_TYPE, OBJECT_TYPE, etc. The *unsigned long ul_value* is a COUNTER_TYPE or GAUGE_TYPE value. The *long sl_value* is an INTEGER_TYPE or TIME_TICKS value. The *OctetString *os_value* is a value such as OCTET_PRIM_TYPE, IP_ADDR_PRIM_TYPE, etc. The *OID *oid_value* is an OBJECT_ID_TYPE value.

SEE ALSO

free_pdu(3I), link_varbind(3I), make_obj_id_from_dot(3I), make_octet_from_text(3I).

NAME

parse_authentication – create a "trivial" authentication header from an incoming SNMP packet

SYNOPSIS

```
#include "snmp.h"
AuthHeader *parse_authentication(packet, packet_len);
unsigned char *packet;
long packet_len;
```

DESCRIPTION

This routine is used to create a library format authentication header data structure from an incoming SNMP packet. If parsing errors occur, a message is output to standard error and the routine returns NULL. Otherwise, the *community_ptr* part of the structure should be checked for authentication and the pointer passed on to *parse_pdu*(3I) for further ASN.1 parsing. It should be noted that the state of the authentication header created during the building phase after a call to *build_authentication*(3I) is nearly symmetrical to the state of the authentication header after this call on the parsing side.

The *unsigned char *packet* is a pointer to an inbound SNMP packet. The *long packet_len* is the length of the packet.

SEE ALSO

build_authentication(3I), parse_pdu(3I).

NAME

parse_pdu – create a fully populated PDU data structure from an incoming SNMP packet

SYNOPSIS

```
#include "snmp.h"
PDU *parse_pdu(auth_ptr)
AuthHeader *auth_ptr;
```

DESCRIPTION

This routine takes a PDU from a fully populated authentication header data structure and parses the information into the library's internal PDU format, including all *var_bind* instances. This routine is usually called with the authentication header pointer returned by *parse_authentication*(3I), which is the same state as the header pointer after *build_authentication*(3I) is called. The PDU pointer returned from this call is the same state as the PDU pointer on a building phase after *build_pdu*(3I) has been called. If this routine fails, it returns a NULL.

The *AuthHeader *auth_ptr* is the pointer returned by the *parse_authentication()* call.

SEE ALSO

parse_authentication(3I).

NAME

print_ascii – print out the octet string as an ASCII string

SYNOPSIS

```
#include "snmp.h"
#include "snmpuser.h"
short print_ascii(octet_ptr)
OctetString *octet_ptr;
```

DESCRIPTION

This routine prints out the contents of an octet string's value as an ASCII string if the value only contains printable characters. It is called with a pointer to the octet string data structure and checks to see if the string is printable. If it is not, it returns a -1 value; otherwise it returns a 1.

The *OctetString *octet_ptr* is the pointer to the octet string to display.

SEE ALSO

print_varbind_list(3I).

NAME

print_octet_string_out – print out the octet string passed to it

SYNOPSIS

```
#include "snmp.h"
#include "snmpuser.h"
print_octet_string_out(octet_ptr, wrap)
OctetString *octet_ptr;
long wrap;
```

DESCRIPTION

This routine prints out the contents of an octet string's value in hex. It is called with a pointer to the octet string data structure and the number of bytes to display on one line (the variable "wrap"). This call is used by the *print_varbind_list*(3I) routine to actually print out octet string values.

The *OctetString *octet_ptr* is the pointer to the octet string to display. The *long wrap* is the number of bytes to display per line.

SEE ALSO

print_varbind_list(3I).

NAME

print_packet_out – print out the buffer passed to it

SYNOPSIS

```
#include "snmp.h"
#include "snmpuser.h"
print_packet_out(packet, length)
unsigned char *packet;
long length;
```

DESCRIPTION

This routine prints out the contents of a buffer in hex at 20 bytes per line. It is called with a pointer to the buffer to be displayed and the number of bytes to display out of the buffer. This call is frequently used in debugging code to display the actual SNMP message that has been received to allow hand parsing of the message. It is generally unsuitable for a production user interface.

The *unsigned char *packet* is the buffer to be displayed. The *long length* is the length of the buffer.

NAME

print_varbind_list – print out the contents of a var_bind list in a user-readable form

SYNOPSIS

```
#include "snmp.h"
#include "snmpuser.h"
print_varbind_list(var_bind_ptr)
VarBind *var_bind_ptr;
```

DESCRIPTION

This routine prints out the contents of a *var_bind* list in a user-readable form. This is a quick user interface for printing out the SNMP responses in simple SNMP utilities. If the PDU structure is pointed to by **pdu_ptr*, the call is:

print_varbind_list(pdu_ptr->var_bind_list)

The *VarBind *var_bind_ptr* is the pointer to a *var_bind* list.

SEE ALSO

parse_pdu(3I).

NOTES

NOTES

NOTES

INTERACTIVE

A Kodak Company

DC00199-2Z