OPEN DESKTOP™

Open Desktop™

Connectivity and DOS Compatibility

OPEN DESKTOP™

The Complete Graphical Operating System

SCO® TCP/IP

Derived from

LACHMAN™ SYSTEM V STREAMS TCP

User's Guide

The Santa Cruz Operation, Inc.

SCO TCP/IP was developed by Lachman Associates.
SCO TCP/IP is derived from LACHMAN™ SYSTEM V STREAMS TCP, a joint development of Lachman Associates and Convergent Technologies.

This document was typeset with an IMAGEN® 8/300 Laser Printer.

SCO Document Number: 11-25-89-1.1.0D
Printed: 4/26/90

Contents

Chapter 1

Introduction

What is TCP/IP?

TCP/IP is a set of protocols used to interconnect computer networks and to route traffic among many different computers. "TCP" means Transmission Control Protocol, and "IP" means Internet Protocol. Protocols are standards which describe allowable formats, error handling, message passing, and communication standards. Computer systems which conform to communications protocols such as TCP/IP are thus able to speak a common language. This enables them to transmit messages accurately to the correct destination, despite major differences in the hardware and software of the various machines.

Many large networks have been implemented with these protocols, including the DARPA Internet (Defense Advanced Research Projects Agency Internet). A variety of universities, government agencies, and computer firms are connected to an internetwork which follows the TCP/IP protocols. Thousands of individual machines are connected to this internet. Any machine on the internet can communicate with any other. (The term internetworking is used to refer to the action of joining two or more networks together. The result can be described as a network of networks, which is called an "internet.") Machines on the internet are referred to as "hosts" or "nodes."

TCP/IP provides the basis for many useful services, including electronic mail, file transfer, and remote login. Electronic mail is designed to transfer short text files. The file transfer application programs can transfer very large files containing programs or data. They also can provide security checks controlling file transfer. Remote login allows users on one computer to log in at a remote machine and carry on an interactive session.

The Internet Protocol (IP)

The Internet Protocol, IP, defines a connectionless packet delivery. This packet delivery connects one or more packet-handling networks into an internet. The term "connectionless" means that the sending and receiving machines are not connected by a direct circuit. Instead, individual packets of data (datagrams) are routed through different machines on the internet to the destination network and receiving machine. Thus, a message is broken up into several datagrams which are sent separately. Note that connectionless packet delivery by itself is not reliable. Individual datagrams may or may not arrive, and they probably won't arrive in the order in which they were sent. TCP add reliability.

A datagram consists of header information and a data area. The header information is used to route and process the datagram. Datagrams may be fragmented into smaller pieces, depending on the physical requirements of the networks they cross. (When a gateway sends a datagram to a network which cannot accommodate the datagram as a single packet, the datagram must be fragmented into pieces that are small enough for transmission.) The datagram fragment headers contain the information necessary to reassemble the fragments into the complete datagram. Fragments do not necessarily arrive in order; the software module implementing the IP protocol on the destination machine must reassemble the fragments into the original datagram. If any fragments are lost, the entire datagram is discarded.

The Transmission Control Protocol (TCP)

The Transmission Control Protocol, TCP, works with IP to provide reliable delivery. It provides a means to ensure that the various datagrams making up a message are reassembled in the correct order at their final destination and that any missing datagrams are sent again until they are correctly received.

The primary purpose of TCP is to provide a reliable, secure, virtual-circuit connection service between pairs of communicating processes on top of unreliable subnetworking of packets, where loss, damage, duplication, delay or misordering of packets can occur. Also, security provisions such as limiting user access to certain machines can be implemented through TCP.

TCP is concerned only with total end-to-end reliability. It makes few assumptions about the possibility of obtaining reliable datagram service. If a datagram is sent across an internet to a remote host, the intervening networks do not guarantee delivery. Likewise, the sender of the datagram has no way of knowing the routing path used to send the datagram. Source-to-destination reliability is provided by TCP in the face of unreliable media; this makes TCP well-suited to a wide variety of multi-machine communication applications.

Reliability is achieved through checksums (error detection codes), sequence numbers in the TCP header, positive acknowledgment of data received, and retransmission of unacknowledged data.

How are Messages Routed?

The following sections explain gateways and network addresses. These two concepts are the key to understanding how datagrams are routed through an internet.

Gateways

The various networks which compose an internet are connected through gateway machines. A gateway is a machine that is connected to two or more networks. It can route datagrams from one network to another. Gateways route the datagrams based on the destination network, rather than the individual machine (host) on that network. This simplifies the routing algorithms. The gateway decides which network should be the next destination of a given datagram. If the destination host for the datagram is on that network, the datagram can be sent directly to that host. Otherwise, it continues to pass from gateway to gateway until it reaches the destination network.

Network Addresses

Each host machine on a TCP/IP internet has a 32-bit network address. The address includes two separate parts: the network id and the host machine id. Machines which serve as gateways will thus have more than one address, since they are on more than one network. Internet addresses are assigned by the Network Information Center (NIC) located at SRI International in Menlo Park, California. The NIC assigns only network id's; the individual network administrators then assign the host machine id's for their network.

There are three classes of network addresses, corresponding to small, medium, and large networks. The larger the network, the larger the number of hosts on that network; likewise, smaller networks have fewer hosts. Thus, when the 32-bit network address is divided between the network id and the host machine id, larger networks will need a larger number of bits to uniquely specify all the hosts on the network. Also, there are only a small number of really large networks, and so fewer bits are needed to uniquely identify these networks. The network addresses have thus been divided into three classes, identified as A, B, or C. The following table lists these classes and their formats.

Class	Network Size Configuration
Class A	Allocates a 7-bit network id and a 24-bit host id.
Class B	Allocates a 14-bit network id and a 16-bit host id.
Class C	Allocates a 21-bit network id and an 8-bit host id.

All network addresses are 32 bits. The first bit of a Class A address is **0** (zero), to identify the address as Class A. Class B addresses begin with the digits **10**, and Class C addresses begin with **11**.

This system of network address classes provides a unique address for the entire statistical distribution of types of networks that might be expected among the various networks using this address system. There are a smaller number of large networks, having many hosts (Class A), a larger number of small networks, consisting of a lesser number of hosts (Class C), and a medium number of networks made up of a medium number of hosts (Class B).

Network addresses are often written as four decimal integers separated by periods (.), where each decimal number represents one octet of the 32-bit network address. For example, a machine might have the address 128.12.3.5.

Ports and Sockets

TCP also uses a 16-bit number called the "port" to address a connection. The port specifies the particular destination program or utility, such as **ftp** (file transfer program).

A socket is an address that specifically includes a port identifier, that is, the concatenation of an internet address with a TCP port. Port connections are displayed in the Active Connections Display of **netstat** (TC).

For more information on sockets and how TCP uses them, see the *SCO TCP/IP Socket Programmer's Guide.*

ICMP Error and Control Messages

ICMP is the Internet Control Message Protocol. It defines the error and control messages for IP. ICMP messages are sent in datagrams, like other network messages. These messages can be error messages, such as unreachable destinations, or requests for information, such as a particular network address. ICMP messages are also used to request timestamps, which are useful when synchronizing the clocks of various hosts on a network.

Protocol Layering

Communications software protocols are divided into different layers, where the lowest layer is the hardware which physically transports the data, and the highest layer is the applications program on the host machine. Each layer is very complex in its own right, and no single protocol could encompass all the tasks of the various layers. As discussed earlier, the Internet Protocol handles the routing of datagrams, while the Transmission Control Protocol, which is the layer above IP, provides reliable transmission of messages which have been divided into datagrams. The applications programs in turn rely on TCP to send information to the destination host.

To the applications programs, TCP/IP appears to provide a full-duplex virtual circuit between the machines. In actuality, all information is divided into datagrams, which may then be further fragmented during transmission. The software modules implementing IP then reassemble the individual datagrams, while the modules implementing TCP make sure that the various datagrams are reassembled in the order in which they were originally sent.

There are several higher-level specialized protocols for specific applications such as terminal traffic (**telnet**(TC)) and file transfer (**ftp**(TC)), and protocols for other network functions such as gateway-status monitoring. In this manual, however, these are not usually referred to as protocols, but rather as programs or services.

Further Reading

The following is a list of useful references where additional information about TCP/IP can be found. Some references are for highly technical users, while others are less technical. References fall into three categories:

- General computer network concepts
- TCP/IP information
- Local Area Network (LAN) and Ethernet information

General Computer Network Concepts

Technical Explanations and Texts:

Tannenbaum, Andrew S., *Computer Networks*, (Prentice-Hall, Englewood Cliffs, N.J., 1981). ISBN 0-13-165183-8.

Stallings, William, *Data and Computer Communications*, (Macmillan Publishing Company, New York, 1988), 2nd Ed. ISBN 0-02-415451-2.

Standards and specifications:

The following documents are available from the American National Standards Association, Inc., 1430 Broadway, New York, NY 10018:

International Standard 7498 (IS 7498), "Information processing systems -- Open Systems Interconnection -- Basic Reference Model," (International Organization for Standardization (ISO), Geneva, 1984).

This document defines the "Reference Model for Open Systems Interconnection," commonly known as the "OSI Reference Model."

Recommendation X.200, "Reference Model of Open Systems Interconnection for CCITT Applications," (International Telegraph and Telephone Consultative Committee (CCITT), Geneva, 1985). ISBN 92-61-02341-X.

This is basically the same document as the ISO version, but as adopted by the CCITT. The CCITT version is published in a bound volume known as

Volume VIII -- Fascicle VIII.5 of the *Red Book*. The *Red Book* is a collection of recommendations on all aspects of telegraph and telephone communications by both humans and computers. Every four years the CCITT approves an updated set of Recommendations, which it is known by the color of the binding. The 1985 Red Book was published in 10 "Volumes," many of which were broken down into several separate "Fascicles," for a total of 42 separately bound books.

TCP/IP Information

Technical Explanations and Texts

Comer, Douglas, *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, (Prentice-Hall, Englewood Cliffs, N.J, 1988). ISBN 0-13-470154-2.

Gives good explanations of the protocols, how they should be implemented, and references for further information such as "Requests For Comments" (RFCs).

Stallings, William S., et. al., *Handbook of Computer Communications Standards*, **Volume 3**: Department of Defense (DOD) Protocol Standards, (Macmillan Publishing Company, New York, 1988). ISBN 0-02-948072-8.

Davidson, John, *An Introduction to TCP/IP*, (Springer-Verlag Inc., New York, 1988). ISBN 0-387-96651-X.

Standards and Specifications

Feinler, Elizabeth J., et. al. (Eds.), *DDN Protocol Handbook*, (SRI International, Menlo Park, Calif., 1985). 3 volumes. Available at a cost of about US$110.00 from:

DDN Network Information Center
SRI International
333 Ravenswood Avenue, Room EJ291
Menlo Park, CA 94025 USA
Telephone 1-800-235-3155

or:

Defense Technical Information Center (DTIC)
Cameron Station
Alexandria, VA 22314 USA

The *DDN Protocol Handbook* is a compilation of various documents including relevant Internet RFCs and "Internet Engineering Notes" (IENs). The RFCs and IENs are identified by a number, such as RFC 791 or IEN 48. The RFCs and IENs are normally made available to network researchers and other interested parties in electronic form on the ARPA Internet, but can also be obtained in printed form from the DDN Network Information Center listed above. Many important RFCs have been issued since 1985 when the *DDN Protocol Handbook* was published, so the above volumes should be considered a starting point. Some of the newer RFCs supercede information contained in those printed in this set of volumes. Generally, RFCs numbered higher than RFC 961 will not be found in these volumes.

LAN and Ethernet Information

Technical Explanations and Texts

Stallings, William S., *Handbook of Computer Communications Standards*, **Volume 2**: Local Network Standards, (Macmillan Publishing Company, New York, 1987). ISBN 0-02-948070-1.

Chorafas, Dimitris N., *Designing and Implementing Local Area Networks*, (McGraw-Hill, Inc., New York, 1984). ISBN 0-07-010819-6.

Hammond, Joseph L., and O'Reilly, Peter J.P.,*Performance Analysis of Local Computer Networks*, (Addison-Wesley, Reading, Mass., 1986). ISBN 0-201-11530-1.

Although this selection is very mathematical and focuses on performance analysis, it is a good source of information about how local area networks actually function.

Standards and Specifications

ANSI/IEEE Std 802.2-1985 (ISO Draft International Standard 8802/2), *An American National Standard : IEEE Standards for Local Area Networks: Logical Link Control* (The Institute of Electrical and Electronic Engineers, Inc., 1984). ISBN 471-82748-7.

ANSI/IEEE Std 802.3-1985 (ISO Draft International Standard 8802/3), *An American National Standard : IEEE Standards for Local Area Networks: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications* (The Institute of Electrical and Electronic Engineers, Inc., 1985). ISBN 471-82749-5.

Chapter 2

Using Network Commands

Introduction

This chapter is an overview of UNIX internetworking commands. You should read this chapter if you are a network user, a new system administrator, or a programmer. This chapter introduces key concepts necessary to properly use the internetworking commands. It also includes introductions to several of the commands. Subjects discussed in this chapter include:

- the available network commands
- user equivalence
- identifying machine addresses within commands
- access and password problems
- remote login
- using a virtual terminal
- transferring files to and from remote machines
- remote command execution

Overview of TCP/IP Networking Commands

The TCP/IP commands are derived from both the Berkeley UNIX environment and the ARPANET networking environment. (ARPA is an acronym for [Defense] Advanced Research Projects Agency.) The commands derived from Berkeley UNIX can only be used with UNIX or UNIX-compatible systems. Those derived from ARPANET are designed to work with any operating system.

The major difference between these two different types of commands is that the 4.3BSD (Berkeley UNIX) commands propagate UNIX-style permissions across the network. The ARPANET commands do not understand the UNIX-style permissions.

Included in the TCP/IP commands is a set of commands often referred to in a Berkeley UNIX environment as the **r**-commands. The **r** stands for remote. This set includes such commands as **rcp**, **rcmd**, and **rlogin**. These commands are similar to their Berkeley UNIX counterparts. These 4.3BSD type commands are designed to be UNIX-specific and are most suitably used when you are working on a UNIX type host.

Commands such as **telnet** and **ftp** originated from ARPANET. They are designed to be operating-system independent. The protocols used in these commands are in accord with the Department of Defense (DoD) Internet specification.

The networking commands are listed alphabetically in the table below with brief descriptions. Not all of these commands are intended for use by network users. Some provide network administrative functions.

TCP/IP Networking Commands

Command	Description
ftp(TC)	file transfer program
ifconfig(ADMN)	configure network interface parameters
logger(TC)	make entries in the system log
mkhosts(ADMN)	make node name commands
netstat(TC)	show network status
rcmd(TC)	remote shell command execution

rcp(TC)	remote file copy
rlogin(TC)	remote login
ruptime(TC)	display status of nodes on local network
rwho(TC)	who is logged in on the local network nodename
slattach(ADMN)	attach serial lines as network interfaces
sldetach(ADMN)	detach serial lines as network interfaces
talk(TC)	talk to another user
telnet(TC)	user interface to DARPA TELNET protocol
trpt(ADMN)	print protocol trace

UNIX Networking Commands

A UNIX network is a group of UNIX or UNIX compatible machines linked together, usually through Ethernet. A UNIX internetwork is two or more such networks joined together by gateways to form a larger network. The internetworking gateways are invisible at the command interface level, giving the appearance of a single network. (Gateways are also referred to as IP routers or bridges.)

UNIX is a command-oriented operating system, and so to make use of the remote resources in a UNIX internetworking environment, certain network-specific commands are available. These commands are fully integrated with UNIX and can be invoked from the shell command line and shell scripts. Alternatively, they can be executed from within user programs by using the **fork**(S) or **exec**(S) system calls, or the **system**(S) library routine.

These commands are user processes of the operating system but they require network software to function. In UNIX, the name of the command is the same as the name of the file that contains the process program.

Some of the many things you can do as a user whose machine is connected in a UNIX network are:

- Remotely log onto another machine on which you have an account.
- Move logically from one remote machine to another without having to enter your password (if your system administrator has "equated" the machines or if you have created a user equivalence for that machine)
- Execute commands on any machine in the network. This means, for example, that you can execute commands from wherever the data is located. The advantage of this is that you do not need to move files. Alternatively, you can choose to execute commands where the load is lowest, or you can construct sequences of UNIX commands including *pipes* that move data between machines for processing.
- Access public data from all machines.
- Copy or transfer files from one machine to another if you have permission to do so (see **chmod**(C)).

- Share remote devices such as printers and tape drives.
- Access electronic mail systems that have been implemented for the network.
- Run applications resident on other machines.
- Access other UNIX machines that are running the appropriate communications protocol.

Note that there are three types of UNIX networking objects:

- executable commands and server programs (sometimes called *daemons*) supporting the commands
- configuration files
- library and system calls for use by programmers

Concepts Important to Using Network Commands

This section discusses several concepts which you must understand in order to use network commands properly. These include:

- user equivalence
- connections and addresses
- machine access and passwords

User Equivalence

User equivalence applies only to the commands **rcp**, **rcmd**, and **rlogin**. The command **rcp** cannot be used without user equivalence. The command **rlogin** prompts for a user name and password when user equivalence is not established; when there is user equivalence, this step is omitted. The command **rcmd** cannot be used normally without user equivalence. (If **rcmd** is invoked with a host name and no command when there is no user equivalence, the effect is the same as invoking **rlogin** without user equivalence. That is, the program will prompt for a user name and password for login.)

There are several files which are used to establish user equivalence. One is the */etc/hosts.equiv* file, which covers the system as a whole, except for the root account. The other is the *.rhosts* file in the individual account's home directory. This file covers only the individual account. (For root, this is */.rhosts*.) These two files work together with a third file, */etc/passwd*, to determine the extent of user equivalence.

There are two ways to establish user equivalence:

- An entry in *.rhosts* and in */etc/passwd*, or
- An entry in */etc/hosts.equiv* and in */etc/passwd*.

In both cases, */etc/passwd* must contain an entry for the user name from the remote machine. Do not edit this file to insert entries for equivalence. Rather, use the **sysadmsh**(ADM) utility to create user accounts and entries in the */etc/passwd* file for user equivalence. XENIX users may note that they can edit the */etc/passwd* file to add equivalence entries. This is prohibited under UNIX.

The two methods of making equivalence listed above have differing scopes. If the file *.rhosts* is used in a particular account, then user equivalence is established for that account only. However, if there is an entry in */etc/hosts.equiv* for a host name and an account on that host, then that account has user equivalence for any account (except root). If the entry in */etc/hosts.equiv* has only the remote host name, then any user on that host has user equivalence for all local accounts (except root).

Entries for *.rhosts* must include both the system name and the account name. The file */etc/hosts.equiv* does allow entries for the system name only, as discussed earlier.

If there are entries in both *.rhosts* and */etc/hosts.equiv* for the same machine or machine/account combination, then the entry from */etc/hosts.equiv* determines the extent of user equivalence.

Connections, Names and Addresses

In order to communicate between your machine and a remote machine over the internet, you must first establish a connection to the remote machine.

TCP/IP performs the mechanics of establishing connections for you, but for several programs, **telnet** and **ftp** in particular, you must be aware of connections and give the commands to establish them.

As in dialing a telephone, you must first know how to reach the recipient of your call when setting up a connection. Each host on the internet has a unique address at which it can be called to establish a connection. Because network addresses are not always easy to remember, the internet software allows for the use of names instead of addresses. Host names are established by your system administrator. If you do not know the names of the hosts that you need to use, ask your system administrator. Since hosts may be used for several purposes, it is possible to have several names (aliases) for the same host address. However, each name always stands for a single host address and will connect you to the same host each time you use it.

Access Privileges

Often in an internetworking environment, different host machines are under the jurisdiction of different departments and personnel. Those in charge of a host machine often want to limit access to their host for various security and procedural reasons. Privileges to access a machine can be granted only from the machine in question. If you are unable to access

a machine that you need to use, you or your supervisor should consult the network administrator of the host machine in question.

If you need access beyond anonymous **ftp** (see "Transferring Files" later in this chapter), the administrator can set up a machine or user equivalence between your native host and the remote host. You will need an account and password for the remote machine. If you have an account on a remote machine, you can set up a user equivalence yourself. (See "What Is User Equivalence?" earlier in this chapter.)

Virtual Terminals and Remote Login

The command **rlogin**(TC) and the ARPANET command **telnet**(TC) provide a choice of virtual terminal capability. A virtual terminal is created when you use your local machine to log onto a remote machine. The impression given is that your terminal is logically attached to the remote machine. Switching your terminal between UNIX-compatible machines can be as easy as typing the name of the machine to which you intend to connect.

Virtual terminal capability differs from remote command execution in that the user can use programs that depend on accessing the terminal directly, such as **vi**(C). These commands use the terminal in raw mode. That is, they read from the terminal character-by-character, instead of line-by-line.

The following is a brief overview of **telnet** and **rlogin**. For more information on these commands, see Chapter 4, "Using Remote Terminals."

The telnet Command

The **telnet** command provides virtual terminal access to other machines on the internet. Using **telnet**, you can log into any host on the network for which you have an account, just as if you were a local user of that machine. Once **telnet** is invoked and your connection is established, your terminal is linked to a remote machine, and data that you type is passed to that machine. Responses from the remote machine will be displayed on the screen of your terminal.

For more information on **telnet**, see Chapter 4, "Using Remote Terminals."

Remote Login with rlogin

You can use **rlogin** to remotely log into another UNIX-compatible machine. To use this command, you need a password on the host where you intend to log on. However, if you already have user equivalence on the remote machine, you do not need a password. The **rlogin** command can only be used to connect to UNIX-compatible hosts.

For more information on **rlogin**, see Chapter 4, "Using Remote Terminals."

Transferring Files

The **ftp** command enables you to manipulate files on two machines simultaneously. Using **ftp**, you can examine directories and move single or multiple files between systems. This program is designed to be mostly independent of the type of operating system.

An additional feature of **ftp** is that it allows an anonymous user who does not have an account on your machine to pick up or deposit certain files without a password from a protected area of the **ftp** home directory. The **ftp** command does not require (or understand) user equivalence.

The remote file copy command **rcp** does require user equivalence. The command **rcp** is a UNIX-specific command, and it can only be used when you are transferring files between UNIX compatible hosts.

For more information of **ftp** and **rcp**, see Chapter 5, "Transferring Files."

Executing Remote Commands

The **rcmd** command enables you to send commands to remote UNIX machines for execution and have the results returned to you. You do not have to log onto the remote machine to use **rcmd**; it acts like a pipe to the other machine. This command is useful for constructing distributed shell programs which execute commands on remote machines over the network. To use **rcmd**, you must have equivalence on the target machine (the machine on which you are trying to execute the command).

This command can only be used with remote machines that are running UNIX or a compatible operating system. The **rcmd** command passes its standard input and output to the remotely executed command, and returns to the issuing system all output that the remote command generates on standard output and standard error.

You must have */usr/hosts* in your search path to access machines directly. (For more information on **rcmd**, see Chapter 3, "Executing Remote Commands.")

Chapter 3

Executing Remote Commands

Using rcmd

The **rcmd** command enables you to send commands to remote UNIX machines for execution with the results returned to you. You do not have to log onto the remote machine to use **rcmd**. (The command acts like a pipe to another machine.) The **rcmd** command is useful for constructing distributed shell programs. You must have equivalence on the target machine to use **rcmd**. (User equivalence is discussed in Chapter 2.) The target machine is the machine on which you are trying to execute the command.

This command can be used only with remote machines running UNIX or a compatible operating system. The **rcmd** command passes the standard input (for the command to be executed) to the remote machine, and then it outputs the command's standard output and standard error to the local machine.

You must have */usr/hosts* in your search path to access machines directly.

Invoking rcmd

The **rcmd** command is given from the UNIX shell. You must specify the name of a remote machine and one or more commands to be executed, for example:

```
# rcmd machine-name command
```

In most cases, you can omit specifying **rcmd** to the shell and simply use the name of the remote machine and a command. For example:

```
# machine-name command
```

In order for you to be able to use this feature, your system administrator must have configured UNIX to accept the name of the remote machine without specifying **rcmd**. Your system administrator can advise you on how your machine is configured.

Options of rcmd

There are two options you can specify when you invoke **rcmd**. These options are:

-l *user* Normally, the command you specify is executed under your user name on the remote machine. This option allows you to specify that the command be executed under another user name, for example:

```
# rcmd machine-name -l tom command
```

Whether you use your user name or another user name, you must have established permission for yourself on the remote machine that will execute the command. The system administrator of the remote machine can advise you on how the remote machine is configured.

-n This option prevents **rcmd** from sending standard input to the remote command you specify and prevents **rcmd** from "reading up" standard input. This is done by making the command's standard input */dev/null* instead of **rcmd**'s standard input. For example:

```
# rcmd machine-name -n -l tom command
```

"Reading up" means reading and buffering the data. The **rcmd** command buffers standard input data regardless of whether the remote command reads it.

A Sample Session Using rcmd

The following example shows **rcmd** being used to run the **who**(C) command on a remote machine called *admin*. The output is placed in a file on the local machine by redirecting standard output. In this example, standard output is redirected to the file */tmp/admin.who*.

```
# rcmd admin who > /tmp/admin.who
```

Remote Printing

The **rcmd** command can be used for remote printing, as in the following example, which prints a file called *templ* on the default printer of a system called *systemx*:

```
$ cat templ | rcmd systemx lp
```

Shellscript Programming

Many useful shell programs can be written by using the ability of the TCP/IP networking commands to use pipes across the network. (See **sh**(C) and **pipe**(S) for more information on piping.) Some examples of systems based on shell programs are:

- remote line printer spooling using **rcmd** and *lp*.
- distributed text processing using **troff** (CT). In this system, macroprocessing is done at the user's node, the font processing is done on a lightly loaded back-end machine, and printing is done on a machine with a laser printer.
- using a remote tape drive to read/write a cpio archive.
- killing a process on a remote machine.
- backing up or restoring remote file systems.

Chapter 4

Using Remote Terminals

Introduction

This chapter explains how to use two TCP/IP commands that provide virtual terminal capability. "Virtual" means that no physical connection is made to the remote machine. Rather, the command simulates a physical line between your terminal and a remote machine. "Terminal" means that the command allows your terminal on your local machine to act as a terminal on a remote machine over the internet.

The virtual terminal commands described in this chapter are:

- **telnet**(TC)
- **rlogin**(TC)

The **telnet** command provides virtual terminal access to other machines on the internet. Using **telnet**, you can log into any host on the network for which you have permission, just as if you were a local user of that machine. Once **telnet** is invoked, your terminal is linked to a remote machine, and data that you type is passed to that machine. Responses from the remote machine are displayed on the screen of your terminal.

The **rlogin** command can be used in place of **telnet** to communicate with other machines running the UNIX operating system. The **rlogin** command provides a virtual terminal access that is specific to the UNIX operating system. For more information, see the section titled "The rlogin Command" later in this chapter.

Communicating Using telnet

The **telnet** program is an interactive program that enables you to communicate with a remote machine in a terminal session. Once you invoke **telnet**, you interact with **telnet** until you exit and return to the shell (the calling program).

Command and Input Modes

Whenever you open a **telnet** connection to a remote machine, **telnet** operates in input mode. Input mode transfers all the characters you type to the remote machine and displays on your terminal screen all data sent to you by the remote machine. The one exception to this is a special character called the escape character (^]). If you type this, it places **telnet** in command mode. (This escape character is not the same as the <ESC> command of your keyboard. The escape character for **telnet** is produced by typing <CTL>]).

In command mode, data that you type is interpreted by **telnet** to allow you to control **telnet** operation. Command mode is active when **telnet** is not connected to a remote host.

When **telnet** is in input mode, it communicates with the remote host based on a number of options. These options specify how operating system and terminal-specific properties of terminal-to-computer communications will be performed. An example of such an option is whether the echoing of the characters you type is done by **telnet** locally or by the remote machine. The **telnet** program and the remote machine you specify will negotiate these options and establish a compatible set of options for your terminal when you connect to a host.

Invoking the telnet Program

The **telnet** program is invoked from the UNIX shell with the command **telnet**.

Optionally, you can specify the name of the remote machine with which you intend to communicate. The following example shows a connection being made to a remote machine called *admin*:

```
telnet admin
```

Machine names are defined by your system administrator. You can examine the machine names available to you by listing the contents of the file */etc/hosts*.

When you specify a machine name to invoke **telnet**, it establishes a network connection to that machine and enters input mode. You can also invoke **telnet** without a machine name, for example:

```
telnet
```

In this case, you will be in command mode, since no machine was specified. If you do not specify a machine name, you must open a connection from within **telnet** by using **telnet**'s **open** command to access a remote host. More details are given in the next section, "Using telnet Commands."

Using telnet Commands

You can enter **telnet** commands whenever the **telnet** command mode prompt is displayed. The **telnet** command prompt looks like this:

```
telnet>
```

If you are not connected to a remote machine, the **telnet** program is in command mode. The same applies when you enter the escape character (^]) from input mode.

If command mode was not entered from input mode, **telnet** generally remains in command mode and displays the command mode prompt again after you enter each command. If you use the **open** command to establish a **telnet** connection to a remote machine, **telnet** enters input mode.

If command mode was entered from input mode, **telnet** generally returns to input mode after processing your command. If you use the **close** command to close the remote host connection, **telnet** remains in command mode after the command is processed. If you use the **quit** command, **telnet** exits and returns you to the calling program (usually the shell).

Each command you give to **telnet** in command mode must be followed by <Return>. The **telnet** program will not start a command until it receives <Return> from you. If you make a mistake while typing a command, you can use the shell line-editing commands **erase** (<BKSP>) and **kill** (<Cancel>) to edit the characters that you have typed. However, these shell line-editing commands do not work when you are in input mode. Instead, you must use special **telnet send** commands. These are discussed later in this section.

When entering a command, you do not have to enter the full command name. You need only enter enough characters to distinguish the command from other **telnet** commands. The definitive syntax for all **telnet** commands is given on the manual page **telnet**(TC) in the *TCP/IP User's Reference Manual.* These are the **telnet** commands:

open This command establishes a **telnet** connection to a remote machine. You should specify the name of the remote machine as an option of the command. This example opens a **telnet** connection to the machine *admin*:

```
telnet> open admin
```

close This command closes the connection to the remote host and stops **telnet** operation. It is functionally equivalent to the **quit** command.

quit This command terminates your **telnet** session and exits **telnet**. The quit command closes the connection to the remote machine if one is active.

z This command suspends **telnet** on systems with job control. On other systems, the command provides the user with another shell.

mode The following are subcommands and options of the **mode** command, whose syntax is described in the man page **telnet**(TC):

```
mode [ line | character ]
```

line The remote host is asked for permission to go into line-at-a-time mode.

character The remote host is asked for permission to go into character-at-a-time mode.

display This command displays all or some of the **set** or **toggle** values. (See the **set** and **toggle** commands later in this section.)

send This command sends one or more special character sequences to the remote host. The subcommands and options of the **send** command are fully described in the man page **telnet**(TC):

```
send [ ao | ayt | brk | ... ]
```

ao This command causes **telnet** to tell the remote machine to abort sending any output that is in progress. This command is useful if the remote host is sending you data that you do not wish to see and you would like **telnet** to return to command mode on the remote machine. The only output aborted is that currently being sent; you can continue to communicate with the remote machine once the current output has been stopped.

ayt This command causes **telnet** to send an "are you there?" message to the remote machine. The remote machine will send you back a message if it is active. This message is often simply a command which causes the bell on your terminal to sound, although it may be a string of text that is displayed on your terminal. This message is useful if the remote host has not responded to your input and you wish to see whether it is inactive or just busy.

brk This command sends a message to the remote machine that has the same significance as pressing the <Break> key on your terminal would for your local machine. Since **brk** is implemented between a terminal and a local machine as a set of physical signals, rather than data, pressing the <Break> key on your terminal affects only the local machine; the message is not sent to the machine to which you are connected via **telnet**. You must use the **brk** command if you want to send a break indication to a remote machine.

ec This command sends the **telnet** erase character message to the remote machine. The **ec** command has the same meaning as the shell erase (<BKSP>) command has on your local machine. Since different operating systems implement the erase-character operation differently, you may have to use the **ec** command, rather than the shell erase character, when interacting with a remote machine. The shell erase character can be used when you are in command mode because command mode's operation is local to your machine.

el This command sends the **telnet** erase-line message to the remote machine. The **el** command has the same meaning as the shell kill (erase line) command

has on your local machine. Since different operating systems implement the erase-line operation differently, you may have to use the **ec** command, rather than the shell kill command, when interacting with a remote machine. The shell kill command can be used in command mode, because command mode's operation is local to your machine.

ip This command sends the **telnet** interrupt process message to the remote machine. The **ip** command has the same meaning as the shell interrupt character does on your local machine. Since different operating systems implement the interrupt operation differently, you must use the **ip** command, rather than the shell interrupt command, when interacting with a remote machine. The shell interrupt command can be used in command mode, because command mode's operation is local to your machine.

synch This command sends a message to the remote machine telling it to ignore any input you have sent that has not yet been processed on the remote machine. This command is useful if you have typed ahead a number of commands and wish to cancel those commands without terminating the **telnet** connection to the remote machine.

escape This command sends the current **telnet** escape character.

nop This command sends the **telnet** no-operation sequence.

toggle This command toggles various flags that control **telnet** processing. The flags are toggled between TRUE and FALSE. The subcommands and options of the **toggle** command are fully described in the man page **telnet**(TC):

```
toggle [ localchars | autoflush | ... ]
```

set This command allows you change **telnet** variable values. There are subcommands and options of the **set** command, and their syntax is described in the man page **telnet**(TC):

```
set [ echo | escape | interrupt | ... ]
```

status This command shows you the status of the connection to the remote host, as well as the current options and escape character.

? This command displays information on your terminal about operating **telnet**. If you specify a **telnet** command name after the help command (**?**), then information about that command is displayed. If you just enter the help command, a list of all **telnet** commands is displayed.

Some Sample Sessions

Two sample sessions are shown below . They illustrate how **telnet** can be used in a variety of ways. Communications with a host named "there" are shown.

Description of Session 1

This is a simple session illustrating basic **telnet** use. The **telnet** program is invoked with a host name. A connection to that host is opened as a result. The **telnet** program displays the following message while establishing the connection:

"Trying..."

This indicates that **telnet** is attempting to establish a connection. A second message is displayed when the connection is established. The **telnet** program displays the current escape character. (There is no options-

status display.) At this point, **telnet** has established the connection to the remote machine, and the remote machine displays its login prompt. The user then logs into the machine using the same procedures that would be used for a local terminal on that machine. The user produces a directory listing on the remote machine. Work completed, the user then types the escape character, and **telnet** enters command mode and displays the command mode prompt. The user enters the **quit** command, and **telnet** closes the connection to the remote machine and returns to the local shell.

```
laiter$ telnet there
Trying 192.9.200.101 ...
Connected to there.
Escape character is '^]'.


System V.3.2 UNIX (there.Lachman.COM)

login: stevea
Password:
UNIX System V/386 Release 3.2
there
Copyright (C) 1984, 1986, 1987, 1988 AT&T
Copyright (C) 1987, 1988 Microsoft Corp.
All Rights Reserved
Login last used: Mon Feb 27 17:14:18 1989
there$ ls -xF
bell/          blot/          connect.h      connection.c   dhry/
hi*            hi+.c          hi.c           hin*           hin.c
hn*            hn.c           indent/        intel/         ip_icmp.h
jam/           linger*        linger.c       mailstats.c+   maketd/
maketd+/       maxmin         ot*            ot.c           ot2*
ot2.c          ping+*         ping.c         profiler/      qt/
ripsoak*       ripsoak.c      sr.sh*         st.c           sw/
t*             t.c            tcp/           tcp.sh*        tcp0227/
there$
^]
telnet> quit
Connection closed.
laiter$
```

Description of Session 2

This session illustrates alternative ways to log into and out of a remote machine with **telnet**. The **telnet** program is invoked without a machine name and enters command mode. The user does a status command, and **telnet** indicates that no connection is established. The user then uses the **telnet open** command to establish a connection and place **telnet** in input mode. The user receives a login message from the remote system. The user then logs into the machine, using the same procedures that would be used for a local terminal on that machine. Work completed, the user logs

out of the remote machine. The remote machine then closes the connection. The **telnet** program terminates automatically and returns to the local shell.

```
# telnet
telnet> status
No Connection.
Escape character is '^]'
local echo is off
telnet> open there
Trying...
Connected to there
Escape character is '^]'
System V.3 UNIX (there)
login: mary
TERM = (ansi)
$ ls
passwd
volcopy
whodo
$ ^D
Connection closed by foreign host.
#
```

The rlogin Command

The **rlogin**(TC) command connects you to a shell on a remote machine. The **rlogin** program is similar to **telnet** but specific to UNIX-compatible machines. The **rlogin** command allows you to access the same UNIX commands on a remote machine as **telnet**. However, **rlogin** is more convenient than **telnet**, because once you have logged onto a remote machine, you have the impression of working on your local machine. You do not have to know the special commands used in **telnet**. This command can only be used with remote machines running UNIX or a compatible operating system. The TERM variable in the remote shell is set to the value you are using in your local shell.

Once invoked, **rlogin** passes all data you input to the remote machine and displays all output from that machine on your terminal's screen.

Invoking the rlogin Program

The **rlogin** program is invoked from the UNIX shell. You must specify the name of a remote machine, as in this example which logs onto the machine *admin*:

```
rlogin admin
```

In some cases, you may omit specifying **rlogin** to the shell and simply put the name of the remote machine, for example, *admin*. This is only possible when your system administrator has configured UNIX to accept the name of the remote machine without specifying **rlogin**. You must also have */usr/hosts* in your search path. Your system administrator can advise you on how your machine is configured.

Leaving the rlogin Program

To leave **rlogin** and return control to your local shell, type the escape character (the tilde) and a period (~.).

Simply exiting your remote shell also causes **rlogin** to return control to your local shell.

Options for rlogin

You can specify three options when invoking **rlogin**. These options are:

-e*c* The **-e** option causes **rlogin** to use the character c instead of tilde (~) as the escape character to use when exiting **rlogin**. For example:

```
rlogin admin -e!
```

sets the exclamation point (!) as the **rlogin** escape character.

-8 The **-8** option tells rlogin to turn off the stripping of parity bits and pass 8 bit characters through to the remote end.

Whether you use your own user name or another user name, you must have established user equivalence for yourself on the remote machine to which you are logging in. The system administrator of the remote machine can advise you on the configuration of that machine. (User equivalence is discussed in Chapter 2.)

Using a Tilde in the Text

If your escape character is tilde (~), the default escape character, then you cannot normally send a line of input beginning with a tilde to the remote machine. If you need to send such a line, begin that line with a second tilde. That is, the line should begin with two tildes (~~).

Chapter 5

Transferring Files

Introduction

This chapter describes two command programs that you can use to transfer files. These programs are called **ftp** (file transfer program) and **rcp** (remote copy program). Information in this chapter includes:

- when and why to use the commands
- how to invoke and exit the commands
- how to use the command options
- sample sessions

The **ftp**(TC) command makes it possible to transfer files between your current node and other machines on the internet. It is an interactive program that enables you to input a variety of commands for file transmission and reception. In addition, **ftp** enables you to examine and modify file systems of machines on the network. When you invoke **ftp**, you interact with **ftp**'s command mode until you exit **ftp** and return to the calling program. The **ftp** program is available under a wide range of operating systems.

When you are communicating with machines running the UNIX operating system, the **rcp**(TC) command can be used in place of **ftp**. The **rcp** command is specific to UNIX-compatible operating systems.

Working with ftp

To use the **ftp** program, you need to open a connection over the internet to a remote machine before you transfer files to or from the remote machine with **ftp**. The **ftp** program allows you to have several connections active simultaneously, although generally you can only issue commands that operate on a single connection. The multiple connection facility allows you to communicate with several remote machines within a single **ftp** session. You do not have to log in and out of these machines every time you want to change connections. The connection that **ftp** uses at any given time is called the current connection.

File-Transfer Modes in ftp

The **ftp** program allows you to transfer files in one of two modes, ASCII or binary. Use ASCII mode for text files that can be represented in standard ASCII code. Binary mode is used for binary data that must be represented as strings of contiguous bits. For communication between UNIX machines, the ASCII mode can be used for most file transfers. (ASCII is the default mode.) The binary mode may be required for transferring some files, such as program-object modules, when communicating with non-UNIX machines, Your system administrator can advise you on when to use which file transfer mode.

File-Naming Conventions in ftp

If the first character of a file name that you specify to **ftp** is a hyphen (-), **ftp** uses its standard input (for reading) or the standard output (for writing).

If the first character of a file name that you specify to **ftp** is a vertical bar (|), the remainder of the file name is interpreted as a shell command. The **ftp** program creates a shell with the file name supplied as a command, and then uses its standard input (for reading) or the standard output (for writing). If the shell command includes spaces, the file name must be appropriately quoted. For example:

```
"| ls -ls"
```

The pipe symbol (|) can appear either inside or outside the quote marks.

Invoking ftp

To invoke **ftp** from the UNIX shell, enter the command **ftp**. After **ftp** is started, its prompt is displayed on your terminal. The **ftp** prompt looks like this:

```
ftp>
```

Optionally, you can specify the name of the remote machine with which you intend to communicate. The following example shows how to specify a remote machine called *admin*:

```
$ ftp admin
```

Machine names are defined by your system administrator. Before using **ftp**, you can examine the machine names available to you by listing the contents of the file */etc/hosts*.

When you specify a machine name while invoking **ftp**, the program establishes a network connection to that machine to allow you to transfer files. This is equivalent to using **ftp**'s **open** command to open a connection to the host you name. You can also invoke **ftp** without a machine name, as in this example:

```
$ ftp
```

If you do not specify a machine name from the shell, you must open a connection from within **ftp**. This is done by using **ftp**'s **open** command before you transfer any files. See the section "Description of the ftp Commands" later in this chapter for details of the **open** command.

Command Options in ftp

In addition to specifying a host name when invoking **ftp**, you can also specify a number of options that affect how **ftp** operates. These options must be placed after the command name (**ftp**) but before the host name if you are specifying one. The options you can specify when invoking **ftp** each consist of a hyphen (-) followed by a single letter, for example, **-v**.

Each of the available options has a corresponding command of the same name that can be used within **ftp**. You should compare the use of the options with the corresponding **ftp** command. See the section "Description of the ftp Commands" for details of the **ftp** commands.

-v causes **ftp** to operate in verbose mode. In verbose mode, the **ftp** protocol messages sent by the remote machine to **ftp** are displayed on your terminal. Also, if you use verbose mode, statistics are displayed after the completion of each file transfer. Verbose mode is on by default if **ftp** is run interactively. If **ftp** is run in a script, verbose mode is off, and the **-v** option turns verbose mode on. You can also change whether verbose-mode information is displayed from within **ftp** with **ftp**'s **verbose** command.

-d causes **ftp** to operate in debug mode. In debug mode, the **ftp** protocol messages sent by **ftp** to the remote machine are displayed on your terminal. If you do not use the **-d** option, this information is not displayed. You can also change whether debug mode information is displayed from within **ftp** with **ftp**'s **debug** command.

-i means that there is no interactive prompt.

-n prevents **ftp** from using autologin mode when connecting to a remote machine. When autologin mode is used, **ftp** will try to identify you automatically to the remote machine and log you into that machine. (See the section "Using the .netrc File for Automatic Login" later in this chapter for more information.) If you use the **-n** option to turn off autologin, you will have to use **ftp**'s **user** command to log into the remote machine manually.

-g causes **ftp** to disable expansion of UNIX filename wild cards such as *. If you do not use the **-g** option, **ftp** will expand your filenames containing wild cards into lists of files. You can also change whether wild card expansion is used from within **ftp** with **ftp**'s **glob** command.

Here are examples that show the use of some **ftp** options:

```
$ ftp -v -d admin
```

The above command invokes **ftp** with verbose and debug modes on and causes **ftp** to open a connection to the remote machine named *admin*. In debug mode, the commands sent to the remote machine are displayed. Verbose mode displays the responses received and the statistics in bytes received.

```
$ ftp -v -d
```

The above command invokes **ftp** with verbose and debug modes on but does not cause any connection to be opened.

```
$ ftp -n -g admin
```

The above command invokes **ftp** with autologin and wild card expansion mode off and causes **ftp** to open a connection to the remote machine named *admin*.

```
$ ftp -n -g
```

The above command invokes **ftp** with autologin and wild card expansion mode off but does not cause any connection to be opened.

Using the .netrc File for Automatic Login

You can create a file named *.netrc* in your home directory as an optional convenience feature. This file contains a line entry of the login data for each machine that you need **ftp** to open automatically. See **netrc**(F) for detailed information on this file.

When you invoke **ftp** specifying a machine, or when you subsequently **open** a machine, **ftp** reads the *.netrc* file. If you have an entry for that particular machine, **ftp** automatically conducts the login protocol exchange with its counterpart at the remote machine. It supplies your login name and password if you have entered your password in the file. If you open a machine in verbose mode, you can see the transactions taking place.

The format of the file consists of blank-separated fields introduced by keywords:

```
machine name  login name  password password
```

where **machine**, **login**, and **password** are keywords followed by the literal data needed for login:

machine The name of the node.

login The user login name for that node.

password The user's password on that node. (The password is given in normal, unencrypted text.) If you include your password in the *.netrc* file, you must read/write protect the file, by setting permissions,

to prevent discovery of your password; otherwise, **ftp** will not let you use the file. File permissions must be set to 400 or 600 for a *.netrc* file which includes passwords. See **chmod**(C) for more information on file permissions. (There is still some risk here in putting your password in the file. You must weigh the security considerations.) Ask your system administrator before using this feature.

If you do not enter your password in the file, **ftp** prompts you for your password. For example:

```
machine admin  login guido  password open
```

where *admin* is the node, *guido* is the user who logs into *admin*, and *open* is *guido*'s password.

Restrictions on ftp Commands

In addition to **ftp** commands that use standard **ftp** protocol functions, SCO TCP/IP provides a number of commands that use optional **ftp** protocol functions. Such commands should be used only to communicate with machines that are running UNIX or a compatible operating system. The commands whose use should be restricted in this way are indicated in the command descriptions described later in this chapter. When communicating with a remote machine that does not run UNIX, you should ask your system administrator whether it supports these **ftp** commands before using them. Some **ftp** servers do not support all the optional commands.

Many **ftp** servers can provide a list of supported commands. When communicating with a remote machine that has such a server, **ftp**'s **remotehelp** command can be used to obtain this information.

Description of the ftp Commands

When **ftp** displays its prompt, you can enter one of the commands described in this section. When the command is complete, the **ftp** prompt is displayed again. Depending on whether you turn on verbose or debug mode, other messages may also appear on your terminal.

Each command you give to **ftp** must be followed by <Return>. The **ftp** program does not start a command until it receives a <Return> from you. If you make a mistake while typing a command, you can use the shell line-editing commands **erase** (<BKSP>) and **kill** (<Cancel>) to edit the characters that you have typed.

You do not have to enter the full command name, only enough characters to distinguish the command from other **ftp** commands. In most cases, this is the first one or two characters of the command.

This section lists most, but not all, of the commands available for **ftp**. See the manual page **ftp**(TC) for a complete list of commands.

!

The **!** command suspends **ftp** and invokes a shell on the local machine. Any character(s) you type after entering the exclamation point are then executed locally as a shell command. You can return to **ftp** by exiting the shell. All **ftp** options and remote machine connections are returned in the same state as before you gave this command. If a shell command is typed on the same line as the ! character, only that single command is executed. The **ftp** program then returns to command mode when the given command is complete.

append

The **append** command causes **ftp** to add the contents of a local file to the end of a file on the remote machine to which you are currently connected. You can specify the files to be used when invoking the command, for example:

```
ftp> append localfile remotefile
```

Alternatively, you can just use the command name and have **ftp** prompt you for the file names, for example:

```
ftp> append
(local-file) localfile
(remote-file) remotefile
```

When you use the **append** command, the remote machine you are connected to must be a machine running UNIX or a compatible operating system.

ascii

The **ascii** command causes **ftp** to transfer files in ASCII mode. (The default mode is ASCII.)

bell

The **bell** command causes **ftp** to sound the bell at your terminal after each file transfer is completed. The next time you enter the bell command, **ftp** will stop sounding the bell after file transfers.

binary The **binary** command causes **ftp** to transfer files in binary mode. (The default mode is ASCII.)

bye The **bye** command terminates your **ftp** session and exits **ftp**. The **bye** command closes all your open connections.

cd The **cd** command changes your directory on the remote machine to a new directory name. You can specify the new directory name when invoking the command, as in the following example:

```
ftp > cd /usr/bin
```

Alternatively, you can just use the command name, in which case **ftp** prompts you for the new directory, as in the following example:

```
ftp> cd
(remote-directory) /usr/bin
```

close The **close** command closes the current connection.

debug The **debug** command turns debug mode on and off. If debug mode is on, messages sent by **ftp** to the remote machine are displayed on your terminal. If debug mode is off, this information is not displayed.

delete The **delete** command deletes a file on the remote machine to which you are currently connected. You can specify the name of the file to be deleted when invoking the command, for example:

```
ftp> delete remotefile
```

If you prefer, you can just use the command name. The **ftp** program then prompts you for the file name, as in the following example:

```
ftp> delete
(remote-file) remotefile
```

dir The **dir** command displays a detailed listing of the contents of a directory on the remote machine to which you are currently connected. (Compare **ls**,

below.) You can specify the name of the directory to be listed when invoking the command, as shown here:

```
ftp> dir /usr/bin
```

If you do not specify a directory name, the current working directory on the remote machine is listed.

You can also specify that the results of this command are placed in a file rather than displayed on your terminal. Do this by giving **ftp** a file name on your local machine in which to store the directory listing, for example:

```
ftp> dir /usr/bin printfile
```

You must specify a directory name before the output file name (here, *printfile*). Thus, if you want to list the current directory in a file called *printfile*, use:

```
ftp> dir . printfile
```

where "." stands for the current directory.

form The **form** command displays the file format used. Currently, only the nonprint format is supported.

get The **get** command copies a file from the remote machine to which you are currently connected. The file is copied to your local machine. (Use the **mget** command to copy several files at one time.) When you invoke the command, you can specify the name of a file on the remote machine and a file name on your machine where the file is to be stored, as in this example:

```
ftp> get remotefile localfile
```

If you simply specify the name of a file to be copied from the remote machine, then the file created on your local machine is given the same name as the file on the remote machine. Here is an example that does this:

```
ftp> get remotefile
```

If you prefer, you can just use the command name. The **ftp** program prompts you for the filenames to use. Here is an example:

```
ftp> get
(remote-file) remotefile
(local-file) localfile
```

If you omit the local filename, the **get** command will create a file on your machine with the same name as the file on the remote machine.

glob The **glob** command causes **ftp** to disable expansion of UNIX file-name wild cards such as '*'. This command toggles off and then on; that is, the next time you enter the **glob** command, wild card expansion will be re-enabled. When wild card expansion is enabled, **ftp** will expand your file names which contain wild cards into lists of files.

hash The **hash** command causes **ftp** to display a pound sign (#) after each block of data it sends to or receives from the remote host. The size of a data block may vary with the software release; use verbose mode with the **hash** command to see the current value. The **hash** command toggles on and then off; that is, the next time you enter the **hash** command, **ftp** will stop displaying pound signs after each data block.

help The **help** command displays information on your terminal about operating **ftp**. If you specify a command name after **help**, information about that command is displayed. If you just enter **help**, a list of all the **ftp** commands is displayed.

lcd The **lcd** command changes the working directory used by **ftp** on your local machine. You can specify a directory name to be used as the working directory, for example:

```
ftp> lcd /usr/deb
```

If you do not specify a directory name, your home directory will be used.

ls The **ls** command displays an abbreviated listing of the contents of a directory on the remote machine

to which you are currently connected. You can specify the name of the directory to be listed, for example:

```
ftp> ls /usr/bin
```

If you do not specify a directory name, the current working directory on the remote machine is listed.

You can also specify that the results of this command are placed in a file rather than displayed on your terminal by giving **ftp** a file name on your local machine in which to store the directory listing, as in this example:

```
ftp> ls /usr/bin printfile
```

You must specify a directory name before the output file (here, *printfile*). For example, if you want to list the current directory in a file called *printfile*, the command is:

```
ftp> ls . printfile
```

where "." stands for the current directory.

mdelete

The **mdelete** command deletes a list of files on the remote machine to which you are currently connected. You can specify the names of the files to be deleted when invoking the command, for example:

```
ftp> mdelete remotefile1 remotefile2 ...
```

Alternatively, you can simply use the command name. The **ftp** program prompts you for the filename(s), for example:

```
ftp> mdelete
(remote-files) remotefile1 remotefile2 ...
```

mdir

The **mdir** command obtains a directory listing for a list of remote files and places the result in a local file. You can specify the list of remote files and the local file when invoking the command, for example:

```
ftp> mdir remotefile1 remotefile2 printfile
```

(Notice that the last filename in the list is assumed to be the printfile.) It is also possible to use just the command name. The **ftp** program then prompts you for the filename, as in the following example:

```
ftp> mdir
(remote-files) remotefile1 remotefile2 printfil
local-file printfile? y
```

mget

The **mget** command copies one or more files from the remote machine to which you are currently connected and stores them on your local machine. The files stored on your local machine will have the same names as the files on the remote machine.

You can specify the list of remote files when invoking the command, for example:

```
ftp> mget remotefile1 remotefile2 ...
```

If you prefer you can just use the command name. The **ftp** program prompts you for the filenames as shown here:

```
ftp> mget
(remote-files) remotefile1 remotefile2 ...
```

mkdir

The **mkdir** command creates a directory on the remote machine to which you are currently connected. You can specify the name of the directory to be created when invoking the command, for example:

```
ftp> mkdir /u/mydir
```

Alternatively, you can just use the command. The **ftp** program then prompts you for the directory name, for example:

```
ftp> mkdir
(directory-name) /u/mydir
```

Not all **ftp** servers support the **mkdir** command.

mls

The **mls** command obtains an abbreviated directory listing for a group of remote files or directories and places the result in a local file. You can

specify the list of remote files or directories and the local file when invoking the command, for example:

```
ftp> mls remotefile1 remotefile2 printfile
```

or you can just use the command name and have **ftp** prompt you for the filenames, for example:

```
ftp> mls
(remote-files) remotefile1 remotefile2 printfile
local-file printfile? y
```

mput The **mput** command copies one or more files from your local machine to the remote machine where you are currently connected. The files stored on the remote machine will have the same names as the files on your local machine.

You can specify the list of files when invoking the command, for example:

```
ftp> mput localfile1 localfile2 ...
```

You may prefer just to use the command name and have **ftp** prompt you for the file names as in the following example:

```
ftp> mput
(local-files) localfile1 localfile2 ...
```

nmap Use this command to set or unset the filename mapping mechanism. This command is useful when connecting to a remote computer which is not UNIX compatible and has different file naming conventions. It affects the mapping of local filenames with the **get** and **mget** commands and the mapping of remote filenames with the **put** and **mput** commands. The **nmap** command is complex; see the **ftp**(TC) manual pages for more detailed information.

ntrans Use this command to set or unset the filename character translation mechanism. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions. It affects the translation of characters in local filenames with the **get** and **mget** commands and in

remote filenames with the **put** and **mput** commands. The **ntrans** command is complex; see the **ftp**(TC) manual pages for more detailed information.

open

The **open** command establishes a connection to a remote machine that can then be used for file transfer commands. You can specify the name of the remote machine when invoking the command, for example:

```
ftp > open admin
```

The command name can be used on its own. The **ftp** program then prompts you for the machine name, as in this example:

```
ftp> open
(to) admin
```

If you specify a host name when invoking the command, you can also optionally specify a port number on the remote machine. If a port number is specified, **ftp** will attempt to open a connection to the remote machine at that port rather than the default port for **ftp**. You should only use this option if you are asked to do so by your system administrator. If you do not specify a port number, **ftp** will not prompt you for one.

prompt

The **prompt** command prevents **ftp** from asking you for permission to proceed between files in multiple file commands such as **mget**. This command toggles off and then on; that is, the next time you enter the **prompt** command, **ftp** will start asking you for permission to proceed between files.

put

The **put** command transfers a file from your local machine to the remote machine where you are currently connected. (Use the **mput** command to transfer several files at one time.) You can specify the name of a file on your local machine and a file name on the remote machine when you invoke the command, for example:

```
ftp> put localfile remotefile
```

or:

```
ftp> put localfile
```

Alternatively, you can just use the command name and have **ftp** prompt you for the filename(s) to use, for example:

```
ftp> put
(local-file) localfile
(remote-file) remotefile
```

If you omit the remote filename, the **put** command will create a file on the remote machine with the same name as the file on the local machine.

pwd The **pwd** command causes **ftp** to print the name of the current working directory on the remote machine to which you are currently connected.

quit (This is the same as the **bye** command above.)

quote The **quote** command causes the arguments you enter to be sent to the remote machine for execution. Arguments must be **ftp** commands and arguments. The **ftp** commands that a remote host supports can be displayed with the **remotehelp** command. You can enter the command string to be sent when invoking the command, for example:

```
ftp> quote NLST
```

or you can just use the command name and have **ftp** prompt you for the command line to use, for example:

```
ftp> quote
(command line to send) NLST
```

You should not use this command unless asked to do so by your system administrator.

recv (This is the same as the **get** command above.)

remotehelp The **remotehelp** command requests help from **ftp** at the remote machine to which you are currently connected. The information returned by the remote machine indicates which **ftp** commands it supports.

rename The **rename** command renames a file on the remote machine to which you are currently

connected. You can enter the filenames to be used when invoking the command, for example:

```
ftp> rename remotefile1 remotefile2
```

Alternatively, you can just use the command name and have **ftp** prompt you for the file names to use, for example:

```
ftp> rename
(from-name) remotefile1
(to-name) remotefile2
```

rmdir The **rmdir** command removes a directory on the remote machine to which you are currently connected. You can specify the name of the directory to be removed when invoking the command, for example:

```
ftp> rmdir /u/mydir
```

or you can just use the command name and have **ftp** prompt you for the directory name, for example:

```
ftp> rmdir
(directory-name) /u/mydir
```

Not all **ftp** servers support the **rmdir** command.

send (The same as the **put** command above)

sendport The **sendport** command causes **ftp** to disable the ability to specify a local port to the remote machine for a data connection. This command toggles off and then on; that is, the next time you enter the **sendport** command, specification of local ports will be re-enabled. The default mode for local port specification when **ftp** is invoked is on. You should not use this command unless asked to do so by your system administrator.

status The **status** command causes **ftp** to display its current status on your terminal. This status includes the modes selected with the **bell**, **form**, **hash**, **glob**, **port**, **prompt**, and **type** commands.

type The **type** command sets the file transfer type to one that you specify. Valid values are ASCII and binary. The **type** command is another way of invoking the **ascii** and **binary** commands. If you do not specify a type when invoking this command, **ascii** is used.

trace The **trace** command causes **ftp** to enable packet tracing. This command toggles on and then off; that is, the next time you enter the **trace** command, packet tracing will be disabled. You should not use this command unless asked to do so by your system administrator.

user The **user** command allows you to identify yourself to the remote host when establishing a connection. If autologin was not disabled with the **-n** option when **ftp** was invoked, this command is not required. (See the section "Using the .netrc File for Automatic Login" earlier in this chapter.) If autologin is disabled or an autologin is not configured for you on the remote machine, you will have to use the **user** command to identify yourself to the remote machine.

Three pieces of information are used to tell the remote machine who you are: a login name, a password, and an account name.

Whereas a user name is required for all machines, password and account names are required only by some systems. Your system administrator can tell you the requirements of your machines. You should also consult your system administrator to find out valid user and account names and passwords for a machine that you intend to use.

You can enter the information for the **user** command when invoking it, as in this example:

```
ftp> user mike cat myaccount
```

Also, you can just use the command name and have **ftp** prompt you for the information to use, for example:

```
ftp> user
(username) mike
password:
Account: myaccount
```

Note that **ftp** will not echo your password when you type it, in order to protect the security of this information. If a password or account is not required on the remote machine with which you are connecting, the password or account prompts will not be displayed.

verbose The **verbose** command causes **ftp** to disable verbose mode. This command toggles off and then on; that is, the next time you enter the **verbose** command, verbose mode will be enabled. In verbose mode, the **ftp** protocol messages sent by the remote machine to **ftp** are displayed on your terminal. Also, if you use verbose mode, statistics are displayed after the completion of each file transfer. If you do not use verbose mode, this information is not displayed.

? (Another name for the **help** command.)

Some Sample ftp Sessions

This section illustrates how **ftp** can be used. Three examples are shown. Two hosts are used in these sessions, the local host *HERE* and the remote host *THERE*.

Description of Session 1

This is a simple session illustrating **ftp** use for sending and receiving files. The **ftp** command is invoked with a host name and automatically logs the user into that host, because the **-n** (disable autologin) option was not used.

Verbose mode is disabled with the **verbose** command. The user then changes working directory on the remote machine to the */etc* directory. Since the **-d** (debug) option was not used and verbose mode was disabled, no messages other than the **ftp** prompt are displayed by **ftp**.

The user does a directory listing of the */etc* directory on *THERE* using the **ls** command for an abbreviated listing. The **ftp** command shows three files in */etc* on *THERE*. The command **get** *passwd* is then issued to copy the file *passwd* from *THERE* to *HERE*. A file named *passwd* is created on *HERE* since no local filename was specified.

The **put** command is then used to copy a file called *wall* from the current working directory on the local machine to the remote working directory (*/etc*) on the remote machine (*THERE*). Once again, the same filename is used since no remote filename was specified. After the transfer is complete, a directory listing is requested that now shows four files in */etc* on *THERE* including the file *wall*, which was just sent from *HERE*.

The **bye** command is then used to exit **ftp** and return to the local shell.

```
$ ftp THERE
Connected to THERE
220 THERE FTP server (Version 4.160 #1) ready.
Name (THERE:stevea):
Password (THERE:stevea):
331 Password required for stevea.
230 User stevea logged in.
ftp> verbose
Verbose mode off.
ftp> cd /etc
ftp> ls
passwd
volcopy
whodo
ftp> get passwd
ftp> put wall
ftp> ls
passwd
volcopy
wall
whodo
ftp> bye
$
```

Description of Session 2

This session illustrates the displays caused by using a number of **ftp** options. After invoking **ftp** with the remote host name, the user issues a command to turn on debug mode. The **ftp** command displays messages indicating that this option is now enabled.

The user then changes the remote working directory to */etc*. Since debug and verbose modes are on, **ftp** displays messages showing the command sent to the remote machine (---> **CWD** */etc*) and the response received from the remote machine (250 **CWD** command successful.). Note that the **cd** command, which has the same form as UNIX's change-directory command, is sent as a **CWD** command (for change working directory) to the remote machine. The **CWD** command is **ftp**'s way of saying **cd** independently of any specific operating-system command language.

Following the **cd** command, the user does a **pwd** command to verify the working directory. Once again, The **ftp** command displays the messages sent between the local and remote machines and then displays the current remote working directory. The user then turns on the **hash** option. **ftp** displays a message indicating that this option is now enabled.

The command **get** wall myfile tells **ftp** to retrieve the file *wall* and place it in the file *myfile* in the user's local working directory. The **ftp** command displays the messages sent between the two hosts to begin the transfer and then prints a hash mark for each block of information received. After the transfer is complete, statistics are displayed showing the total time required and the data rate for the transfer.

After the file is received, the user closes the connection with the **close** command and exits **ftp** with the **bye** command.

```
$ ftp THERE
Connected to THERE
220 THERE FTP server (Version 4.160 #1) ready.
Name (THERE:stevea):
Password (THERE:stevea):
331 Password required for stevea.
ftp> debug
Debugging on (debug = 1)
ftp> cd /etc
---> CWD /etc
200 CWD command okay.
ftp> pwd
---> PWD
251
ftp> hash
Hash mark printing on (1024 bytes/hash mark).
ftp> get wall myfile
---> PORT 3,20,0,2,4,51
200 PORT command okay.
---> RETR wall
150 Opening data connection for wall (3.20.0.2,1075) (24384 bytes).
######################
226 Transfer complete.
24550 bytes received in 12.00 seconds (2 Kbytes/s)
ftp> close
---> QUIT
221 Goodbye.
ftp> bye
$
```

The rcp Command

Another command that enables you to copy files between any two UNIX machines on the internet is **rcp** (remote copy). The **rcp** command is similar to **ftp** but has a syntax much like the UNIX **cp** command. This command can only be used with remote machines running UNIX or a compatible operating system.

Invoking rcp

The **rcp** program is invoked from the UNIX shell. You must specify the names of files to copy and the location to which they are to be copied. Note that **rcp** is similar to the **cp** command. You can use it to copy from a local file to a remote file or vice versa. The following example shows a file called *remotefile* on the machine *admin* being copied to *localfile* on the local machine.

As shown, filenames for **rcp** follow a convention that is an extension of the UNIX filename convention. Filenames can take one of three forms, where a filename names a file or a directory. Valid forms for filenames are:

- *user@machine:filename*
- *machine:filename*
- *filename*

where:

machine — is the name of the machine which contains or will contain the file. If you do not specify a machine, the file is assumed to reside on your local machine.

user — is the user name on the machine you specify. If you do not specify a user name, your user name on your local machine is used. Whether you use your user name or another user name, you must have established permission for yourself on the machine where the file is located. The system administrator of the remote machine can advise you on how the remote machine is configured.

filename is a standard UNIX filename which can include a directory path. If the filename you specify does not begin with a slash (/), the filename is assumed to be relative to the specified user's home directory. The filename can include wild cards but these filenames may have to be quoted to prevent their expansion by the shell on your local machine.

If you specify only a directory name for the destination of an **rcp** command, the file(s) you specify are copied into that directory with the same names as the files.

The Options of rcp

You can specify the following options when invoking **rcp**:

-r This option allows the copying of directory trees. If the file specified for copying is a directory and you specify **-r**, the entire directory tree under that directory is copied. When **-r** is specified, the destination of the **rcp** command must be a directory. When you do not specify the **-r** option, requesting the copying of a directory is an error.

-p This option allows the preserving of modification times and modes of the source files in its copies, ignoring the *umask*. When you select *-p*, the modification times are duplicated. When you do not select *-p*, the *umask* is observed.

Some Sample rcp Sessions

In the following examples, two remote machines on the network named *THERE-C* and *THERE-C1* are used.

The first example copies a file named *list* from the user's current directory to the user's home directory on *THERE-C*:

```
$ rcp list THERE-C:list
```

The second example copies the directory hierarchy */net/src* on the local machine to a directory tree rooted in the directory *src* within the user's home directory on *THERE-C*.

```
$ rcp -r /net/src THERE-C:src
```

The third example shows the user copying the file *list* from the home directory of a user named *mike* on *THERE-C* to the */usr/tmp* directory on *THERE-C1*. The copy on *THERE-C1* is to belong to a user named *deb*.

```
$ rcp mike@THERE-C:list  deb@THERE-C1:/usr/tmp
```

Chapter 6

The Time Synchronization Protocol

Introduction

The Time Synchronization Protocol (TSP) was designed for specific use by the program **timed**(ADMN). This program is a local area network clock synchronizer for the UNIX operating system with enhanced networking capabilities provided by SCO TCP/IP. The **timed** program is built on the DARPA UDP protocol and based on a master-slave scheme.

TSP serves two purposes. First, it supports messages for the synchronization of the clocks of the various hosts in a local area network. Second, it supports messages for the election for a new master that occurs among slave time daemons when, for any reason, the master disappears.

Briefly, the synchronization software, which works in a local area network, consists of a collection of time daemons (one per machine) and is based on a master-slave structure. The present implementation keeps processor clocks synchronized within 20 milliseconds if supported by the hardware. Otherwise, 1 second is the best that can be done. A master time daemon measures the time difference between the clock of the machine on which it is running and those of all other machines. The current implementation uses ICMP *Time Stamp Requests* to measure the clock difference between machines. The master computes the network time as the average of the times provided by nonfaulty clocks. A clock is considered to be faulty when its value is more than a small specified interval apart from the majority of the clocks of the machines on the same network. It then sends to each slave time daemon the correction that should be performed on the clock of its machine. This process is repeated periodically. Since the correction is expressed as a time difference rather than an absolute time, transmission delays do not interfere with synchronization. When a machine comes up and joins the network, it starts a slave time daemon. This asks the master for the correct time and resets the machine's clock before any user activity can begin. The time daemons therefore maintain a single network time in spite of the drift of clocks away from each other.

Additionally, a time daemon on gateway machines may run as a submaster. A submaster time daemon functions as a slave on one network that already has a master and as master on other networks. In addition, a submaster is responsible for propagating broadcast packets from one network to the other.

To ensure that the service provided is continuous and reliable, it is necessary to implement an election algorithm that will elect a new master. This election occurs if the machine running the current master crashes, the master terminates (for example, because of a run-time error), or the

network is partitioned. Under this algorithm, slaves are able to realize when the master has stopped functioning; the slaves then elect a new master from among themselves. It is important to note that since the failure of the master results only in a gradual divergence of clock values, the election need not occur immediately.

All the communication occurring among time daemons uses the TSP protocol. While some messages need not be sent in a reliable way, most communication in TSP requires reliability not provided by the underlying protocol. Reliability is achieved by the use of acknowledgements, sequence numbers, and retransmission when message losses occur. When a message that requires acknowledgment is not acknowledged after multiple attempts, the time daemon that has sent the message will assume that the addressee is down. This chapter does not describe the details of how reliability is implemented, but only points out when a message type requires a reliable transport mechanism.

The message format in TSP is the same for all message types; however, in some instances, one or more fields are not used. The next section describes the message format. The following sections describe in detail the different message types, their use, and the contents of each field.

Note

The message format is likely to change in future versions of **timed**.

Message Format

All fields are based upon 8-bit bytes. Fields should be sent in network byte order if they are more than one byte long. The structure of a TSP message is the following:

1. A one-byte message type.

2. A one-byte version number, specifying the protocol version which the message uses.

3. A two-byte sequence number to be used for recognizing duplicate messages that occur when messages are retransmitted.

4. Eight bytes of packet-specific data. This field contains two four-byte time values and a one-byte hop count, or it may be unused, depending on the type of the packet.

5. A zero-terminated string of up to 256 ASCII characters with the name of the machine sending the message.

The TSP Messages

The following charts describe the message types, show their fields, and explain their usages. For the purpose of the following discussion, a time daemon can be considered to be in one of three states: slave, master, or candidate for election to master. Also, the term *broadcast* refers to the sending of a message to all active time daemons.

Adjtime Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
Seconds of Adjustment			
Microseconds of Adjustment			
Machine Name			
...			

Type: TSP_ADJTIME (1)

The master sends this message to a slave to communicate the difference between the clock of the slave and the network time which the master has just computed. The slave will adjust the time of its machine accordingly. This message requires an acknowledgment.

Acknowledgment Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_ACK (2)

Both the master and the slaves use this message for acknowledgment only. It is used in several different contexts; for example, it is used in reply to an Adjtime message.

Master-Request Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_MASTERREQ (3)

A newly-started time daemon broadcasts this message to locate a master. No other action is implied by this packet. It requires a Master Acknowledgment.

Master Acknowledgement

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_MASTERACK (4)

The master sends this message to acknowledge the Master Request message and the Conflict Resolution Message.

Set Network Time Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
Seconds of Time to Set			
Microseconds of Time to Set			
Machine Name			
...			

Type: TSP_SETTIME (5)

The master sends this message to slave time daemons to set their time. This packet is sent to newly-started time daemons and when the network date is changed. It contains the master's time as an approximation of the network time. It requires an acknowledgment. The next synchronization round will eliminate the small time difference caused by the random delay in the communication channel.

Master-Active Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_MASTERUP (6)

The master broadcasts this message to solicit the names of the active slaves. Slaves will reply with Slave Active messages.

Slave-Active Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_SLAVEUP (7)

A slave sends this message to the master in answer to a Master Active message. This message is also sent when a new slave starts up, to inform the master that it wants to be synchronized.

Master-Candidature Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_ELECTION (8)

A slave eligible to become a master broadcasts this message when its election timer expires. The message declares that the slave wishes to become the new master.

Candidature Acceptance Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_ACCEPT (9)

A slave sends this message to accept the candidature of the time daemon that has broadcast an Election message. The candidate will add the slave's name to the list of machines that it will control, should it become the master.

Candidature Rejection Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_REFUSE (10)

After a slave accepts the candidature of a time daemon, it will reply to any election messages from other slaves with this message. This rejects any candidature other than the first received.

Multiple Master Notification Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_CONFLICT (11)

When two or more masters reply to a Master Request message, the slave uses this message to inform one of them that more than one master exists.

Conflict-Resolution Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_RESOLVE (12)

A master that has been informed of the existence of other masters broadcasts this message to determine who the other masters are.

Quit Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_QUIT (13)

This message is sent by the master in three different contexts:

- to a candidate that broadcasts an Master Candidature message,
- to another master when notified of its existence, or
- to another master if a loop is detected.

In all cases, the recipient time daemon will become a slave. This message requires an acknowledgement.

Set-Date Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
Seconds of Time to Set			
Microseconds of Time to Set			
Machine Name			
...			

Type: TSP_SETDATE (22)

The program **date** (1) sends this message to the local time daemon when a super user wants to set the network date. If the local time daemon is the master, it will set the date; if it is a slave, it will communicate the desired date to the master.

Set-Date-Request Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
Seconds of Time to Set			
Microseconds of Time to Set			
Machine Name			
...			

Type: TSP_SETDATEREQ (23)

A slave that has received a Set Date message will communicate the desired date to the master, using this message.

Set Date Acknowledgment Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_DATEACK (16)

The master sends this message to a slave in acknowledgment of a Set Date Request Message. The same message is sent by the local time daemon to the program **rdate(ADMN)** to confirm that the network date has been set by the master.

Start-Tracing Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_TRACEON (17)

The controlling program **timedc** sends this message to the local time daemon to start the recording in a system file of all messages received.

Stop-Tracing Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_TRACEOFF (18)

Timedc sends this message to the local time daemon to stop the recording of messages received.

Master-Site Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_MSITE (19)

Timedc sends this message to the local time daemon to find out where the master is running.

Remote Master Site Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_MSITEREQ (20)

A local time daemon broadcasts this message to find the location of the master. It then uses the Acknowledgement message to communicate this location to **timedc**.

Test Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
(unused)			
(unused)			
Machine Name			
...			

Type: TSP_TEST (21)

For testing purposes, **timedc** sends this message to a slave to cause its election timer to expire.

Note

timed is not normally compiled to support this message.

Loop Detection Message

Byte 1	Byte 2	Byte 3	Byte 4
Type	Version No.	Sequence No.	
Hop Count	(unused)		
(unused)			
Machine Name			
...			

Type: TSP_LOOP (24)

This packet is initiated by all masters occasionally to attempt to detect loops. All submasters forward this packet onto the networks over which they are master. If a master receives a packet it sent out initially, it knows that a loop exists and tries to correct the problem.

SCO® TCP/IP

Derived from

LACHMAN™ SYSTEM V STREAMS TCP

User's Reference

The Santa Cruz Operation, Inc.

SCO TCP/IP was developed by Lachman Associates.
SCO TCP/IP is derived from LACHMAN™ SYSTEM V STREAMS TCP, a joint development of Lachman Associates and Convergent Technologies.

This document was typeset with an IMAGEN® 8/300 Laser Printer.

SCO Document Number: 11-25-89-1.1.0D
Printed: Tue May 1 14:51:32 PDT 1990

Contents

Networking Commands (**TC**)

intro

Introduction to networking commands

Description

This section describes publicly accessible networking utilities in alphabetical order.

See Also

The (ADMN) and (ADMP) sections for network administration commands.

Diagnostics

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

finger

User information look-up program

Syntax

finger [options] name ...

Description

By default, *finger* lists the login name, full name, terminal name and write status (as an * before the terminal name if write permission is denied), idle time, log-in time, and office location and phone number (if they are known) for each current UNIX 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 which the person has placed in the file **.plan** in their home directory, and the project on which they are working from the **.project** file, 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 (current users)
/etc/wtmp	who file (past logins)
/etc/passwd	for users names, offices, ...
$HOME/.lastlogin	last log-in information
$HOME/.plan	plans
$HOME/.project	projects

See Also

who(C), fingerd(ADMN)

Notes

Only the first line of the **.project** file is printed.
There is no way to pass arguments to the remote machine, as *finger* uses an internet standard port.

ftp

ARPANET file-transfer program

Syntax

ftp [**-v**] [**-d**] [**-i**] [**-n**] [**-g**] [**host**]

Description

ftp is the user interface to the ARPANET standard File Transfer Protocol. 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* immediately attempts 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 provided to the user. 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 as 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 which is required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user is 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**, **format**, **mode**, and **structure**.

ascii
Set the file transfer **type** to network ASCII. This is the default type.

bell
Arrange for a bell to sound 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 (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.

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/linefeed sequence during ascii type file transfer. When **cr** is on (the default), carriage returns are stripped from this sequence to conform to the UNIX single-linefeed record delimiter. Records on non-UNIX remote systems may contain single linefeeds; when an ascii type transfer is made, these linefeeds may 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 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 **structure** are used while transferring the file.

glob
Toggle filename 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 *sh*(C). 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 with the command:

mls *remote-files* **-**

Note: **mget** and **mput** are not meant to transfer entire directory subtrees of files. That can be done by transferring a *tar*(C) archive of the subtree (in binary mode).

hash
Toggle hash-sign (#) printing for each data block transferred. The size of a data block is BUFFERSIZE bytes. BUFFERSIZE is defined in the *ftp* source.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

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 an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the

current working directory is used. 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 (consisting of consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There are limits of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed. The macro processor interprets $ and \ 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 signal to the 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 filename 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 the command:

lcd *directory*

new local directories can be created with the command:

! mkdir *directory*

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Like **ls**, 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 **mls** output.

mode [*mode-name*]
Set the file-transfer **mode** to *mode-name*. The default mode is stream mode.

mput *local-files*
Expand wild cards 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 filename expansion. Resulting file names will then be processed according to **ntrans** and **nmap** settings.

nmap [*inpattern outpattern*]
Set or unset the filename-mapping mechanism. If no arguments are specified, the filename-mapping mechanism is unset. If arguments are specified, remote filenames are mapped during those **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during those **mget** commands and **get** commands issued without a specified local target filename. The **nmap** command is useful when connecting to a non-UNIX remote computer with different file-naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *Inpattern* is a template for incoming filenames (which may already have 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 filename. The sequences $1, $2, ..., $9 are replaced by any value resulting from the *inpattern* template. The sequence $0 is replaced by the original filename. Additionally, the sequence [*seq1*,*seq2*] is replaced by *seq1* unless *seq1* is a null string; in that case, it is replaced by *seq2*. For example, the command **nmap $1.$2.$3 [$1,$2].[$2,file]** would yield the output filename *myfile.data* for input filenames *myfile.data* and *myfile.data.old*, *myfile.file* for the input filename *myfile*, and *myfile.myfile* for the input filename *myfile.myfile*. Spaces may be included in *outpattern*, as in the example: nmap $1 |sed "s/ *$//" > $1 . Use the \ character to prevent special treatment of the $, [,] and , characters.

ntrans [*inchars* [*outchars*]]
Set or unset the filename-character translation mechanism. If no arguments are specified, the filename-character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during those **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during those **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file-naming conventions or practices. Characters in a filename 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. For example, the command **ntrans * .** would modify the filenames of files copied with *ftp*. This command translates the character "*" to the character "." in filenames. Thus, if you used the *ftp* command **get test*exe**, the file test*exe would be copied as test.exe.

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 *auto-login* option is on (default), *ftp* will also attempt to log the user automatically in to the FTP server. (See below.)

prompt
Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to retrieve or store files selectively. 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** transfer files from the host on the primary control connection to the host on the secondary control connection; and **put**, **mput**, and **append** 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 in naming the remote file after processing according to any **ntrans** or **nmap** settings. File transfer uses the current settings for **type**, **format**, **mode**, and **structure**.

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**.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified, it is supplied to the server as well.

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.

rmdir *directory-name*
Delete a directory on the remote machine.

runique
Toggle storage of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename 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 filename will be reported. Note that **runique** will not affect local files generated from a shell command. 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 that do ignore PORT commands but, incorrectly, indicate that they have been accepted.

status
Show the current status of *ftp*.

struct [*struct-name*]
Set the file transfer *structure* to *struct-name*. By default, file structure is used.

sunique
Toggle storage of files on a remote machine under unique file names. Remote *ftp* server must support *ftp* protocol STOU command for successful completion. The remote server will report a unique name. Default value is off.

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.

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. When an account field is specified, an account command will be relayed to the remote server after the log-in 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.

xmkdir *directory-name*
Make a directory on the remote machine. This sends an XMKD command instead of MKD, and is useful for backwards compatability with 4.2BSD UNIX machines.

xpwd
Print the name of the current working directory on the remote machine. This sends an XPWD command instead of PWD, and is useful for backwards compatability with 4.2BSD UNIX machines.

xrmdir *directory-name*
Delete a directory on the remote machine. This sends an XRMD command instead of RMD, and is useful for backwards compatability with 4.2BSD UNIX machines.

? [*command*]
A synonym for **help**.

Command arguments which have embedded spaces may be quoted with quotation (") marks.

Aborting a File Transfer

To abort a file transfer, use the terminal interrupt key (usually Ctrl-C). The sending of transfers is immediately halted. The receiving of transfers is halted by sending an *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 finished 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. Then *ftp* forks a shell, using *popen*(S) with the argument supplied, and reads from the stdout (or writes to the stdin). If the shell command includes spaces, the argument must be quoted, for instance, "| 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 the *sh*(C); see the **glob** command. If the *ftp* command expects a single local file (such as **put**), only the first filename generated by the globbing operation is used.

4) For **mget** commands and **get** commands with unspecified local file names, the local filename is the remote filename, which may be altered by a **case**, **ntrans**, or **nmap** setting. The resulting filename may then be altered if **runique** is on.

5) For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, which may be altered by an **ntrans** or **nmap** setting. The resulting filename may then be altered by the remote server if **sunique** is on.

File Transfer Parameters

The FTP specification specifies many parameters that may affect a file transfer. The **type** may be one of *ascii*, *image* (binary), *ebcdic*, and *local byte size* (for PDP-10's and PDP-20's mostly). The *ftp* command supports the ascii and image types of file transfer, plus local byte size 8 for **tenex** mode transfers.

The *ftp* command 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 report on data transfer statistics. Ordinarily, this is on by default, unless the standard input is not a terminal.

The **-n** option restrains *ftp* from attempting auto-login upon initial connection. If auto-login is enabled, *ftp* will check the file **.netrc** (discussed 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 log-in name (default being the user identity on the local machine), and, if necessary, prompt for a password and an account with which to log in.

The **-i** means there is no interactive prompt.

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-lines:

machine *name*
Identify a remote machine name. The auto-login process searches the **.netrc** file for a **machine** token that matches the remote ma-

chine 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** token is encountered.

login *name*
Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

password *string*
Supply a password. If this token is present, the auto-login process will supply the specified string when the remote server requires a password as part of the login process. Note that if this token is present in the **.netrc** file, *ftp* will abort the auto-login process if the **.netrc** is readable by anyone besides the user.

account *string*
Supply an additional account password. If this token is present, the auto-login process will supply the specified string when the remote server requires an additional account password, or the auto-login process will initiate an ACCT command when it does not.

macdef *name*
Define a macro. This token functions like the *ftp* **macdef** command. 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.

Notes

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 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. Avoid this problem by using the binary image type.

hostname

Set or print name of current host system

Syntax

hostname [nameofhost]

Description

The *hostname* command prints the name of the current host, as given before the log-in prompt. The super user can set the hostname by giving an argument; this is usually done at boot time in a startup script.

See Also

gethostname(SLIB), sethostname(SLIB), uname(C)

logger

make entries in the system log

Syntax

logger [**-t** tag] [**-p** pri] [**-i**] [**-f** file] [message ...]

Description

Logger provides a program interface to the *syslog*(3) 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.

Examples

logger System rebooted

logger -p local0.notice -t OPER -f /tmp/msg

See Also

syslog(SFF), syslogd(ADMN).

netstat

Show network status

Syntax

netstat [**-AainrsS**] [**-f** *address_family*] [**-I** *interface*] [**-p** *protocol_name*] [*interval*] [*namelist*] [*corefile*]

Description

The *netstat* command symbolically displays the contents of various network-related data structures. The options have the following meanings:

-A show the address of any associated protocol control blocks; used for debugging

-a show the state of all sockets; normally sockets used by server processes are not shown

-i show the state of interfaces that have been auto-configured. (Interfaces statically configured into a system, but not located at boot time, are not shown.)

-n show network addresses as numbers. (Normally *netstat* interprets addresses and attempts to display them symbolically.)

-r show the routing tables

-s show per-protocol statistics

-S show serial line configuration

-f limit statistics and control block displays to *address-family*. The only address-family currently supported is **inet**

-I show interface state for *interface* only.

-p limit statistics and control block displays to *protocol_name*, such as **tcp.**

The arguments *namelist* and *corefile* allow substitutes for the defaults **/unix** and **/dev/kmem.**

If an *interval* is specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces, pausing *interval* seconds before refreshing the screen.

There are a number of display formats, depending on the information presented. The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and, optionally, 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 data bases **/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 in the Internet dot format; refer to *rhosts*(SFF) for more information regarding this format. 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 address (currently Internet specific) of the interface and the maximum transmission unit (mtu) are also displayed.

The routing table display indicates the available routes and their status. 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), and whether the route is to a gateway (G). Direct routes are created for each interface attached to the local host. The refcnt field gives the current number of active uses of the route. Connection-oriented protocols normally hold onto a single route for the duration of a connection, while connectionless protocols obtain a route then discard it. 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 summarizing information for all interfaces and a column for the interface with the most traffic since the system was last rebooted. 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 serial line display shows the mapping of serial line units to serial devices. The baud rate and protocols in use are also shown.

See Also

slattach(ADMN), hosts(ADMN), networks(SSC), protocols(SFF), services(SFF).

Bugs

Interface statistics are dependent on the link driver. If it does not attach itself to the *ifstats* structure in the kernel, the message "No Statistics Available" will be printed for that interface.

nslookup

Query name servers interactively

Syntax

nslookup [*host-to-find* / - [*server address* / *server name*]]

Description

The *nslookup* command queries DARPA Internet domain name servers. Interactive mode allows the user to query the name server for information about various hosts and domains or to print a list of hosts in the domain. Non-interactive mode prints just the name and Internet address of a host or domain.

Arguments

Interactive mode is entered in the following cases:

a) when no arguments are given (the default name server will be used), and

b) when the first argument is a hyphen (-) and the second argument is the host name of a name server.

Non-interactive mode is used when the name of the host to be looked up is given as the first argument. The optional second argument specifies a name server.

Interactive Commands

Commands may be interrupted at any time by typing a control-C. To exit, type a control-D (EOF). The command line length must be less than 80 characters. **N.B.** an unrecognized command will be interpreted as a host name.

host [server]
Look up information for *host* using the current default server, or using *server* if it is specified.

server *domain*
lserver *domain*
Change the default server to *domain*. The **lserver** command uses the initial server to look up information about *domain* while **server** uses the current default server. If an authoritative answer can't be found, the names of servers that might have the answer are returned.

root
Changes the default server to the server for the root of the domain name space. Currently, the host sri-nic.arpa is used. (This command is a synonym for the **lserver sri-nic.arpa**.) The name of the root server can be changed with the **set root** command.

finger [*name*] [> *filename*]
finger [*name*] [>> *filename*]
Connects with the finger server on the current host. The current host is defined when a previous lookup for a host was successful and returned address information. (See the **set querytype=A** command.) *Name* is optional. > and >> can be used to redirect output in the usual manner.

ls *domain* [> *filename*]
ls *domain* [>> *filename*]
ls -a *domain* [> *filename*]
ls -a *domain* [>> *filename*]
ls -h *domain* [> *filename*]
ls -h *domain* [>> *filename*]
ls -d *domain* [> *filename*]
List the information available for *domain*. The default output contains host names and their Internet addresses. The **-a** option lists aliases of hosts in the domain. The **-h** option lists CPU and operating system information for the domain. The **-d** option lists all contents of a zone transfer. When output is directed to a file, hash marks are printed for every 50 records received from the server.

view *filename*
Sorts and lists the output of the previous **ls** command with *more*(C).

help
? Prints a brief summary of commands.

set *keyword*[=*value*]
This command is used to change state information that affects the lookups. Valid keywords are:

all Prints the current values of the various options to **set**. Information about the current default server and host is also printed.

[no]debug
Turn debugging mode on. A lot more information is printed about the packet sent to the server and the resulting answer.
(Default = nodebug; abbreviation = [no]deb)

[no]d2
Turn exhaustive debugging mode on. Essentially all fields of every packet are printed.
(Default = nod2)

[no]defname
Append the default domain name to every lookup.
(Default = defname; abbreviation = [no]def)

[no]search
With defname, search for each name in parent domains for the current domain.
(Default = search)

domain=*name*
Change the default domain name to *name*. The default domain name is appended to all look-up requests if the **defname** option has been set. The search list is set to parents of the domain with at least two components in their names.
(Default = value in hostname or /etc/resolv.conf; abbreviation = do)

type=*value*

querytype=*value*
Change the type of information returned from a query to one of:

A	the host's Internet address (the default)
CNAME	the canonical name for an alias
HINFO	the host CPU and operating system type
MD	the mail destination
MX	the mail exchanger
MG	the mail group member
MINFO	the mailbox or mail list information

MR the mail rename domain name

NS nameserver for the named zone

Other types specified in the RFC1035 document are valid but are not very useful.
(Abbreviation = q)

[no]recurse
Tell the name server to query other servers if it does not have the information.
(Default = recurse; abbreviation = [no]rec)

retry=*number*
Set the number of retries to *number*. When a reply to a request is not received within a certain amount of time (changed with **set timeout**), the request is resent. The retry value controls how many times a request is resent before giving up.
(Default = 2; abbreviation = ret)

root=*host*
Change the name of the root server to *host*. This affects the **root** command.
(Default = sri-nic.arpa; abbreviation = ro)

timeout=*number*
Change the time-out interval for waiting for a reply to *number* seconds.
(Default = 10 seconds; abbreviation = t)

[no]vc
Always use a virtual circuit when sending requests to the server.
(Default = novc; abbreviation = [no]v)

Diagnostics

If the look-up request was not successful, an error message is printed. Possible errors are:

Time-out
The server did not respond to a request after a certain amount of time (changed with **set timeout**=*value*) and a certain number of retries (changed with **set retry**=*value*).

No information
Depending on the query type set with the **set querytype** command, no information about the host is available, though the host name is valid.

Non-existent domain
The host or domain name does not exist.

Connection refused
Network is unreachable
The connection to the name or finger server could not be made at the current time. This error commonly occurs with **finger** requests.

Server failure
The name server found an internal inconsistency in its database and could not return a valid answer.

Refused
The name server refused to service the request.

The following error should not occur and it indicates a bug in the program:

Format error
The name server found that the request packet was not in the proper format.

Files

/etc/resolv.conf initial domain name and name server addresses.
/usr/lib/nslookup.hlp help file

See Also

resolver(SLIB), resolver(SFF), named(ADMN), RFC974, RFC1034, RFC1035

rcmd

Remote shell command execution

Syntax

rcmd node [**-l** user] [**-n**] [command]

Description

rcmd sends *command* to *node* for execution. It passes the resulting remote command its own standard input and outputs the remote command's standard output and standard error. *Command* can consist of more than one parameter. The second, simplified form of the command is equivalent to the first, but is only available if the system administrator previously ran *mkhosts*(ADMN). Interrupt, quit, and terminate signals received by *rcmd* are also received by the remote command; *rcmd* normally terminates at the same time as the remote command.

If *command* is omitted, *rcmd* simply runs *rlogin*(TC).

By default, the command belongs to the user on the remote node with the same name as the user who ran *rcmd*. This means that the resulting processes belong to the remote user and begin with the remote user's home directory as their working directory. Options permit you to specify another user on *node* as the owner. In any case, the remote system must have declared the local user equivalent to the remote user: an entry in **/etc/hosts.equiv** or in a **.rhosts** file in the current directory (normally the home directory) of the target user will demonstrate equivalence. [See *rcmd*(SLIB).]

rcmd understands the following options:

-l *user* The command is to belong to *user* on *node*.

-n Prevent the remote command from blocking on input by making its standard input be **/dev/null** instead of the standard input of *rcmd*.

If **-n** is not specified, *rcmd* reads the local standard input, regardless of whether the remote machine reads standard input.

Examples

The following command runs **who** on a node called "central,". putting the output in a file on the local machine.

```
rcmd central who > /tmp/c.who
```

The next example puts the same output on the remote machine.

```
rcmd central who \> /tmp/c.who
```

Files

$HOME/.rhosts	(on the target machine)
/etc/hosts.equiv	(on the target machine)

See Also

mkhosts(ADMN), rlogin(TC), rshd(ADMN), rhosts(SFF).

Requirements

rshd(ADMN) must be running on the target machine.

Notes

In some installations, this command is called **rsh**, so as to be like other versions of the software.

Unlike **rlogin** and **telnet**, **rcmd** does not actually use a pseudo-tty. The remote program can only read and/or write; therefore, programs such as **more** or **vi** will hang. Hit the <Break> key to continue.

Warnings

As the above examples illustrate, metacharacters to be interpreted by the remote shell must be hidden from the local shell. Thus:

```
rcmd central cd /etc ; cat passwd
```

clearly doesn't do what was intended because the semicolon is interpreted by the local shell, not the remote shell, and the remote shell never even sees the **cat** command. Either of the following commands properly escapes the semicolon:

```
rcmd central cd /etc \; cat passwd
rcmd central 'cd /etc ; cat passwd'
```

Credit

This document was developed at the University of California at Berkeley and is used with permission.

rcp

Remote file copy

Syntax

rcp [**-r**] [**-p**] file1 [file2 ...] target

Description

rcp copies files between two nodes. *rcp* works like the **cp** command (see *cp*(C)), with some extensions.

file1 is copied to *target*. If *target* is a directory, one or more files are copied into that directory; the copies have the same names as the originals.

File and directory names follow a convention which is an extension of the normal UNIX convention. Names take one of three forms:

user @ *host* : *path*
host : *path*
path

where

host is the name of the system which contains or will contain the file. If no host is specified (the simple *path* form of the name), the system on which the command is executed is assumed.

user is the name of a user on the specified system. If no user is specified in the name, then the user on the remote system whose name is the same as the user who executed the *rcp* command is used. (That is, this rule applies if the *host*:*path* or *path* form of the name was used.)

Access to the file system is as if by the specified user who has just logged in. Created files belong to the specified user and the specified user's group (taken from the password file). File and directory modifications can only occur if the specified user has permission to make them. If *path* does not begin with a slant (/), it is assumed to be relative to the specified user's home directory.

For you to use a user name on a remote system, the remote system must have declared it equivalent to your user name. See *rhosts*(SFF).

path is a conventional UNIX path name. *Path* can include file-name-generation sequences (*, ?, [...]); it may be necessary to quote these to prevent their expansion on the local system.

The **-r** (recursive) option copies directory hierarchies. If a file specified for copying is a directory and **-r** is specified, the entire hierarchy under it is copied. When **-r** is specified, *target* must be a directory.

When **-r** is not specified, copying directories is an error.

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*(SSC) 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*.

Note that a third system (not the source or target system of the copy) can execute *rcp*.

Examples

The following examples are executed on system alpha, by user fred. Alpha is networked to beta and gamma.

The first example copies *list* from fred's home directory on alpha to fred's home directory on beta.

```
rcp list beta:list
```

The next example copies a directory hierarchy. The original is rooted at *src* in fred's home directory on beta. The copy is to be rooted in *src* in the working directory.

```
rcp -r beta:src .
```

Finally, fred copies a file from mike's home directory on beta to **/usr/tmp** on gamma; the copy on gamma is to belong to deb. Both mike and deb must have previously declared fred on alpha equivalent to their own user names; see *rhosts*(SFF).

```
rcp mike@beta:file deb@gamma:/usr/tmp
```

Note that *junk* is not placed in deb's home directory because the *path* part of the name begins with a slash.

Files

/etc/hosts.equiv
$HOME/.rhosts

See Also

ftp(TC)

Requirements

Both nodes involved in the copy must be running the *rshd*(ADMN) server.

Diagnostics

Most diagnostics are self-explanatory. "Permission denied" means either that the remote user does not have permission to do what you want or that the remote user is not equivalent to you.

Warnings

If a remote shell invoked by *rcp* has output on startup, *rcp* will get confused. This is never a problem with *sh*(C), because it is not called as a log-in shell.

The **-r** option doesn't work correctly if the copy is purely local, because it relies on underlying support from **cp** , which is only available on BSD-derived systems. Use *cpio*(C), instead.

rlogin

Remote login

Syntax

rlogin rhost [**-ec**] [**-8**] [**-L**] [**-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 *rhost*'s with which it shares account names. (The host names must be the standard names as described in *rcmd*(TC).) When you **rlogin** as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file *rhosts* in his or her 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*(TC). 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.

The argument **-L** allows the rlogin session to be run in without any output post-processing, (*e.g.* **stty -opost**.) 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.

Notes

The control character for closing **rlogin** connections (~ by default) does not appear until after you have typed in the expected character (. by default).

When using **rlogin** to a 3.2 system, the login id is always requested, regardless of host equivalence;

See Also

netlogin (ADMN), rcmd(TC), rlogind(ADMN), rhosts(SFF).

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Files

/usr/hosts/* for *rhost* version of the command

Bugs

More of the environment should be propagated.

When using **rlogin** to a 3.2 system, the -l option is ignored.

ruptime

Show host status of local machines

Syntax

ruptime [**-a**] [**-r**] [**-l**] [**-t**] [**-u**]

Description

The *ruptime* command gives a status line for each machine on the local network; these are formed from packets broadcast by each host on the network, once every 1 - 3 minutes.

Machines for which no status report has been received for 5 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(TC), rwhod(ADMN)

rwho

Who is logged in on local network

Syntax

rwho [-a]

Description

The *rwho* command lists users logged in on machines on the local network. The format is similar to that of *who*(C). Without options, only users who have typed in the last hour are listed. For each user listed, *rwho* displays the user name; the host name; and the date and time the user logged in. If the user has not typed in the last minute, *rwho* also displays the user's idle time in hours and minutes.

The *rwho* command understands the following option:

-a List all users on active nodes. (Users idle for more than an hour are listed.)

If information from a host is more than five minutes old, the host is assumed to be down and its users are not listed.

Requirements

Each host to be listed must be running the *rwhod*(ADMN) server, which broadcasts a status packet once every 1 - 3 minutes. The local host must also be running this server to maintain the data files. Since broadcasts do not cross gateways, hosts on other networks will not be listed.

Files

/usr/spool/rwho/whod.*	information about other nodes

See Also

ruptime(TC), rwhod(ADMN).

talk

talk to another user

Syntax

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 just the person's login name. If you wish to talk to a user on another host, then *person* is of the form :

host!user or
host.user or
host:user or
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, it sends the message

Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine

to the user you wish to talk to. At this point, the recipient of the message should reply by typing

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 control L will cause the screen to be reprinted, while your erase and kill characters will work in talk as normal. In addition, control-W is defined as a word-kill character. To exit, just type your 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*(TC) command. At the outset talking is allowed. Certain commands, in particular *nroff*(TC) and *pr*(TC) 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

mesg(TC), who(TC), mail(TC), write(TC), talkd(ADMN).

Bugs

The version of *talk*(TC) released with System V STREAMS TCP uses a protocol that is incompatible with the protocol used in the version released with 4.2BSD. The new protocol is compatible with 4.3BSD. The older protocol was not portable across different machine architectures.

Talk may be confused if you attempt to use the host.user format with a fully qualified hostname.

telnet

User interface to the TELNET protocol

Syntax

telnet [host [port]]

Description

The *telnet* command is used to communicate with another host using the **TELNET** protocol. If *telnet* is invoked without arguments, it enters command mode, 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 with those arguments. (See below.)

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 ^E) may be used to turn the local echo off and on. (This would mostly be used to enter passwords without the passwords being echoed.)

In either mode, if the *localchars* toggle is TRUE (the default in line mode, discussed 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 (**toggle** *autoflush* and **toggle** *autosynch* described below) which 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 ^]). 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 (such as *hosts*(ADMN)) or an Internet address specified in the dot notation. (See *inet*(SLIB).)

close
Close a **TELNET** session and return to command mode. (This is virtually identical to **quit**.)

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*. On System V systems, this command provides the user with an escape to a shell running on the local machine.

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 to which one is connected, as well as the current mode. In addition, both the local and the remote **TELNET** options in effect are shown.

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 are the arguments which 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. (It may not work if the remote system is a 4.2 BSD system. If it doesn't 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 (for entering, say, 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 (discussed below) and the *interrupt* character is typed, a **TELNET IP** sequence (**send** *ip*, discussed 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 (discussed below) and the *quit* character is typed, a **TELNET BRK** sequence (**send** *brk*, discussed 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 (discussed below) and the *flushoutput* character is typed, a **TELNET AO** sequence (**send** *ao*, discussed 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 (discussed below), **and** if *telnet* is operating in character-at-a-time mode, then when this character is typed, a **TELNET EC** sequence (**send** *ec*, discussed 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 (discussed below), **and** if *telnet* is operating in character-at-a-time mode, then when this character is typed, a **TELNET EL** sequence (**send** *el*, discussed 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 the 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 (discussed under **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* character is recognized (and transformed into **TELNET** sequences; detailed under **set** above), *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 had not done an stty noflsh, otherwise FALSE. (See *stty*(C).)

autosynch
If *autosynch* and *localchars* are both TRUE, then when either the *intr* or *quit* characters (described above) is typed, the **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 the super user). The initial value for this toggle is FALSE.

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.

do *option*

dont *option*

will *option*

wont *option*

These commands allow the user to send the appropriate **TELNET** option sequence. If no option is specified, *telnet* will prompt for one.

Notes

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.

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.

tftp

User interface to the DARPA TFTP protocol

Syntax

tftp [host [port]]

Description

The *tftp* command is the user interface to the DARPA standard Trivial File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which *tftp* is to communicate may be specified on the command line. If this is done, *tftp* will immediately attempt to establish a connection to a TFTP server on that host. Otherwise, *tftp* will enter its command interpreter and await instructions from the user. When *tftp* is awaiting commands from the user, the prompt:

tftp>

appears. The following commands are recognized by *tftp*:

connect *host-name* [*port*]
Set the *host* (and, optionally, *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, 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* command.

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 filename 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 a filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be a *UNIX* machine. Note that the file or files must previously exist and be publicly writeable on the remote system for **put** to successfully transfer the desired file or files.

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 filename 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 filename at the same time. If the latter form is used, the last hostname 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.

Files

/etc/hosts

See Also

tftpd(ADMN).

Warnings

Because there is no user-login or validation within the *TFTP* protocol, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site.

Index

N

P

R

S

T

U

V

SCO® TCP/IP

Derived from

LACHMAN™ SYSTEM V STREAMS TCP

Administrator's Guide

The Santa Cruz Operation, Inc.

SCO TCP/IP was developed by Lachman Associates.
SCO TCP/IP is derived from LACHMAN™ SYSTEM V STREAMS TCP, a joint development of Lachman Associates and Convergent Technologies.

This document was typeset with an IMAGEN® 8/300 Laser Printer.

SCO Document Number: 11-25-89-1.1.0D
Printed: 4/26/90

Contents

Chapter 1

Network Administration

Introduction

This chapter covers topics related to setting up and administering your TCP/IP network. When you installed your system, many of these tasks were performed automatically to configure a basic networked system. If you want to customize your installation or expand your network, you should read this chapter.

If your network is not performing well, the section "Network Tuning and Troubleshooting" at the end of this chapter might provide helpful suggestions.

Kernel Configuration

The following table lists the drivers that must be included in the kernel, along with their associated device nodes.

Name	Device Node	Description
arp	/dev/inet/arp	Address Resolution Protocol
arpproc	(none)	
ip	/dev/inet/ip	Internet Protocol
icmp	/dev/inet/icmp	Internet Control Message Protocol
tcp	/dev/inet/tcp	Transmission Control Protocol
udp	/dev/inet/udp	User Datagram Protocol
llcloop	/dev/llcloop	Loopback interface
socket	/dev/socksys	Socket compatibility package
cp	/dev/socksys1	Copy protection driver
vty	/dev/ptyp*nn*	Virtual TTY driver†
ttyp	/dev/ttyp*nn*	

† The Virtual TTY driver is used by **rlogin**(TC) and **telnet**(TC). There must be one ptyp device and one ttyp device for each virtual TTY configured. Following ptyp or ttyp in the device node name is a two-digit hexadecimal number corresponding to the minor number of the device. For example, **vty** minor 0 is referenced by device node */dev/ptyp00*, and **ttyp** minor 0 is referenced by device node */dev/ttyp00*.

In addition to the drivers listed above, you may also include one or more drivers for your network interface hardware:

Name	Device Node	Description
e3A*nn*	/dev/e3A*nn*	3COM 3C501 ethernet board
e3B*nn*	/dev/e3B*nn*	3COM 3C503 ethernet board
wdn*n*	/dev/wdn*n*	Western Digital WD8003E ethernet board
sl*n*	/dev/slip	Serial Line IP interface

The character *n* in the device nodes indicates any one of the digits 0 through 3. That is, up to four boards of each type are supported. If there were two 3COM 3C503 Ethernet boards, their device nodes would be */dev/e3A0* and */dev/e3A1*.

The interrupt vectors you choose for the various Ethernet boards should be consistent with your hardware requirements.

All drivers must have references in the following files:

- An entry in */etc/conf/cf.d/mdevice*
- A file corresponding to that driver in the */etc/conf/sdevice.d* directory
- An entry in */etc/conf/cf.d/sdevice*

These drivers are normally added to the kernel configuration during installation of TCP/IP. The following display shows the information from a partial *mdevice* file:

```
ip      ocis   iSc    ip      0      23      0      256     -1
rip     ocis   iSc    rip     0      24      0      256     -1
socket  ocrwis ic     sock    0      25      0      256     -1
ttyp    ocrwi  ict    ttyp    0      26      0      16      -1
vty     ocrwi  ic     vty     0      27      0      16      -1
arpprococi     iS     app     0      0       0      256     -1
e3A0    I      iScH   e3c     0      28      1      1       -1
icmp    ocis   iSc    icmp    0      29      0      256     -1
llcloopocis    iSc    lo_     0      30      0      256     -1
slip    s      iSc    sl      0      31      0      256     -1
tcp     ocis   iSc    tcp     0      33      0      256     -1
udp     ocis   iSc    udp     0      35      0      256     -1
```

Some of the information in this file may vary depending on the system configuration. In these cases, the numbers that are used depend on the specific system configuration and are probably different from the values shown in this example.

Column six contains the major block device number, which varies depending upon the drivers that were installed in the system and the order in which they were installed. The actual value for any given driver does not actually matter as long as each driver has a different number and the number in this file matches the major number of the device name in the */dev* directory that is supposed to refer to it. The **arpproc** module is a special case, as it has no - pathname in */dev*; for this driver, the block major corresponding device number is 0.

The following is a partial *sdevice* file (comments have been removed for clarity):

```
sio     Y       1       7       1       4       3f8     3ff     0       0
sio     Y       1       7       1       3       2f8     2ff     0       0
slip    Y       256     0       0       0       0       0       0       0
socket  Y       256     0       0       0       0       0       0       0
sp      Y       0       0       0       0       0       0       0       0
spt     Y       0       0       0       0       0       0       0       0
ss      Y       0       0       0       0       0       0       0       0
str     Y       0       0       0       0       0       0       0       0
svdsp   Y       1       0       0       0       0       0       0       0
svkbd   Y       1       0       0       0       0       0       0       0
sxt     N       1       0       0       0       0       0       0       0
sy      Y       1       0       0       0       0       0       0       0
tcp     Y       256     0       0       0       0       0       0       0
timod   Y       1       0       0       0       0       0       0       0
tirdwr  Y       1       0       0       0       0       0       0       0
```

The *sdevice* file is actually assembled from component files in the directory */etc/conf/sdevice.d*. Each component file contains the line describing that driver. The "Y" or "N" in the second column indicates whether the the driver is to be linked into the kernel. Column six is the interrupt vector, which varies depending upon which cards are in the system and the vectors for which they are setup.

The format of these files is defined in **sdevice**(F) and **mdevice**(F).

Runtime Configuration of STREAMS Drivers

STREAMS configuration (linking the various STREAMS drivers and modules together) is handled by the **slink**(ADMN) program, which is normally executed at boot time by **tcp**(ADMN). The **slink** program reads the file */etc/strcf*, which contains a list of STREAMS operations to perform. Most of */etc/strcf* is the same on every system. However, under unusual circumstances, it may be necessary to edit the section of */etc/strcf* that configures the network interfaces. Examples for various types of network drivers are provided. In some cases, it is necessary to write new driver setup procedures. See **slink**(ADMN) and **strcf**(SFF) for further information.

SLIP drivers are handled automatically by the **slattach**(ADMN) command, which is invoked in the */etc/tcp* script. This portion of the script is set up during installation of the SLIP driver.

The following sections present examples of **slink** configuration commands for several different driver types.

Cloning Drivers with One Major Number per Interface

Drivers of this type, such as the 3COM 3C503 *e3B0* driver or Western Digital WD8003E *wd* driver, use cloning but do not support a method of selecting a particular network interface (such as **unit select**). Rather, this is done by allocating a separate major device number to each network interface. The **slink** function **cenet** configures an interface of this type. The command line to configure such an interface has the form:

```
cenet ip /dev/e3B0 e3B0 0
```

To add a second interface, add the following line:

```
cenet ip /dev/e3B0 e3B0 1
```

Note that the device node actually used is formed by concatenating the given device node name prefix (*/dev/e3B0*) and the given unit number (*0* or *1*). The interface name is formed in a similar manner using the supplied interface name prefix (*e3B0*) and the unit number. Thus, the first example configures an interface named *e3B0*, which accesses the device referred to by */dev/e3B0*.

Cloning Drivers Using unit select or DL_ATTACH

These drivers have only one device node and one major number, which are used for all interfaces. (The SLIP drivers are of this type, but they are a special case in that individual SLIP interfaces do not need explicit configuration in */etc/strcf*. The STREAMS configuration of SLIP drivers is handled by the **slattach**(ADMN) command, which is invoked from */etc/tcp* during system startup. The appropriate **slattach** command is automatically placed in the */etc/tcp* file during installation of TCP/IP Runtime.) The desired interface is selected using either the **unit select** or the **DL_ATTACH** primitive. (Normally, a given driver recognizes only one of these primitives.) A primitive is a type of command used to invoke a primitive operation. A primitive operation can be described as part of an interface between two programs or pieces of software. In this case, a primitive operation is a service provided by one of the protocol layers.

The **slink** functions **uenet** and **denet** configure this type of driver; **uenet** uses **unit select**, while **denet** uses **DL_ATTACH**. The command line to configure an interface of this type has the form:

```
uenet ip /dev/abc en 0
```

For a driver that uses **DL_ATTACH**, use **denet** in place of **uenet**. To configure a second interface, add the following line:

```
uenet ip /dev/abc en 1
```

The **denet** and **uenet** functions form the interface name in the same manner as does **cenet** (see previous section), but the device node name is unchanged (*/dev/abc* is open in both of these examples).

Non-Cloning Drivers

Drivers of this type have a separate device node for each minor device, with some fixed number of minor devices allocated to each network interface. The **slink** functions **senetc** and **senet** are used for this driver type. (The **senetc** function allows the specification of a convergence module.) The following command line configures such an interface:

```
senetc ip eli /dev/emd0 /dev/emd1 en0
```

If a convergence module is not required, use **senet** in place of **senetc** and omit "eli."

The last argument (*en0* in this example) gives the name by which the newly created interface is known for the purpose of performing interface-configuration operations via **ifconfig**(ADMN). For further in-

formation, refer to the section entitled "Setting Interface Parameters" later in this chapter.

Assuming that there are four minor devices assigned to each network interface, a second interface would be configured as follows:

```
senetc ip eli /dev/emd4 /dev/emd5 en1
```

Setting Interface Parameters

All network interface drivers, including the loopback interface, require that their host addresses be defined at boot time. This is done with **ifconfig**(ADMN) commands included in the */etc/tcp* shell script. These commands are normally set up automatically during installation. This configuration applies only to simple, basic configurations. For example, if you want to use the network feature of **ifconfig**, you need to edit */etc/tcp* manually and modify the **ifconfig** commands there.

ifconfig can also be used to set options for an interface at boot time. Options are set independently for each interface and apply to all packets sent using that interface. These options include disabling the use of the Address Resolution Protocol. This may be useful if a network is shared with hosts running software that does not yet provide this function. Alternatively, translations for such hosts can be set in advance or published by a UNIX System host by use of the **arp**(ADMN) command.

Local Subnetworks

In TCP/IP, the DARPA Internet support includes the concept of the subnetwork. This is a mechanism that enables several local networks to appear as a single Internet network to off-site hosts. Subnetworks are useful because they allow a site to hide the local topology, requiring only a single route in external gateways. This also means that local network numbers may be locally administered.

To set up local subnetworks, you first need to know how much of the available address space is to be partitioned. The term "address" is used here to mean the Internet host part of the 32-bit address. Sites with a class A network number have a 24-bit address space with which to work, sites with a class B network number have a 16-bit address space; and sites with a class C network number have an 8-bit address space. To define local subnets you must steal some bits from the local host address space for use in extending the network portion of the internet address.

This reinterpretation of internet addresses is done only for local networks. It is not visible to off-site hosts. For example, if your site has a class B network number, hosts on this network have an Internet address that contains the network number, 16 bits, and the host number, another 16 bits. To define 254 local subnets, each possessing at most 255 hosts, 8 bits may be taken from the local part to be used for the subnetwork ID. (The use of subnets 0 and all-*1*'s, 255 in this example, is discouraged to avoid confusion about broadcast addresses.) New network numbers are then constructed by concatenating the original 16-bit network number with the extra 8 bits containing the local subnetwork number.

The existence of local subnetworks is communicated to the system when a network interface is configured with the **netmask** option to the **ifconfig**(ADMN) program. A network mask defines the portion of the internet address that is to be considered the network part for that network. This mask normally contains the bits corresponding to the standard network part as well as the portion of the local part that was assigned to subnets. If no mask is specified when the address is set, a mask is set according to the class of the network. For example, at Berkeley (class B network 128.32), 8 bits of the local part are reserved for defining subnetworks. Consequently, the */etc/tcp* file contains lines of the form:

```
/etc/ifconfig e3B0 netmask 0xffffff00 128.32.1.7
```

This specifies that for interface *e3B0*, the upper 24 bits of the internet address should be used in calculating network numbers (netmask 0xffffff00). The internet address of the interface is 128.32.1.7 (host 7 on

network 128.32.1). Hosts *m* on subnetwork *n* of this network would then have addresses of the form 128.32.*n.m*. For example, host 99 on network 129 would have an address 128.32.129.99. For hosts with multiple interfaces, the network mask should be set for each interface, although in practice only the mask of the first interface on each network is actually used.

Internet Broadcast Addresses

The broadcast address for internet networks is defined according to RFC-919 as the address with a host part of all *1*'s. The address used by 4.2BSD was the address with a host part of *0*. The UNIX System uses the standard broadcast address (all *1*'s) by default, but allows the broadcast address to be set (with **ifconfig**) for each interface. This allows networks consisting of both 4.2BSD and UNIX System hosts to coexist while the upgrade process proceeds. In the presence of subnets, the broadcast address uses the subnet field as for normal host addresses, with the remaining host part set to *1*'s (or *0*'s, on a network that has not yet been converted). The UNIX System hosts recognize and accept packets sent to the logical-network broadcast address as well as those sent to the subnet broadcast address, and, when using an all-*1*'s broadcast, also recognize and receive packets sent to host *0* as a broadcast.

Routing

If your environment allows access to networks not directly attached to your host, you need to set up routing information to allow packets to be properly routed. Two schemes are supported by the system. The first employs the routing table management daemon **routed**(ADMN) to maintain the system routing tables. The routing daemon uses a variant of the Xerox Routing Information Protocol to maintain up-to-date routing tables in a cluster of local-area networks. By using the */etc/gateways* file, the routing daemon can also initialize static routes to distant networks. (See the next section for further discussion.) When the routing daemon is started (usually from */etc/tcp*), it reads */etc/gateways* if it exists and installs those routes defined there. It then broadcasts on each local network to which the host is attached to find other instances of the routing daemon. If any responses are received, the routing daemons cooperate in maintaining a globally consistent view of routing in the local environment. This view can be extended to include remote sites also running the routing daemon by setting up suitable entries in */etc/gateways*. See **route**(ADMN) for a more thorough discussion.

The second approach is to define a default or wildcard route to a smart gateway and depend on the gateway to provide ICMP routing redirect information to create dynamically a routing data base. This is done by adding an entry to */etc/tcp* as in the following example:

```
/etc/route add default smart-gateway 1
```

See **route**(ADMN) for more information. The system uses the default route as a last resort in routing packets to their destinations. Assuming the gateway to which packets are directed can to generate the proper routing redirect messages, the system then adds routing table entries based on the information supplied. This approach has certain advantages over the routing daemon, but it is unsuitable in an environment where there are only bridges. (For example, pseudo-gateways do not generate routing-redirect messages.) Further, if the smart gateway goes down, there is no alternative, save manual alteration of the routing table entry, to maintain service.

The system always listens to, and processes, routing redirect information, and so it is possible to combine both of the above facilities. For example, the routing table management process might be used to maintain up-to-date information about routes to geographically local networks, while

employing the wildcard routing techniques for distant networks. The **netstat**(TC) program displays routing table contents as well as various routing-oriented statistics. The following example displays the contents of the routing tables:

```
netstat -r
```

Alternatively, the following shows the number of routing table entries dynamically created as a result of routing redirect messages and so forth:

```
netstat -r -s
```

Using UNIX System Machines as Gateways

Any UNIX System machine that is connected to more than one network functions as a gateway. At a gateway machine, packets received on one network that are destined for a host on another network are automatically forwarded. If a packet cannot be forwarded to the desired destination, an ICMP error message is sent to the originator of the packet. When a packet is forwarded back through the interface on which it arrived, an ICMP redirect message is sent to the source host if it is on the same network. This improves the interaction of UNIX System gateways with hosts that configure their routes via default gateways and redirects.

Local-area routing within a group of interconnected Ethernets and other such networks can be handled by **routed**(ADMN). Gateways between the ARPANET or MILNET and one or more local networks require an additional routing protocol, the Exterior Gateway Protocol (EGP), to inform the core gateways of their presence and to acquire routing information from the core. (EGP is not currently supported in this product.)

Network Servers

In the UNIX System, most of the server programs are started by a super server, called the "internet daemon." The internet daemon, */etc/inetd*, acts as a master server for programs specified in its configuration file, */etc/inetd.conf*, listening for service requests for these servers, and starting up the appropriate program whenever a request is received. The configuration file includes lines containing a service name (as found in */etc/services*), the type of socket the server expects (for example, stream or dgram), the protocol used with the socket (as found in */etc/protocols*), whether to wait for each server to complete before starting up another, the user name under which the server should run, the server program's name, and at most five arguments to pass to the server program. Some trivial services are implemented internally in **inetd**(SFF), and their servers are listed as internal. For example, an entry for the file-transfer protocol server would appear as:

```
ftp    stream    tcp    nowait    root    /etc/ftpd    ftpd
```

Consult **inetd**(ADMN) for more details on the format of the configuration file and the operation of the Internet daemon.

Network Databases

Several data files are used by the network library routines and server programs. Most of these files are host independent and updated only rarely. The following table lists the data files used.

File	Manual Reference	Use
/etc/hosts	hosts (SFF)	host names
/etc/networks	networks (SFF)	network names
/etc/services	services (SFF)	list of known services
/etc/protocols	protocols (SFF)	protocol names
/etc/hosts.equiv	rshd (ADMN)	list of "trusted" hosts
/etc/ftpusers	ftpd (ADMN)	list of "unwelcome" **ftp** users
/etc/inetd.conf	inetd (ADMN)	list of servers started by **inetd**

The files distributed are set up for ARPANET or other internet hosts. Local networks and hosts should be added to describe the local configuration. Network numbers must be chosen for each Ethernet. For sites not connected to the Internet, these can be chosen more or less arbitrarily; otherwise, the normal channels should be used for allocation of network numbers.

The /etc/hosts.equiv File

There are several files that are used to establish user equivalence. One is the */etc/hosts.equiv* file, which covers the system as a whole, except for the root account. The other is the *.rhosts* file in the individual user account's home directory. This file covers only the individual user account. (For root, this is */.rhosts*.) These two files work together with a third file, */etc/passwd*, to determine the extent of user equivalence.

There are two ways to establish user equivalence:

- An entry in *.rhosts* and in */etc/passwd*
- An entry in */etc/hosts.equiv* and in */etc/passwd*

In both cases, */etc/passwd* must contain an entry for the user name from the remote machine. However, the two methods have differing scopes. If the file *.rhosts* is used in a particular account, then user equivalence is established for that account only. However, if there is an entry in */etc/hosts.equiv* for a host name and an account on that host, then that account has user equivalence for any account (except root). If the entry

in */etc/hosts.equiv* has only the remote host name, then any user on that host has user equivalence for all local accounts (except root). Such a host is considered a "trusted host."

Note

Entries in */etc/hosts.equiv* can create large holes in system security. Be sparing in their use. In most circumstances, it is unwise to create entries that allow all users on remote machines to access all accounts on your local machine.

For example, suppose you have an account under the user name "Test1" on machine "Admin." You want to establish user equivalence on the remote machine "Systemb." The administrator for the machine Systemb must add an entry to the */etc/passwd* file for an account name Test1. Note that this file cannot be edited directly under UNIX. You must use the **sysadmsh**(ADM) utility to add a user to the */etc/passwd* file. They must also include the following entry in the file */etc/hosts.equiv* on Systemb:

```
Admin   Test1
```

This gives user equivalence for all accounts except root to user Test1 on the machine Systemb. Suppose that Test1 really only needed access to the account Testb on Systemb. Then it would be better to remove the above entry from */etc/hosts.equiv* on Systemb and use the following entry in the file *.rhosts* in the home directory for Testb:

```
Admin   Test1
```

Note that entries for *.rhosts* must include both the system name and the account name. The file */etc/hosts.equiv* does allow entries for the system name only, as discussed earlier.

If there are entries in both *.rhosts* and */etc/hosts.equiv* for the same machine or machine/account combination, then the entry from */etc/hosts.equiv* determines the extent of user equivalence.

The /etc/ftpusers File

The **ftp** server included in the system provides support for an anonymous **ftp** account. Because of the inherent security problems with such a facility, you should read this section carefully if you want to provide such a service.

An anonymous account is enabled by creating a user called **ftp**. When a client uses the anonymous account, a **chroot**(ADM) system call is performed by the server to restrict the client from moving outside that part of the filesystem where the **ftp** home directory is located. Because a **chroot** call is used, certain programs and files used by the server process must be placed in the **ftp** home directory. Further, you must be sure that all directories and executable images are unwritable. The following directory setup is recommended:

```
# cd ~ftp
# chmod 555 .; chown ftp .; chgrp ftp .
# mkdir bin etc pub lib dev
# chown root bin etc lib dev
# chmod 555 bin etc lib dev
# chown ftp pub
# chmod 777 pub
# cd bin
# cp /bin/sh /bin/ls .
# chmod 111 sh ls
# cd ../etc
# cp /etc/passwd /etc/group .
# chmod 444 passwd group
# cd ../lib
# cp /shlib/libc_s .
# cd ..
# find /dev/socksys -print | cpio -dumpv .
```

When local users want to place files in the anonymous area, they must place them in a subdirectory. In the setup here, the directory *~ftp/pub* is used.

Another issue to consider is the */etc/passwd* file placed here. It can be copied by users who use the anonymous account. They can then try to break the passwords of users on your machine for further access. A good choice of users to include in this copy might be root, daemon, uucp, and the **ftp** user. All passwords here should probably be *.

Aside from the problems of directory modes and such, the **ftp** server provides a loophole for interlopers if certain user accounts are allowed. The file */etc/ftpusers* is checked on each connection. If the requested user name is located in the file, the request for service is denied. It is suggested that this file contain at least the following names:

```
uucp
root
```

Accounts with nonstandard shells should be listed in this file. Accounts without passwords need not be listed in this file; the **ftp** server does not service these users.

Network Tuning and Troubleshooting

It is likely that from time to time you will encounter problems using your network. The first thing to do is check your network connections. On networks such as the Ethernet a loose cable tap or poorly placed power cable can result in severely deteriorated service. The **ping**(ADMN) command is particularly useful for confirming the existence of network connections. If there is no hardware problem, check next for routing problems and addressing problems.

The **netstat**(TC) program can also be helpful in tracking down hardware malfunctions. In particular, look at the **-i** and **-s** options in the manual page. The **netstat**(TC) program also shows detailed information about network behavior. Examples of **netstat** displays appear later in this chapter.

If you think a communication protocol problem exists, consult the protocol specifications and attempt to isolate the problem in a packet trace. The **SO_DEBUG** option can be supplied before establishing a connection on a socket, in which case the system traces all traffic and internal actions (such as timers expiring) in a circular trace buffer. This buffer can then be printed out with the **trpt**(ADMN) program. Most of the servers distributed with the system accept a **-d** option forcing all sockets to be created with debugging turned on. Consult the appropriate manual pages for more information.

STREAMS Tuning

The **crash**(ADM) command can be used to display STREAMS usage of buffers of various sizes. Typical symptoms of inadequate STREAMS buffer space include the following: lost connections for no reason; processes that communicate over the network hang; and programs that communicate over the network suddenly malfunction. Use the UNIX Link Kit **configure** command to increase STREAMS buffer resources.

Active Connections Display

The active connections display is the default display of the **netstat**(TC) command. It displays a line of information for each active connection on the local machine under the headings described below.

netstat -a

Active Internet connections (including servers) are as follows:

```
scobox$ netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address  (state)
ip         0      0  *.*                    *.*
tcp        0      0  scobox.telnet          scoter.2460      ESTABLISHED
tcp        0      0  *.smtp                 *.*              LISTEN
tcp        0      0  *.1024                 *.*              LISTEN
tcp        0      0  *.sunrpc               *.*              LISTEN
tcp        0      0  *.chargen              *.*              LISTEN
tcp        0      0  *.daytime              *.*              LISTEN
tcp        0      0  *.time                 *.*              LISTEN
tcp        0      0  *.domain               *.*              LISTEN
tcp        0      0  *.finger               *.*              LISTEN
tcp        0      0  *.exec                 *.*              LISTEN
tcp        0      0  *.ftp                  *.*              LISTEN
tcp        0      0  *.telnet               *.*              LISTEN
tcp        0      0  *.login                *.*              LISTEN
tcp        0      0  *.shell                *.*              LISTEN
tcp        0      0  scobox.listen          *.*              LISTEN
tcp        0      0  scobox.nterm           *.*              LISTEN
tcp        0      0  *.*                    *.*              CLOSED
udp        0      0  *.1035                 *.*
udp        0      0  *.1034                 *.*
udp        0      0  *.1033                 *.*
udp        0      0  *.1032                 *.*
udp        0      0  *.2049                 *.*
udp        0      0  *.1028                 *.*
udp        0      0  *.sunrpc               *.*
udp        0      0  scobox.domain          *.*
udp        0      0  localhost.domain       *.*
scobox$
```

Descriptions of the Display Headings

- The protocol used in the connection.
- Receive queue. The number of received characters (bytes) of data waiting to be processed.
- Send queue. The number of characters (bytes) of data waiting to be transmitted.
- The port number of the local connection, displayed symbolically. The port numbers are taken from the */etc/services* file.
- The port number of the remote connection, displayed symbolically. The port numbers are taken from the */etc/services* file.
- The current state of the connection. Each protocol has its own set of states. For the protocol-dependent states that can be displayed, see the appropriate protocol specification.

Interfaces

This display describes activities on all the local machine's interfaces to the net, in the form of a table of cumulative statistics. This display is available through **netstat** with the **-i** option.

netstat -i

```
scobox$ netstat -i
Name Mtu  Network     Address      Ipkts   Ierrs Opkts   Oerrs Collis
en0  1500 sco-eng-ne  scobox       No Statistics Available
e3B0 1500 128.174.14  128.174.14.1 0       0     0       0     0
lo0  2048 loopback    localhost    189     0     189     0     0
scobox$
```

Descriptions of the Display Headings

Each interface is described by a line with the following headings:

Name	The name of the network interface. For example, *en0* is the name of the first Ethernet interface board.
Mtu	Maximum transmission unit (in bytes). This is the largest size permitted for any single packet sent through this interface.
Network	The name of the network address of the interface as given in */etc/networks*.
Address	The name of the machine address of the interface in */etc/hosts*.
Ipkts	Input packets. The number of packets received on the interface.
Ierrs	Input errors. The number of errors detected in packets of data received on this interface.
Opkts	Output packets. The number of packets transmitted on the interface.
Oerrs	Output errors. The number of errors detected and corrected in packets of data transmitted on this interface.
Collis	Collisions that occurred on the network.

Routing Tables

The Routing Table display provides information about the usage of each route you have configured. A route consists of a destination host or network and a network interface used to exchange packets. Direct routes are created for each interface attached to the local host.

netstat -r

```
scobox$ netstat -r
Routing tables
Destination     Gateway          Flags    Refcnt Use     Interface
localhost       localhost        UH       4      0       lo0
sco-eng-net     scobox           U        4      537     en0
128.174.14      128.174.14.1     U        0      0       e3B0
128.174         scoffle          UG       0      0       en0
scobox$
```

Descriptions of the Display Headings

The information displayed for each route is as follows.

Destination — The network or machine to which this route allows you to connect.

Gateway — The name of the gateway you configured for this route. If you are directly connected, this is a local address. Otherwise, it is the name of the machine through which packets must be routed.

Flags — The state of the route. Valid states are:

U	up
G	a route to a gateway
N	a route to a network
H	a route to a host

Refcnt — The current number of active connections using the route. Connection-oriented protocols normally hold on to a single route for the duration of the connection, while connectionless protocols obtain a route and then discard it as needed.

Use — The current number of packets sent using this route.

Interface — The name of the physical network interface used to begin the route.

Statistics Display

The Protocol Statistics display provides protocol-specific errors. The errors in the display are grouped under headings for each higher-level protocol in your system. The headings are protocol-specific.

- Internet Protocol (ip)
- Internet Control Message Protocol (icmp)
- Transmission Control Protocol (tcp)
- User Datagram Protocol (udp)

netstat -s

```
ip:
        3209 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        0 fragments received
        0 fragments dropped (dup or out of space)
        0 fragments dropped after timeout
        0 packets forwarded
        0 packets not forwardable
        0 redirects sent
icmp:
        1 call to icmp_error
        0 errors not generated because old message was icmp
        Output histogram:
                destination unreachable: 1
```

(Continued on next page.)

(Continued)

```
        0 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                destination unreachable: 640
        0 message responses generated
tcp:
        348 packets sent
                202 data packets (3661 bytes)
                0 data packets (0 bytes) retransmitted
                101 ack-only packets (60 delayed)
                0 URG only packets
                0 window probe packets
                0 window update packets
                45 control packets
        411 packets received
                233 acks (for 3654 bytes)
                19 duplicate acks
                0 acks for unsent data
                200 packets (1677 bytes) received in-sequence
                0 completely duplicate packets (0 bytes)
                0 packets with some dup. data (0 bytes duped)
                9 out-of-order packets (0 bytes)
                0 packets (0 bytes) of data after window
                0 window probes
                0 window update packets
                0 packets received after close
                0 discarded for bad checksums
                0 discarded for bad header offset fields
                0 discarded because packet too short
        25 connection requests
        12 connection accepts
        21 connections established (including accepts)
        72 connections closed (including 0 drops)
        16 embryonic connections dropped
        233 segments updated rtt (of 259 attempts)
        0 retransmit timeouts
                0 connections dropped by rexmit timeout
```

(Continued on next page.)

(Continued)

```
        0 persist timeouts
        0 keepalive timeouts
                0 keepalive probes sent
                0 connections dropped by keepalive
        0 connections lingered
                0 linger timers expired
                0 linger timers cancelled
                0 linger timers aborted by signal
udp:
        0 incomplete headers
        0 bad data length fields
        0 bad checksums
```

Chapter 2

Introduction to sendmail

Introduction

The **sendmail** program acts as a central "post office" which routes internetwork mail. Such mail has more complex addresses than local mail or mail within a single network, because the various networks which compose an internetwork often have different address standards which must be reinterpreted if the mail is to be routed correctly. The **sendmail** program interprets and translates addresses to ensure that mail reaches the intended destination.

The **sendmail** program does not interface with the user. Neither does it perform the actual mail delivery. Rather, it collects a message generated by a user interface program such as UNIX **mail**(C), edits the message as required by the destination network, and calls appropriate mailers to carry out the mail delivery or queueing for network transmission. This allows the insertion of new mailers at minimum cost. The **sendmail** program is designed to interface with such mail delivery channels as UUCP and SMTP (Simple Mail Transfer Protocol).

Communicating with sendmail

There are several ways in which **sendmail** can communicate with the outside world, both in receiving and in sending mail:

- the conventional UNIX argument vector/return status (that is, a user interface program which invokes **sendmail**)
- SMTP (Simple Mail Transfer Protocol) over a pair of UNIX pipes
- SMTP over a Berkeley-style socket

These methods are discussed in the sections that follow.

User Interface Program

This technique is the standard UNIX method for communicating with the process. In this method, a user interface program invokes **sendmail**. A list of recipients is sent in the argument vector (that is, the list of arguments), and the message body is sent on the standard input. Anything that the mailer prints is simply collected and sent back to the sender if there were any problems. The Return or Exit status from the mailer is collected after the message is sent, and a diagnostic is printed if appropriate.

Here is an example which illustrates the concept of argument vectors:

```
main(argc, argv)
int argc;           /* Number of arguments */
char *argv[];       /* argument vector (list of arguments) */
{
  int i;

  for (i = 1; i < argc, i++)
    printf(%s%c, argv[i], (i<argc-1) ? ' ' : '\n');
  exit (0);
}
```

Since *argv[0]* is the name by which the program was invoked, *argc* will be at least 1.

SMTP over Pipes

The Simple Mail Transfer Protocol (SMTP) can be used to run an interactive lock-step interface with the mailer. A subprocess is still created, but no recipient addresses are passed to the mailer via the argument list. Instead, they are passed one at a time in commands sent to the process's standard input. Anything appearing on the standard output must be a reply code in a special format. The pipes are between the parent process and the child (sub)process's stdout and stdin. (The interactive lock-step interface is the interactive control between the parent process and the subprocess.) For more information on SMTP, see its definition in RFC821. There is also some related material in RFC822.

SMTP over a Berkeley-Style Socket

This technique is similar to the previous one, except that it uses a Berkeley-style socket. This method is exceptionally flexible, as it is not necessary for the mailer to reside on the same machine. This technique is normally used to connect to a **sendmail** process on a different machine.

Overview of sendmail Operation

When a sender wants to send a message, a request is issued to **sendmail** using one of the three methods described above. The **sendmail** program operates in two distinct phases. During the first phase, it collects and stores the message. During the second phase, the message is delivered. If errors occur during the second phase, **sendmail** creates and returns a new message describing the error. Alternatively, it may return a status code to indicate what went wrong.

Argument Processing and Address Parsing

If **sendmail** is called using SMTP over pipes or through a socket, the following sequence occurs. The arguments are first scanned and option specifications are processed. Recipient addresses are then collected, either from the command line or from the SMTP command RCPT (recipient), and a list of recipients is created. Aliases are expanded at this point. This includes any aliases that are part of a mailing list. At this stage, as much validation of the addresses as possible is done. Syntax is checked and local addresses are verified, but detailed checking of host names and addresses is deferred until delivery. Forwarding is also performed as the local addresses are verified.

The **sendmail** program appends each address to the recipient list after parsing the recipient list. When a name is aliased or forwarded, the old name is retained in the list, and a flag is set that tells the delivery phase to ignore this recipient. This list is kept free from duplicates, preventing alias loops and duplicate messages from being delivered to the same recipient, as might occur if a person is in two groups.

Collecting Messages

The **sendmail** program collects the message after the address parsing is complete. The message should have a header at the beginning. No formatting requirements are imposed on the message, except that they must be lines of text; binary data is not allowed. The header is parsed and stored in memory, and the body of the message is saved in a temporary file.

To simplify the program interface, the message is collected even if no addresses are valid. The message is then returned to the sender with an error.

Delivering Messages

For each unique mailer and host in the recipient list, **sendmail** calls the appropriate mailer. Each mailer invocation sends to all users receiving the message on one host. Mailers that accept only one recipient at a time are handled accordingly.

The message is sent to the mailer, using one of the same three interfaces used to submit a message to **sendmail**. Each copy of the message is prepended by a customized header. The mailer status code is caught and checked, with a suitable error message given as appropriate. The exit code must conform to a system standard; otherwise, a generic message such as "Service unavailable" is given.

Queueing for Retransmission

If the mailer returns a status code that indicates the possibility of being able to handle the mail later, **sendmail** puts the mail on the queue and tries again later.

Return to Sender

If errors occur during processing, **sendmail** returns the message to the sender for retransmission. If the user agent (mail) detects the error, then it will be put in the *dead.letter* file located in the sender's home directory. If a **sendmail** server is connecting with a **sendmail** client on another machine, then the user is presumed to have become detached from the transaction, and so the message is mailed back to them.

Editing the Message Header

A certain amount of editing occurs automatically to the message header. Header lines can be inserted under control of the configuration file. Some lines can be merged. For example, a "From:" line and a "Full-name:" line can be merged under certain circumstances.

The Configuration File

Almost all configuration information for **sendmail** is read at runtime from the ASCII file */usr/lib/sendmail.cf*. This file has macro definitions encoded

in it. These define such details as the value of macros used internally, header declarations, mailer definitions and address rewriting rules.

The header declarations are used to tell **sendmail** the format of header lines that will be processed specially. For example, any lines that are added or reformatted receive special processing. The mailer definitions give information such as the location and characteristics of each mailer. The address rewriting rules enable **sendmail** to be highly configurable and customizable, though this comes at the cost of some complexity. See the chapter "Installing and Operating sendmail" for more information on the **sendmail** configuration file.

Sendmail Implementation

The following sections contain information on the implementation of the **sendmail** program.

Sendmail and Arguments

Arguments to **sendmail** can be flags and addresses. The **sendmail** program is initially invoked through a command line in the file */etc/tcp*. Various flags can be set on this command line to control different processing options. For example, there are flags to run in ARPANET mode, to run as a daemon, to initialize the alias database, to use the SMTP protocol, and many other options. Control messages can also be sent to **sendmail** while it is operating.

Address arguments can be given following flag arguments, unless you are running in SMTP mode. These addresses follow the syntax in RFC822 for ARPANET address formats. Briefly, the format is as follows:

- Anything in parentheses is thrown away (as a comment).
- Arguments in angle brackets (<>) are preferred over anything else. This rule implements the ARPANET standard. This means that addresses of the following form will send to the electronic machine-address rather than the human user name.

```
user name <machine-address>
```

- Double quotes (") quote phrases; backslashes (\) quote characters. Backslashes are more powerful in that they will cause otherwise equivalent phrases to compare differently.

Parentheses, angle brackets, and double quotes must be properly balanced and nested.

Mailing to Files and Programs

Any address passing through the initial parsing algorithm as a local address (that is, not appearing to be a valid address for another mailer) is scanned for two special cases. If prefixed by a vertical bar (|), the rest of the address is processed as a shell command. If the user name begins with a slash mark (/), the name references the name of a file instead of a login name.

Files that have setuid or setgid bits set but no execute bits set have those bits honored if **sendmail** is being run by root. For example, if the file permissions are rwSrw-r-- or rw-rwSr--, then these file permission bits will be honored. However, if any execute bits are set, such as rwsr-xr--, then the read and write permissions will not be honored. (See **ls**(C) and **chmod**(C) for more information on file permissions.)

Aliasing, Forwarding and Including Mail

The **sendmail** program reroutes mail in any of the following three ways:

- by aliasing
- by forwarding
- by inclusion

Aliasing applies across the entire system. Forwarding allows all users to reroute incoming mail destined for their accounts. Inclusion directs **sendmail** to read a file for a list of addresses. Forwarding is normally used in conjunction with aliasing. Each of these methods is described in more detail in the next three sections.

Aliasing

Aliasing matches names to address lists using a system-wide file. This file is indexed to speed access. Only names that parse as local are allowed as aliases. This guarantees a unique key. The alias file is usually configured to be */usr/lib/aliases*. This file is not in the same format as the alias file */usr/lib/mail/aliases*. The identity of the alias file is configured through the *sendmail.cf* file. (See the chapter "Installing and Operating sendmail" for more information on alias files.)

Forwarding

After aliasing, recipients that are local and valid are checked for the existence of a *.forward* file in their home directory. If one exists, the message is not sent to that user, but rather to the list of users in that file. Often, this list will contain only one address, and the feature will be used for network mail forwarding.

Forwarding also permits a user to specify a private incoming mailer. For example, forwarding to:

```
"|/usr/local/newmail myname"
```

will use a different incoming mailer.

Including

The syntax for including a file is:

```
:include: pathname
```

An address of this form reads the file specified by *pathname* and sends to all users listed in that file.

The intention is not to support direct use of this feature, but rather to use this as a subset of aliasing. In the following example, the form of the alias used is a method of letting a project maintain a mailing list without interaction with the system administration, even if the alias file is protected.

```
project: :include:/usr/project/userlist
```

It is not necessary to rebuild the index on the alias database when a list of this type is changed. All that is needed is to edit the include file to reflect the changes. In this example, the include file is */usr/project/userlist*.

Collecting Messages

Once all recipient addresses are parsed and verified, the message is collected. The message comes in two parts. These parts are:

- the message header
- the message body

The two parts are separated by a blank line.

The header is formatted as a series of lines of the form:

field-name: field-value

Field-value can be split across lines by starting the lines that follow with a space or a tab. Some header fields have special internal meaning, in which case they are subject to special processing. Other headers are simply passed through. Some header fields, such as time stamps, may be added automatically.

The message body is a series of text lines. It is completely uninterpreted and untouched by **sendmail**, except that lines beginning with a dot (.) have the dot doubled when transmitted over an SMTP channel. This extra dot is stripped by the receiver. (SMTP uses lines beginning with a dot to signal the end of the message.)

Delivering Messages

The send queue is sequenced by the receiving host before transmission in order to implement message batching. Each address is marked as it is sent, and so rescanning the list is safe. An argument list is built as the scan proceeds. Mail to files is detected during the scan of the send list.

After a connection is established, **sendmail** makes the changes to the header necessary for correct interpretation by a particular mailer and sends the result to that mailer. If any mail is rejected by the mailer, a flag is set to invoke the return-to-sender function after all delivery completes.

Queued Messages

If the mailer returns a "temporary failure" exit status, the message is queued. A control file is used to describe the recipients to be sent to and various other parameters. This control file is formatted as a series of lines, each describing a sender, a recipient, the time of submission, or some other significant parameter of the message. The header of the message is stored in the control file, so that the associated data file in the queue is just the temporary file that was originally collected.

Configuration

Configuration is controlled primarily by the configuration file */usr/lib/sendmail.cf*, which is read at startup. The **sendmail** program should not need to be recompiled unless it is necessary to perform any of the following:

- Change operating systems (V6, V7/32V, 4BSD).
- Remove or insert the DBM (UNIX database) library.
- Change ARPANET reply codes.
- Add header fields requiring special processing.

Adding mailers and changing parsing (that is, rewriting) or routing information do not require recompilation of **sendmail**. Instead, these changes are made in the configuration file.

If the mail is being sent by a local user, and the file *.mailcf* exists in the sender's home directory, that file is read as a configuration file after the system configuration file. The primary use of this feature is to add header lines.

The configuration file encodes macro definitions, header definitions, mailer definitions, rewriting rules, and options. The following sections contain a brief description of some of the information contained in the **sendmail** configuration file. See the chapter "Installing and Operating sendmail" for more information on the configuration file.

Macros

Macros can be used in three ways. They can be used to transmit unstructured textual information into the mail system. An example of this is the name that **sendmail** uses to identify itself in error messages. Macros can be used to transmit information from **sendmail** to the configuration file in creating other fields (such as argument vectors to mailers). Examples are the name of the sender, and the host and user of the recipient. Other macros are unused internally. These macros can be used as shorthand in the configuration file.

Header Declarations

Header declarations inform **sendmail** of the format of known header lines. Knowledge of a few header lines is built into **sendmail**, such as the From: and Date: lines.

Most configured headers will automatically be inserted in the outgoing message if they don't exist in the incoming message. Certain headers are suppressed by some mailers.

Mailer Declarations

Mailer declarations tell **sendmail** of the various mailers available to it. The mailer declaration specifies the internal name of the mailer, the pathname of the program to call, some of the flags associated with the mailer, and an argument vector to be used in the call to the mailer.

Rules for Rewriting an Address

The heart of address parsing in **sendmail** is a set of rewriting rules. These are an ordered list of pattern-replacement rules, which are applied to each address. These rewriting rules are contained in the configuration file for **sendmail**. The configuration file also supports the editing of addresses into different formats. For example, an address of the form:

```
ucsfcgl!tef
```

might be mapped into:

```
tef@ucsfcgl.UUCP
```

to conform to the syntax of a different domain. Translations can also be done in the opposite direction.

Setting Options

There are several options that can be set from the configuration file. These include the pathnames of various support files, timeouts, default modes and so forth.

Comparing sendmail with Other Mail Programs

The remainder of this chapter compares **sendmail** with three other mail programs:

- **delivermail**
- MMDF (Multichannel Memorandum Distribution Facility)
- MPM (Message Processing Module)

These comparisons are provided for those who are familiar with these other mail routing programs.

Comparing sendmail with delivermail

The **sendmail** program is an outgrowth of **delivermail**. The primary differences are:

- Configuration information is not compiled in. This change simplifies many of the problems of moving to other machines. It also simplifies debugging of new mailers.

- Address parsing is more flexible. For example, **delivermail** only supported one gateway to any network, whereas **sendmail** can be sensitive to host names and reroute to different gateways.

- The forward and include features of **sendmail** eliminate the requirement that the system alias file be writable by any user (or that an update program be written, or that the system administration make all changes).

- The **sendmail** program supports message batching across networks when a message is being sent to multiple recipients.

- A mail queue is provided in **sendmail**. Mail that cannot be delivered immediately but can potentially be delivered later is stored in this queue for a later retry. The queue also provides a buffer against system crashes; after the message has been collected, it may be reliably redelivered even if the system crashes during the initial delivery.

- The **sendmail** program uses the networking support of 4.2BSD, which provides a direct interface to networks such as ARPANET or Ethernet using SMTP (the Simple Mail Transfer Protocol) over a TCP/IP connection.

Comparing sendmail with MMDF

The Multichannel Memorandum Distribution Facility (MMDF) spans a wider problem set than **sendmail**. For example, the domain of MMDF includes a "phone network" mailer, whereas **sendmail** calls on pre-existing mailers in most cases.

MMDF and **sendmail** both support aliasing, customized mailers, message batching, automatic forwarding to gateways, queueing, and retransmission. MMDF supports two-stage timeout, which **sendmail** does not support. (MMDF uses two-stage timeout when routing mail through machines to users. If a message can't be forwarded to a particular machine or to a particular user on a machine, a warning is sent back to the mail message sender. This is stage 1. At some future time (configurable by the administrator), the message is relayed again. If it fails, a failure message is returned to the sender, and MMDF makes no further attempts to resend the original message. This is stage 2.)

The configuration for MMDF is compiled into the code.

Since MMDF does not consider backwards compatibility as a design goal, the address parsing is simpler but much less flexible.

It is somewhat harder to integrate a new channel into MMDF. In particular, MMDF must know the location and format of host tables for all channels, and each channel must speak a special protocol. This allows MMDF to do additional verification (such as verifying host names) at submission time.

MMDF strictly separates the submission and delivery phases. Although **sendmail** has the concept of each of these stages, they are integrated into one program, whereas in MMDF they are split into two programs.

Sendmail and the Message-Processing Module

The Message Processing Module (MPM) matches **sendmail** closely in terms of its basic architecture. However, like MMDF, the MPM includes the network interface software as part of its domain.

MPM also postulates a duplex channel to the receiver, as does MMDF. This allows simpler handling of errors by the mailer than is possible in **sendmail**. When a message queued by **sendmail** is sent, any errors must be returned to the sender by the mailer itself. Both MPM and MMDF mailers can return an immediate error response, and a single error processor can create an appropriate response.

MPM prefers passing the message as a structured object, with {type, length, value} triples. Such a convention requires a much higher degree of cooperation between mailers than is required by **sendmail**. MPM also assumes a universally agreed-upon internet name space (with each address in the form of a net-host-user tuple), which **sendmail** does not.

Chapter 3

Installing and Operating sendmail

Introduction

The **sendmail** program implements a general purpose internetwork mail-routing facility under the UNIX operating system. It is not tied to any one transport protocol. Its function may be likened to a crossbar switch, relaying messages from one domain into another. Included as part of this process, it can do a limited amount of message-header editing to put the message into a format that is appropriate for the receiving domain. All of this is done under the control of a configuration file.

Due to the requirements of flexibility for **sendmail**, the configuration file can seem somewhat unapproachable. However, there are only a few basic configurations for most sites, for which standard configuration files have been supplied. Most other configurations can be built by adjusting existing configuration files incrementally.

Although **sendmail** is intended to run without the need for monitoring, it has a number of features that may be used to monitor or adjust the operation under unusual circumstances.

Basic Installation

There are two basic steps to installing **sendmail.** They are:

- building the configuration table
- installing the software

The configuration table is a file that **sendmail** reads when it starts up. This file describes the mailers that sendmail knows about, how to parse addresses, how to rewrite the message header, and the settings of various options. Although the configuration table is quite complex, a configuration can usually be built by adjusting an existing off-the-shelf configuration. The second step is performing the actual installation. This means creating the necessary files and so on.

The remainder of this section describes the installation of **sendmail**. The description assumes that you can use one of the existing configurations and that the standard installation parameters are acceptable.

Off-the-Shelf Configurations

Several sample configuration files are included with this release. They are found in the directory */usr/local/lib/sendmail*. The makefile in the **cf** directory makes two files, **node.cf** and **relay.cf**. **Node.cf** is the configuration to use on a host that is not a central mail router. **Relay.cf** should be used on a major mail relay machine in your installation.

Once these variables are changed, the file is now ready for installation as **/usr/lib/sendmail.cf**.

The configuration file you need should be copied to a file with the same name as your system, as in the following example:

```
cp relay.cf laidbak.cf
```

There are some variables that need to be changed in both files, and the **Tune** script in the **cf** directory can be used to take care of this. These variables are as follows:

Tunable Parameters	
Parameter	Value
ZZHOST	Host Name
ZZRELAY	Mail Relay Host Name
ZZDOMAIN	Your subdomain, *e.g.* Lachman.
ZZDOM	Your domain, *e.g.* COM.

You can use the **Tune** script to change these variables. **Tune** takes the qualified hostname and the relay as arguments. It splits the hostname up to generate ZZHOST, ZZDOMAIN, and ZZDOM.

This file is now ready for installation as */usr/lib/sendmail.cf*.

Installation

For details about how to install the sendmail software, see the *TCP/IP Runtime Release and Installation Notes*.

Quick Configuration Startup

A fast version of the configuration file can be set up by using the **-bz** flag, as in the following example:

```
/usr/lib/sendmail -bz
```

This creates the file */usr/lib/sendmail.fc* (frozen configuration). This file is an image of **sendmail**'s data space after reading in the configuration file. If this file exists, it is used instead of */usr/lib/sendmail.cf*. The *sendmail.fc* file must be rebuilt manually every time *sendmail.cf* is changed.

The frozen configuration file will be ignored if a **-C** flag is specified or if **sendmail** detects that it is out of date. However, the heuristics are not strong, and so this should not be trusted.

The System Log

The system log is entered in the file */usr/adm/syslog*.

Format

Each line in the system log consists of a timestamp, the name of the machine that generated it (for logging from several machines over the ethernet), the word "sendmail," and a message.

Levels

A large amount of information can be logged. The log is arranged as a succession of levels. At the lowest level, only extremely strange situations are logged. At the highest level, even the most mundane and ininteresting events are recorded for posterity. As a convention, log levels under ten are consiedered "useful;" log levels above ten are usually for debugging purposes.

The Mail Queue

The mail queue should be processed transparently. However, you may find that manual intervention is sometimes necessary. For example, if a major host is down for a period of time the queue may become clogged. Although **sendmail** ought to recover gracefully when the host comes up, you may find performance unacceptably bad in the meantime.

Printing the Queue

The contents of the queue can be printed using the **mailq** command (or by specifying the **-bp** flag to **sendmail**):

mailq

This produces a listing of the queue identifiers, the size of the message, the date the message entered the queue, and the sender and recipients.

Format of Queue Files

All queue files have the form *x*f*AA99999*, where *AA99999* is the ID for this file and *x* is a type. The types are:

d The data file. The message body (excluding the header) is kept in this file.

l The lock file. If this file exists, the job is currently being processed, and a queue run will not process the file. For that reason, an extraneous lf file can cause a job to seem to disappear. (It will not even time out!)

n This file is created when an ID is being created. It is a separate file to ensure that no mail can ever be destroyed due to a race condition. It should exist for no more than a few milliseconds at any given time.

q The queue control file. This file contains the information necessary to process the job.

t A temporary file. This is an image of the qf file when it is being rebuilt. It should be renamed to a qf file very quickly.

x A transcript file, existing during the life of a session, showing everything that happens during that session.

The qf file is structured as a series of lines, each beginning with a code letter. The lines are as follows:

D The name of the data file. There can only be one of these lines.

H A header definition. There can be any number of these lines. The order is important: it represents the order in the final message. This uses the same syntax as header definitions in the configuration file.

R A recipient address. This will normally be completely aliased, but is actually realiased when the job is processed. There will be one line for each recipient.

S The sender address. There can only be one of these lines.

E An error address. If any such lines exist, they represent the addresses that should receive error messages.

T The job creation time. This is used to compute how long a job remains in the queue undelivered, before being returned to the sender.

P The current message priority. This is used to order the queue. Higher numbers mean lower priorities. The priority changes as the message sits in the queue. The initial priority depends on the message class and the size of the message.

M A message. This line is printed by the **mailq** command, and is generally used to store status information. It can contain any text.

As an example, the following is a queue file sent to "mckee@calder" and "wnj:"

```
DdfA13557
Seric
T404261372
P132
Rmckee@calder
Rwnj
H?D?date: 23-Oct-82 15:49:32-PDT (Sat)
H?F?from: eric (Eric Allman)
H?x?full-name: Eric Allman
Hsubject: this is an example message
Hmessage-id: <8209232249.13557@UCBARPA.BERKELEY.ARPA>
Hreceived: by UCBARPA.BERKELEY.ARPA (3.227 [10/22/82])
       id A13557; 23-Oct-82 15:49:32-PDT (Sat)
HTo: mckee@calder, wnj
```

This shows the name of the data file, the person who sent the message, the submission time (in seconds since January 1, 1970), the message priority, the message class, the recipients, and the headers for the message.

Forcing the Queue

The **sendmail** program should run the queue automatically at intervals. The algorithm is to read and sort the queue, and then to attempt to process all jobs in order. When it attempts to run the job, **sendmail** first checks to see if the job is locked. If so, it ignores the job.

There is no attempt to ensure that only one queue processor exists at any time, since there is no guarantee that a job cannot take forever to process. Due to the locking algorithm, it is impossible for one job to freeze the queue. However, an uncooperative recipient host or a program recipient that never returns can accumulate many processes in your system. Unfortunately, there is no way to resolve this without violating the protocol.

In some cases, you may find that if a major host goes down for a couple of days, this can create a prohibitively large queue. This situation will cause **sendmail** to spend an inordinate amount of time sorting the queue. This situation can be fixed by moving the queue to a temporary place and creating a new queue. The old queue can be run later, when the offending host returns to service.

To do this, it is acceptable to move the entire queue directory:

```
cd /usr/spool
mv mqueue omqueue; mkdir mqueue; chmod 777 mqueue
```

You should then kill the existing daemon (since it will still be processing in the old queue directory) and create a new daemon.

To run the old mail queue, run the following command:

```
/usr/lib/sendmail -oQ/usr/spool/omqueue -q
```

The **-oQ** flag specifies an alternate queue directory, and the **-q** flag says just to run every job in the queue. If you have a tendency toward voyeurism, you can use the **-v** flag to watch what is going on.

When the queue is finally emptied, you can remove the directory:

```
rmdir /usr/spool/omqueue
```

The Alias Database

The alias database exists in two forms. One is a text form, maintained in the file */usr/lib/aliases*. The aliases are of the form

name: *name1, name2, ...*

Only local names can be aliased. For example:

```
eric@mit-xx: eric@berkeley.EDU
```

will not have the desired effect. Aliases can be continued by starting any continuation line with a space or a tab. Blank lines and lines beginning with a pound sign (#) are comments.

The second form is processed by the **dbm**(S) library. This form is in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag*. This is the form that **sendmail** actually uses to resolve aliases. This technique is used to improve performance.

Rebuilding the Alias Database

The DBM version of the database can be rebuilt explicitly by executing the command:

newaliases

This is equivalent to giving **sendmail** the **-bi** flag:

/usr/lib/sendmail -bi

If the "D" option is specified in the configuration, **sendmail** will rebuild the alias database automatically, if possible, when it is out of date. It will do this under either or both of the following conditions:

- The DBM version of the database is mode 666.
- **sendmail** is running setuid to root.

Auto-rebuild can be dangerous on heavily loaded machines with large alias files; if it might take more than five minutes to rebuild the database, there is a chance that several processes will start the rebuild process simultaneously.

Potential Problems

There are a number of problems that can occur with the alias database. They all result when a **sendmail** process accesses the DBM version while it is only partially built. This can happen under two circumstances: either one process accesses the database while another process is rebuilding it, or the process rebuilding the database dies (due to being killed or a system crash) before completing the rebuild.

The **sendmail** program includes two techniques to try to relieve these problems. First, it ignores interrupts while rebuilding the database; this avoids the problem of someone aborting the process and leaving a partially rebuilt database. Second, at the end of the rebuild it adds an alias of the form:

```
@: @
```

(Note that this is not normally legal.) Before **sendmail** will access the database, it checks to ensure that this entry exists. The **sendmail** program will wait for this entry to appear, at which point it will force a rebuild itself.

List Owners

If an error occurs on sending to a certain address, say "*x*," **sendmail** will look for an alias of the form "owner-*x*" to receive the errors. This is typically useful for a mailing list where the submitter of the list has no control over the maintenance of the list itself; in this case the list maintainer would be the owner of the list. For example:

```
unix-wizards: eric@ucbarpa, wnj@monet, nosuchuser,
      sam@matisse
owner-unix-wizards: eric@ucbarpa
```

This would cause "eric@ucbarpa" to get the error that will occur when someone sends to unix-wizards, due to the inclusion of "nosuchuser" on the list.

Per-User Forwarding (.forward Files)

As an alternative to the alias database, any user can put a file with the name *.forward* in his or her home directory. If this file exists, **sendmail** redirects mail for that user to the list of addresses listed in the *.forward* file. For example, if the home directory for user mckee has a *.forward* file with contents:

```
mckee@ernie
kirk@calder
```

then any mail arriving for "mckee" will be redirected to the specified accounts.

Special Header Lines

Several header lines have special interpretations defined by the configuration file. Others have interpretations built into **sendmail** that cannot be changed without changing the code. These built-ins are described here.

Return-Receipt-to:

If this header is sent, a message will be sent to any specified addresses when the final delivery is completed, that is, when successfully delivered to a mailer with the **l** flag (local delivery) set in the mailer descriptor.

Errors-To:

If errors occur anywhere during processing, this header will cause error messages to go to the listed addresses rather than to the sender. This is intended for mailing lists.

Apparently-To:

If a message comes in with no recipients listed in the message (in a To:, Cc:, or Bcc: line), then **sendmail** will add an "Apparently-To:" header line for any recipients it is aware of. This is not put in as a standard recipient line to warn any recipients that the list is not complete.

Arguments

Some important arguments of the **sendmail** program are described here.

Queue Interval

The amount of time between forking a process to run through the queue is defined by the **-q** flag. If you run in mode **f** or **a,** this can be relatively large, since it will only be relevant when a host that was down comes back up. If you run in **q** mode, it should be relatively short, since it defines the maximum amount of time that a message may sit in the queue.

Daemon Mode

If you allow incoming mail over an IPC connection, you should have a daemon running. This should be set by your */etc/rc* file using the **-bd** flag. The **-bd** flag and the **-q** flag may be combined in one call:

```
/usr/lib/sendmail -bd -q30m
```

Forcing the Queue

In some cases, you may find that the queue has gotten clogged for some reason. You can force a queue run using the **-q** flag (with no value). It is entertaining to use the **-v** flag (verbose) when this is done, to watch what happens:

```
/usr/lib/sendmail -q -v
```

Debugging

There is a fairly large number of debug flags built into **sendmail**. Each debug flag has a number and a level, where higher levels cause more information to be printed out. The convention is that levels greater than nine are not required. They print out so much information that you would not normally want to see them, except for debugging that particular piece of code. Debug flags are set using the **-d** option. The syntax is:

```
debug-flag:     -d debug-list
debug-list:     debug-option [ , debug-option ]
debug-option:   debug-range [ . debug-level ]
debug-range:    integer | integer - integer
debug-level:    integer
```

where spaces are for reading ease only. For example,

```
-d12        Set flag 12 to level 1
-d12.3      Set flag 12 to level 3
-d3-17      Set flags 3 through 17 to level 1
-d3-17.4    Set flags 3 through 17 to level 4
```

For a complete list of the available debug flags you will have to look at the code. (They are too dynamic to keep this documentation up to date.)

Trying a Different Configuration File

An alternative configuration file can be specified using the **-C** flag. The following example uses the configuration file *test.cf* instead of the default */usr/lib/sendmail.cf*.

```
/usr/lib/sendmail -Ctest.cf
```

If the **-C** flag has no value, it defaults to *sendmail.cf* in the current directory.

Changing the Values of Options

Options can be overridden using the **-o** flag. For example:

```
/usr/lib/sendmail -oT2m
```

sets the **T** (timeout) option to two minutes for this run only.

Tuning

There are a number of configuration parameters you may want to change, depending on the requirements of your site. Most of these are set using an option in the configuration file. For example, the line "OT3d" sets option "T" to the value "3d" (three days).

Most of these options default appropriately for most sites. However, sites having very high mail loads may find they need to tune them as appropriate for their mail load. In particular, sites experiencing a large number of small messages, many of which are delivered to many recipients, may find that they need to adjust the parameters dealing with queue priorities.

Timeouts

All time intervals are set using a scaled syntax. For example, "10m" represents ten minutes, whereas "2h30m" represents two-and-a-half hours. The full set of scales is:

s	seconds
m	minutes
h	hours
d	days
w	weeks

The argument to the **-q** flag specifies how often a subdaemon will run the queue. This is typically set to between fifteen minutes and one hour.

Read Timeouts

It is possible to time out when reading the standard input or when reading from a remote SMTP server. Technically, this is not acceptable within the published protocols. However, it might be appropriate to set it to something large (such as an hour) in certain environments. This will reduce the chance of large numbers of idle daemons piling up on your system. This timeout is set using the **r** option in the configuration file.

Message Timeouts

After sitting in the queue for a few days, a message will time out. This means that if a message cannot be delivered for some reason, it is returned to the sender. This ensures that at least the sender is aware that the message was not sent. The timeout is typically set to three days. This timeout is set using the **T** option in the configuration file.

The time of submission is set in the queue, rather than the amount of time left until timeout. As a result, you can flush messages that have been hanging for a short period by running the queue with a short message timeout. For example:

```
/usr/lib/sendmail -oT1d -q
```

will run the queue and flush anything that is one day old.

Forking During Queue Runs

When the **Y** option is set, **sendmail** will fork before each individual message while running the queue. This prevents **sendmail** from consuming large amounts of memory, and so it may be useful in memory-poor environments. However, if the **Y** option is not set, **sendmail** will keep track of hosts that are down during a queue run, which can improve performance dramatically.

Queue Priorities

Every message is assigned a priority when it is first instantiated, consisting of the message size (in bytes) offset by the message class multiplied by the "work class factor" and the number of recipients multiplied by the "work recipient factor." The priority plus the creation time of the message (in seconds since January 1, 1970) are used to order the queue. Higher numbers for the priority mean that the message will be processed later, when running the queue.

The message size is included so that large messages are penalized relative to small messages. The message class allows users to send high-priority messages by including a "Precedence:" field in their message; the value of this field is looked up in the **P** lines of the configuration file. Since the number of recipients affects the size of load a message presents to the system, this is also included into the priority.

The recipient and class factors can be set in the configuration file by using the **y** and **z** options respectively. They default to 1000 (for the recipient factor) and 1800 (for the class factor). The initial priority is:

```
pri = size - (class * z) + (nrcpt * y)
```

(Remember, higher values for this parameter actually mean that the job will be treated with lower priority.)

The priority of a job can also be adjusted each time it is processed (that is, each time an attempt is made to deliver it) using the "work time factor," set by the **Z** option. This is added to the priority, so it normally decreases the precedence of the job, on the grounds that jobs that have failed many times will tend to fail again in the future.

Delivery Mode

There are a number of delivery modes for **sendmail**, and they are set by the **d** configuration option. These modes specify how quickly mail will be delivered. Legal modes are:

- i deliver interactively (synchronously)
- b deliver in background (asynchronously)
- q queue only (do not deliver)

There are tradeoffs. Mode "i" passes the maximum amount of information to the sender, but is hardly ever necessary. Mode "q" puts the minimum load on your machine, but means that delivery may be delayed for up to the queue interval. Mode "b" is probably a good compromise. However, this mode can cause large numbers of processes if you have a mailer that takes a long time to deliver a message.

File Modes

There are several files involved with **sendmail** that can have a number of modes. The modes depend on the functionality you want and the level of security you require.

To suid or not to suid?

The **sendmail** program can safely be made setuid to root. At the point where it is about to **exec**(S) a mailer, it checks to see if the userid is zero. If so, it resets the userid and groupid to a default (set by the **u** and **g** options). (This can be overridden by setting the **S** flag to the mailer for mailers that are trusted and must be called as root.) However, this will cause mail processing to be accounted to root rather than to the user sending the mail.

Temporary File Modes

The mode of all temporary files that **sendmail** creates is determined by the **F** option. Reasonable values for this option are 0600 and 0644. If the more permissive mode is selected, it will not be necessary to run **sendmail** as root at all (even when running the queue).

Should My Alias Database Be Writable?

The database that **sendmail** actually uses is represented by two files. These files are:

- *aliases.dir*
- *aliases.pag*

Both files are located in */usr/lib*. The mode on these files should match the mode on */usr/lib/aliases*. If *aliases* is writable and the DBM files (*aliases.dir* and *aliases.pag*) are not, users will be unable to reflect their desired changes through to the actual database. However, if *aliases* is read-only and the DBM files are writable, a slightly sophisticated user can arrange to steal mail anyway.

If your DBM files are not writable by the world or you do not have autorebuild enabled (with the **D** option), then you must be careful to reconstruct the alias database each time you change the text version:

```
newaliases
```

If this step is ignored or forgotten, any intended changes will also be ignored or forgotten.

The Configuration File

This section describes the configuration file in detail, including hints on how to write one of your own if you have to.

There is one point that should be made clear immediately: the syntax of the configuration file is designed to be reasonably easy to parse, since this is done every time **sendmail** starts up. As a result, the configuration file is not particularly easy for a human to read or write.

An overview of the configuration file is given first, followed by details of the semantics.

The Syntax

The configuration file is organized as a series of lines, each of which begins with a single character defining the semantics for the rest of the line. Lines beginning with a space or a tab are continuation lines (although the semantics are not well defined in many places). Blank lines and lines beginning with a pound symbol (#) are comments.

R and S - Rewriting Rules

The core of address parsing is the rewriting rules. These are an ordered production system. The **sendmail** command scans through the set of rewriting rules looking for a match on the left hand side (LHS) of the rule. When a rule matches, the address is replaced by the right hand side (RHS) of the rule.

There are several sets of rewriting rules. Some of the rewriting sets are used internally and must have specific semantics. Other rewriting sets do not have specifically assigned semantics, and may be referenced by the mailer definitions or by other rewriting sets.

The syntax of these two commands are:

S*n*

Sets the current ruleset being collected to *n*. If you begin a ruleset more than once, it deletes the old definition.

R*lhs rhs comments*

The fields must be separated by at least one tab character; there may be embedded spaces in the fields. The *lhs* is a pattern that is applied to the input. If it matches, the input is rewritten to the *rhs*. The *comments* are ignored.

D - Define Macro

Macros are named with a single character. These may be selected from the entire ASCII set, but user-defined macros should be selected from the set of uppercase letters only. Lowercase letters and special symbols are used internally.

The syntax for macro definitions is:

D*x val*

where *x* is the name of the macro and *val* is the value it should have. Macros can be interpolated in most places using the escape sequence **$***x*.

C and F - Define Classes

Classes of words may be defined to match on the left-hand side of rewriting rules. For example, a class of all local names for this site might be created so that attempts to send to oneself can be eliminated. These can either be defined directly in the configuration file or read in from another file. Classes may be given names from the set of uppercase letters. Lowercase letters and special characters are reserved for system use.

The syntax is:

C*c word1 word2...*
F*c file*

The first form defines the class *c* to match any of the named words. It is permissible to split them among multiple lines; for example, the two forms:

CHmonet ucbmonet

and:

CHmonet
CHucbmonet

are equivalent. The second form reads the elements of the class *c* from the named *file*.

M - Define Mailer

Programs and interfaces to mailers are defined in this line. The format is:

M*name*, {*field*=*value* }*

where *name* is the name of the mailer (used internally only) and the "field=name" pairs define attributes of the mailer. Fields are:

Path	The pathname of the mailer
Flags	Special flags for this mailer
Sender	A rewriting set for sender addresses
Recipient	A rewriting set for recipient addresses
Argv	An argument vector to pass to this mailer
Eol	The end-of-line string for this mailer
Maxsize	The maximum message length for this mailer

Only the first character of the field name is checked.

H - Define Header

The format of the header lines that **sendmail** inserts into the message is defined by the **H** line. The syntax of this line is:

H[?*mflags*?]*hname*: *htemplate*

Continuation lines in this spec are reflected directly into the outgoing message. The *htemplate* is macro-expanded before insertion into the message. If the *mflags* (surrounded by question marks) are specified, at least one of the specified flags must be stated in the mailer definition for this header to be automatically output. If one of these headers is in the input, it is reflected to the output, regardless of these flags.

Some headers have special semantics, described below.

O - Set Option

There are a number of "random" options that can be set from a configuration file. Options are represented by single characters. The syntax of this line is:

O*o value*

This sets option *o* to be *value*. Depending on the option, *value* may be a string, an integer, a boolean (with legal values "t", "T", "f", or "F"; the default is TRUE), or a time interval.

T - Define Trusted Users

Trusted users are those who are permitted to override the sender address using the **-f** flag. These typically are "root," "uucp," and "network," but in some cases it may be convenient to extend this list to include other users, perhaps to support a separate UUCP login for each host. The syntax of this line is:

T*user1 user2...*

There may be more than one of these lines.

P - Precedence Definitions

Values for the "Precedence:" field may be defined using the **P** control line. The syntax of this field is:

P*name=num*

When the *name* is found in a "Precedence:" field, the message class is set to *num*. Higher numbers mean higher precedence. Numbers below zero have the special property that error messages will not be returned. The default precedence is zero. For example, our list of precedences is:

```
Pfirst-class=0
Pspecial-delivery=100
Pjunk=-100
```

The Semantics

This section describes the semantics of the configuration file.

Special Macros, Conditionals

Macros are interpolated using the construct **$***x*, where *x* is the name of the macro to be interpolated. In particular: lowercase letters are reserved to

have special semantics, used to pass information in or out of **sendmail**; some special characters are reserved to provide conditionals; and so on.

Conditionals can be specified using the syntax:

$?x *text1* **$|** *text2* **$.**

This interpolates *text1* if the macro **$x** is set, and *text2* otherwise. The "else" ($|) clause may be omitted.

The following macros must be defined to transmit information into **sendmail**:

e	The SMTP entry message
j	The official domain name for this site
l	The format of the UNIX from line
n	The name of the daemon (for error messages)
o	The set of "operators" in addresses
q	default format of sender address

The **$e** macro is printed out when SMTP starts up. The first word must be the **$j** macro. The **$j** macro should be in RFC821 format. The **$l** and **$n** macros can be considered constants, except under terribly unusual circumstances. The **$o** macro consists of a list of characters that will be considered tokens and that will separate tokens when parsing. For example, if "@" were in the **$o** macro, then the input "a@b" would be scanned as three tokens: "a", "@", and "b". Finally, the **$q** macro specifies how an address should appear in a message when it is defaulted. For example, on our system, these definitions are:

```
De$j Sendmail $v ready at $b
DnMAILER-DAEMON
DlFrom $g  $d
Do.:%@!^=/
Dq$g$?x ($x)$.
Dj$H.$D
```

An acceptable alternative for the **$q** macro is "$?x$x $.<$g>". These correspond to the following two formats:

```
eric@Lachman (Eric Allman)
Eric Allman <eric@Lachman>
```

Some macros are defined by **sendmail** for interpolation into argv's for mailers or for other contexts. These macros are:

a	The origination date in Arpanet format
b	The current date in Arpanet format
c	The hop count
d	The date in UNIX (ctime) format
f	The sender (from) address
g	The sender address relative to the recipient
h	The recipient host
i	The queue id
p	**sendmail**'s pid
r	Protocol used
s	Sender's host name
t	A numeric representation of the current time
u	The recipient user
v	The version number of **sendmail**
w	The hostname of this site
x	The full name of the sender
z	The home directory of the recipient

There are three types of dates that can be used. The **$a** and **$b** macros are in Arpanet format; **$a** is the time as extracted from the "Date:" line of the message (if there was one), and **$b** is the current date and time (used for postmarks). If no "Date:" line is found in the incoming message, **$a** is set to the current time also. The **$d** macro is equivalent to the **$a** macro in UNIX (ctime) format.

The **$f** macro is the id of the sender as originally determined; when you are mailing to a specific host, the **$g** macro is set to the address of the sender _relative to the recipient. For example, if I send to "bollard@matisse" from the machine "ucbarpa," the **$f** macro will be "eric" and the $g macro will be "eric@ucbarpa."

The **$x** macro is set to the full name of the sender. This can be determined in several ways. It can be passed as a flag to **sendmail.** The second choice is the value of the "Full-name:" line in the header if it exists, and the third choice is the comment field of a "From:" line. If all of these fail, and if the message is being originated locally, the full name is looked up in the */etc/passwd* file.

When sending, the **$h**, **$u**, and **$z** macros are set to the host, user, and home directories (if local) of the recipient. The first two are set from the **$@** and **$:** parts of the rewriting rules, respectively.

The **$p** and **$t** macros are used to create unique strings (for example, for the "Message-Id:" field). The **$i** macro is set to the queue id on this host;

if put into the timestamp line, it can be extremely useful for tracking messages. The **$v** macro is set to be the version number of **sendmail**; this is normally put in timestamps and has proved extremely useful for debugging. The **$w** macro is set to the name of this host, if it can be determined. The **$c** field is set to the "hop count," that is, the number of times this message has been processed. This can be determined by the **-h** flag on the command line or by counting the timestamps in the message.

The **$r** and **$s** fields are set to the protocol used to communicate with **sendmail** and the sending *hostname*; these are not supported in the current version.

Special classes

The class **$=w** is set to be the set of all names by which this host is known. This can be used to delete local hostnames.

The Left-Hand Side

The left-hand side of rewriting rules contains a pattern. Normal words are simply matched directly. Metasyntax is introduced using a dollar sign. The metasymbols are:

- **$*** Match zero or more tokens
- **$+** Match one or more tokens
- **$-** Match exactly one token
- **$=***x* Match any token in class *x*
- **$~***x* Match any token not in class *x*

If any of these match, they are assigned to the symbol **$***n* for replacement on the right-hand side, where *n* is the index in the LHS. For example, if the LHS:

```
$-:$+
```

is applied to the input:

```
UCBARPA:eric
```

then the rule will match, and the values passed to the RHS will be:

- $1 UCBARPA
- $2 eric

The Right-Hand Side

When the left-hand side of a rewriting rule matches, the input is deleted and replaced by the right-hand side. Tokens are copied directly from the RHS, unless they begin with a dollar sign. Metasymbols are:

$*n*	Substitute indefinite token *n* from LHS
$[***name***$]	Canonicalize *name*
$>*n*	Call ruleset *n*
$#*mailer*	Resolve to *mailer*
$@*host*	Specify *host*
$:*user*	Specify *user*

The **$***n* syntax substitutes the corresponding value from a **$+**, **$-**, **$***, **$=**, or **$~** match on the LHS. It can be used anywhere.

A host name enclosed between **$[** and **$]** is looked up using the **gethostent**(3) routines and replaced by the canonical name. For example, "$[csam$]" would become "lbl-csam.arpa", and "$[[128.32.130.2]$]" would become "vangogh.berkeley.edu."

The **$>***n* syntax causes the remainder of the line to be substituted as usual and then passed as the argument to ruleset *n*. The final value of ruleset *n* then becomes the substitution for this rule.

The **$#** syntax should only be used in ruleset zero. It causes evaluation of the ruleset to terminate immediately, and it signals to **sendmail** that the address has completely resolved. The complete syntax is:

$#***mailer***$@*host***$:***user*

This specifies the {mailer, host, user} 3-tuple necessary to direct the mailer. If the mailer is local, the host part can be omitted. The *mailer* and *host* must be a single word, but the *user* can be multi-part.

A RHS can also be preceded by a **$@** or a **$:** to control evaluation. A **$@** prefix causes the ruleset to return with the remainder of the RHS as the value. A **$:** prefix causes the rule to terminate immediately, but the ruleset to continue. This can be used to avoid continued application of a rule. The prefix is stripped before continuing.

The **$@** and **$:** prefixes can precede a **$>** spec. For example,

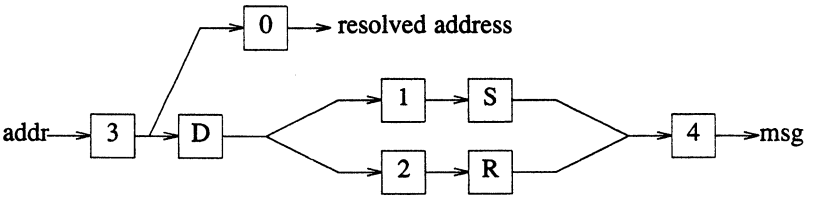
```
R$+     $:$>7$1
```

matches anything, passes that to ruleset seven, and continues; the **$:** is necessary to avoid an infinite loop.

Substitution occurs in the order described; that is, parameters from the LHS are substituted, hostnames are canonicalized, "subroutines" are called and, finally, **$#**, **$@**, and **$:** are processed.

Semantics of Rewriting Rule Sets

There are five rewriting sets that have specific semantics. These are related as depicted by Figure 4-1.



D - sender domain addition
S - mailer-specific sender rewriting
R - mailer-specific recipient rewriting

Figure 3-1 Rewriting Set Semantics

Ruleset three should turn the address into "canonical form." This form should have the basic syntax:

local-part@host-domain-spec

If no "@" sign is specified, then the host-domain-spec can be appended from the sender address (if the **C** flag is set in the mailer definition corresponding to the sending mailer). Ruleset three is applied by **sendmail** before doing anything with any address.

Ruleset zero is applied after ruleset three to addresses that are actually going to specify recipients. It must resolve to a {mailer, host, user} triple. The mailer must be defined in the mailer definitions from the configuration file. The host is defined into the **$h** macro for use in the argv expansion of the specified mailer.

Rulesets one and two are applied to all sender and recipient addresses, respectively. They are applied before any specification in the mailer definition. They must never resolve.

Ruleset four is applied to all addresses in the message. It is typically used to translate internal to external form.

Mailer Flags

There are several flags that can be associated with each mailer, each identified by a letter of the alphabet. Many of them are assigned semantics internally. Any other flags may be used freely to assign headers conditionally to messages destined for particular mailers.

The "error" Mailer

The mailer with the special name "error" can be used to generate a user error. The (optional) host field is a numeric exit status to be returned, and the user field is a message to be printed. For example, the entry:

```
$#error$:Host unknown in this domain
```

on the RHS of a rule will cause the specified error to be generated if the LHS matches. This mailer is only functional in ruleset zero.

Building a Configuration File from Scratch

Building a configuration file from scratch is an extremely difficult job. Fortunately, it is almost never necessary to do so; nearly every situation that may come up may be resolved by changing an existing table. In any case, it is critical that you understand what you are trying to do and come up with a philosophy for the configuration table. This section is intended to explain the real purpose of a configuration table and to give you some ideas as to what your philosophy might be.

What You are Trying to Do

The configuration table has three major purposes. The first and simplest is to set up the environment for **sendmail**. This involves setting the options, defining a few critical macros, and so on. Since these are described in other sections, we will not go into more detail here.

The second purpose is to rewrite addresses in the message. This should typically be done in two phases. The first phase maps addresses in any format into a canonical form. This should be done in ruleset three. The second phase maps this canonical form into the syntax appropriate for the receiving mailer.

The **sendmail** program performs this second phase in the following three subphases: Rulesets one and two are applied to all sender and recipient addresses, respectively. After this, you can specify per-mailer rulesets for

both sender and recipient addresses. This allows mailer-specific customization. Finally, ruleset four is applied to do any default conversion to external form.

The third purpose of the configuration table is to map addresses into the actual set of instructions necessary to get the message delivered. Ruleset zero must resolve to the internal form, which is in turn used as a pointer to a mailer descriptor. This describes the interface requirements of the mailer.

Relevant Issues

The canonical form you use should almost certainly be as specified in the Arpanet standards documents RFC819 and RFC822.
RFC822 describes the format of the mail message itself. The **sendmail** program follows this RFC closely, to the extent that many of the standards described in this document can not be changed without changing the code. In particular, the following characters have special interpretations:

```
< > ( ) " \
```

Any attempt to use these characters for other than their RFC822 purpose in addresses is probably doomed to disaster.

RFC819 describes the specifics of the domain-based addressing. This is touched on in RFC822 as well. Essentially, each host is given a name that is a right-to-left dot-qualified pseudo-path from a distinguished root. The elements of the path need not be physical hosts; the domain is logical rather than physical. For example, at Lachman, one legal host might be "a.CC.Lachman.EDU"; reading from right to left, "EDU" is a top level domain comprising educational institutions, "Lachman" is a logical domain name, "CC" represents the Computer Center, (in this case a strictly logical entity), and "a" is a host in the Computer Center.

When reading RFC819, be aware that there are a number of errors in it.

How to Proceed

Once you have decided on a philosophy, it is worth examining the available configuration tables to decide whether any of them are close enough for you to steal their major parts. Even under the worst of conditions, there is a fair amount of boilerplate that can be collected safely.

The next step is to build ruleset three. This will be the hardest part of the job. Beware of doing too much to the address in this ruleset, because

anything you do will reflect through to the message. In particular, stripping of local domains is best deferred, as this can leave you with addresses with no domain specs at all. Because **sendmail** likes to append the sending domain to addresses with no domain, this can change the semantics of addresses. Also, try to avoid fully qualifying domains in this ruleset. Although technically legal, this can lead to unpleasantly and unnecessarily long addresses reflected into messages. The Lachman configuration files define ruleset nine to qualify domain names and strip local domains. This is called from ruleset zero to get all addresses into a cleaner form.

Once you have ruleset three finished, the other rulesets should be relatively trivial. If you need hints, examine the supplied configuration tables.

Testing the Rewriting Rules: the -bt Flag

When you build a configuration table, you can do a certain amount of testing using the "test mode" of **sendmail**. For example, you could invoke **sendmail** as:

```
sendmail -bt -Ctest.cf
```

which would read the configuration file "test.cf" and enter test mode. In this mode, you enter lines of the form:

```
rwset address
```

where *rwset* is the rewriting set you want to use and *address* is an address to which to apply the set. Test mode shows you the steps it takes as it proceeds, finally showing you the address with which it ends up. You may use a comma-separated list of rwsets for sequential application of rules to an input; ruleset three is always applied first. For example:

```
1,21,4 monet:bollard
```

first applies ruleset three to the input "monet:bollard." Ruleset one is then applied to the output of ruleset three, followed similarly by rulesets twenty-one and four.

If you need more detail, you can also use the "-d21" flag to turn on more debugging. For example:

```
sendmail -bt -d21.99
```

turns on an incredible amount of information. A single-word address will probably print out several pages of information.

Building Mailer Descriptions

To add an outgoing mailer to your mail system, you must define the characteristics of the mailer.

Each mailer must have an internal name. This can be arbitrary, except that the names "local" and "prog" must be defined.

The pathname of the mailer must be given in the P field. If this mailer should be accessed via an IPC connection, use the string "[IPC]" instead.

The F field defines the mailer flags. You should specify an "f" or "r" flag to pass the name of the sender as a **-f** or **-r** flag, respectively. These flags are only passed if they were passed to **sendmail**, so that mailers that give errors under some circumstances can be placated. If the mailer is not picky, you can just specify "-f $g" in the argv template. If the mailer must be called as **root**, the S flag should be given. This will not reset the userid before calling the mailer. If this mailer is local (that is, it will perform final delivery rather than another network hop), the l flag should be given. Quote characters (backslashes and " marks) can be stripped from addresses if the s flag is specified. If this is not given, they are passed through. If the mailer is capable of sending to more than one user on the same host in a single transaction, the m flag should be stated. If this flag is on, then the argv template containing **$u** will be repeated for each unique user on a given host. The e flag will mark the mailer as being expensive, which will cause **sendmail** to defer connection until a queue run.

An unusual case is the C flag. This flag applies to the mailer that the message is received from, rather than the mailer being sent to; if set, the domain spec of the sender (that is, the @host.domain part) is saved and is appended to any addresses in the message that do not already contain a domain spec. For example, a message of the form:

```
From: eric@ucbarpa
To: wnj@monet, mckee
```

will be modified to:

```
From: eric@ucbarpa
To: wnj@monet, mckee@ucbarpa
```

if and only if the C flag is defined in the mailer corresponding to eric@ucbarpa.

The S and R fields in the mailer description are per-mailer rewriting sets to be applied to sender and recipient addresses, respectively. These are applied after the sending domain is appended and the general rewriting

sets (numbers one and two) are applied, but before the output rewrite (ruleset four) is applied. A typical use is to append the current domain to addresses that do not already have a domain. For example, a header of the form:

```
From: eric
```

might be changed to be:

```
From: eric@ucbarpa
```

or:

```
From: ucbvax!eric
```

depending on the domain it is being shipped into. These sets can also be used to do special-purpose output rewriting in cooperation with ruleset four.

The E field defines the string to use as an end-of-line indication. A string containing only newline is the default. The usual backslash escapes (\r, \n, \f, \b) may be used.

Finally, an argv template is given as the E field. It may have embedded spaces. If there is no argv with a **$u** macro in it, **sendmail** will speak SMTP to the mailer. If the pathname for this mailer is [IPC], the argv should be:

IPC $h [*port*]

where *port* is the optional port number to connect to.

For example, the specifications:

```
Mlocal, P=/usr/bin/mail, F=lsDFMmnPS S=10, R=20, A=lmail $u
Mether, P=[IPC], F=meC, S=11, R=21, A=IPC $h, M=100000
```

specify a mailer to do local delivery and a mailer for Ethernet delivery. The first is called local; it is located in the file /bin/mail, takes a picky **-r** flag, and does local delivery; quotes should be stripped from addresses, and multiple users can be delivered at once; ruleset ten should be applied to sender addresses in the message, and ruleset twenty should be applied to recipient addresses. The argv to send to a message will be the word mail, the word -d, and words containing the name of the receiving user. If a **-r** flag is inserted, it will be between the words mail and -d. The second mailer is called ether; it should be connected to via an IPC connection; it can handle multiple users at once; connections should be deferred; and any domain from the sender address should be appended to any receiver

name without a domain. Sender addresses should be processed by ruleset eleven and recipient addresses by ruleset twenty-one. There is a 100,000-byte limit on messages passed through this mailer.

Command Line Flags

Arguments must be presented with flags before addresses. The flags are:

-f *addr* The sender's machine address is *addr*. This flag is ignored unless the real user is listed as a "trusted user" or *addr* contains an exclamation point (because of certain restrictions in UUCP).

-r *addr* An obsolete form of **-f**.

-h *cnt* Sets the "hop count" to *cnt*. This represents the number of times this message has been processed by **sendmail** (to the extent that it is supported by the underlying networks). *cnt* is incremented during processing, and if it reaches MAXHOP (currently 30) **sendmail** throws away the message with an error.

-F*name* Sets the full name of this user to *name*.

-n Do not alias or forward.

-t Read the header for To:, Cc:, and Bcc: lines, and send to everyone in those lists. The Bcc: line will be deleted before sending. Any addresses in the argument vector will be deleted from the send list.

-b*x* Set operation mode to *x*. The operation modes are:

m	Deliver mail (default)
a	Run in ARPANET mode (see below)
s	Speak SMTP on input side
d	Run as a daemon
t	Run in test mode
v	Just verify addresses, do not collect or deliver
i	Initialize the alias database
p	Print the mail queue
z	Freeze the configuration file

The special processing for the ARPANET includes reading the From: line from the header to find the sender, printing ARPANET-style messages (preceded by three-digit reply codes for compatibility with the FTP protocol), and ending lines of error messages with <CRLF>.

-q*time* Try to process the queued up mail. If the time is given, **sendmail** will run through the queue at the specified interval to deliver queued mail; otherwise, it only runs once.

-C*file* Use a different configuration file. The **sendmail** program runs as the invoking user (rather than root) when this flag is specified.

-d*level* Set debugging level.

-o*x value* Set option *x* to the specified *value*. These options are described in the next section.

There are a number of options that can be specified as primitive flags (provided for compatibility with **delivermail**). These are the e, i, m, and v options. Also, the f option can be specified as the **-s** flag.

Configuration Options

The following options can be set using the **-o** flag on the command line or the **O** line in the configuration file. Many of them cannot be specified unless the invoking user is trusted.

A*file* Use the named *file* as the alias file. If no file is specified, use *aliases* in the current directory.

a*N* If set, wait up to *N* minutes for an @:@ entry to exist in the alias database before starting up. If it does not appear in *N* minutes, rebuild the database (if the **D** option is also set) or issue a warning.

B*c* Set the blank substitution character to *c*. Unquoted spaces in addresses are replaced by this character.

c If an outgoing mailer is marked as being expensive, do not connect immediately. This requires that queueing be compiled in, since it will depend on a queue run process to actually send the mail.

d*x* Deliver in mode *x*. Legal modes are:

- i Deliver interactively (synchronously)
- b Deliver in background (asynchronously)
- q Just queue the message (deliver during queue run)

D If set, rebuild the alias database if necessary and possible. If this option is not set, **sendmail** will never rebuild the alias database unless explicitly requested using **-bi**.

e*x* Dispose of errors using mode *x*. The values for *x* are:

- p Print error messages (default)
- q No messages, just give exit status
- m Mail back errors
- w Write back errors (mail if user not logged in)
- e Mail back errors and give zero exit stat always

F*n* The temporary file mode, in octal. 644 and 600 are good choices.

f — Save UNIX-style "From" lines at the front of headers. Normally, they are assumed redundant and discarded.

g*n* — Set the default group id for mailers to run into *n*.

H*file* — Specify the help file for SMTP.

i — Ignore dots in incoming messages.

L*n* — Set the default log level to *n*.

M*x value* — Set the macro *x* to *value*. This is intended only for use from the command line.

m — Send to me, too, even if I am in an alias expansion.

N*netname* — The name of the home network (ARPA is the default). The argument of an SMTP HELO command is checked against hostname.netname, where *hostname* is requested from the kernel for the current connection. If they do not match, Received: lines are augmented by the name that is determined in this manner, so that messages can be traced accurately.

o — Assume that the headers may be in old format; that is, spaces delimit names. This actually turns on an adaptive algorithm: if any recipient address contains a comma, parenthesis, or angle bracket, it will be assumed that commas already exist. If this flag is not on, only commas delimit names. Headers are always output with commas between the names.

Q*dir* — Use the named *dir* as the queue directory.

q*factor* — Use *factor* as the multiplier in the map function to decide when just to queue up jobs rather than run them. This value is divided by the difference between the current load average and the load average limit (**x** flag) to determine the maximum message priority that will be sent. Defaults to 10,000.

r*time* — Timeout reads after *time* interval.

S*file* — Log statistics in the named *file*.

s — Be extra safe when running things, that is, always instantiate the queue file, even if you are going to attempt immediate delivery. The **sendmail** program

always instantiates the queue file before returning control the client.

T*time* — Set the queue timeout to *time*. After this interval, messages that have not been successfully sent will be returned to the sender.

t*S,D* — Set the local time zone name to *S* for standard time and *D* for daylight time. This is only used under version six.

u*n* — Set the default userid for mailers to *n*. Any mailer without the *S* flag in the mailer definition will run as this user.

v — Run in verbose mode.

x*LA* — When the system-load average exceeds *LA*, just queue messages (that is, do not try to send them).

X*LA* — When the system load average exceeds *LA*, refuse incoming SMTP connections.

y*fact* — The indicated *fact*or is added to the priority (thus *lowering* the priority of the job) for each recipient, that is, this value penalizes jobs with large numbers of recipients.

Y — If set, deliver each job that is run from the queue in a separate process. Use this option if you are short of memory, since the default tends to consume considerable amounts of memory while the queue is being processed.

z*fact* — The indicated *fact*or is multiplied by the message class (determined by the Precedence: field in the user header and the **P** lines in the configuration file) and subtracted from the priority. Thus, messages with a higher Priority will be favored.

Z*fact* — The *fact*or is added to the priority every time a job is processed. Thus, each time a job is processed, its priority will be decreased by the indicated value. In most environments, this should be positive, since hosts that are down are all too often down for a long time.

Mailer Flags

The following flags can be set in the mailer description.

f The mailer wants a **-f***from* flag, but only if this is a network forward operation (that is, the mailer will give an error if the executing user does not have special permissions).

r Same as **f**, but sends a **-r** flag.

S Do not reset the userid before calling the mailer. This would be used in a secure environment where **sendmail** ran as root. This could be used to avoid forged addresses. This flag is suppressed if given from an unsafe environment (for example, a user's mail.cf file).

n Do not insert a UNIX-style From: line on the front of the message.

l This mailer is local (that is, final delivery will be performed).

s Strip quote characters off the address before calling the mailer.

m This mailer can send to multiple users on the same host in one transaction. When a **$u** macro occurs in the *argv* part of the mailer definition, that field will be repeated as necessary for all qualifying users.

F This mailer wants a From: header line.

D This mailer wants a Date: header line.

M This mailer wants a Message-Id: header line.

x This mailer wants a Full-Name: header line.

P This mailer wants a Return-Path: line.

u Uppercase should be preserved in user names for this mailer.

h Uppercase should be preserved in host names for this mailer.

A This is an Arpanet-compatible mailer, and all appropriate modes should be set.

U This mailer wants UNIX-style From: lines with the ugly UUCP-style "remote from <host>" on the end.

e This mailer is expensive to connect to, so try to avoid connecting normally. Any necessary connection will occur during a queue run.

X This mailer want to use the hidden dot algorithm as specified in RFC821. Basically, any line beginning with a dot will have an extra dot prepended (to be stripped at the other end). This ensures that lines in the message containing a dot will not terminate the message prematurely.

L Limit the line lengths as specified in RFC821.

P Use the return-path in the SMTP "MAIL FROM:" command, rather than just the return address. Although this is required in RFC821, many hosts do not process return paths properly.

I This mailer will be speaking SMTP to another **sendmail**. As such, it can use special protocol features. This option is not required (that is, if this option is omitted, the transmission will still operate successfully, although perhaps not as efficiently as possible).

C If mail is received from a mailer with this flag set, any addresses in the header that do not have an at sign (@) after being rewritten by ruleset three will have the @domain clause from the sender tacked on. This allows mail with headers of the form:

```
From: usera@hosta
To: userb@hostb, userc
```

to be rewritten automatically as:

```
From: usera@hosta
To: userb@hostb, userc@hosta
```

E Escape lines beginning with From in the message with a '>' sign.

Summary of Support Files

This is a summary of the support files that **sendmail** creates or generates.

/usr/lib/sendmail	The binary of **sendmail**.
/usr/bin/newaliases	A link to */usr/lib/sendmail*; causes the alias database to be rebuilt. Running this program is completely equivalent to giving **sendmail** the **-bi** flag.
/usr/bin/mailq	Prints a listing of the mail queue. This program is equivalent to using the **-bp** flag to **sendmail**.
/usr/lib/sendmail.cf	The configuration file, in textual form.
/usr/lib/sendmail.fc	The configuration file represented as a memory image.
/usr/lib/sendmail.hf	The SMTP help file.
/usr/lib/sendmail.st	A statistics file; need not be present.
/usr/lib/aliases	The textual version of the alias file.
/usr/lib/aliases.{pag,dir}	The alias file in **dbm**(S) format.
/usr/spool/mqueue	The directory in which the mail queue and temporary files reside.
*/usr/spool/mqueue/qf**	Control (queue) files for messages.
*/usr/spool/mqueue/df**	Data files.
*/usr/spool/mqueue/lf**	Lock files
*/usr/spool/mqueue/tf**	Temporary versions of the qf files, used during queue-file rebuild.
*/usr/spool/mqueue/nf**	A file used when creating a unique id.
*/usr/spool/mqueue/xf**	A transcript of the current session.

Chapter 4

Name Server Operations Guide for BIND

Introduction

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. The Berkeley Internet Name Domain (BIND) Server implements the DARPA Internet name server for the UNIX operating system. In effect, this is a distributed database system for objects in a computer network. BIND is fully integrated into 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 does not use BIND. BIND is initially disabled. If you want to use it, you must first set up the necessary configuration files.

The Name Service

The basic function of the name server is to provide information about network objects by answering queries. The advantage of using a name server over the host table lookup for host-name resolution is to avoid 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 well for small networks where there are only a few machines and the different organizations responsible for them cooperate. However, 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, 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. Additional top-level domains include:

COM	Commercial Organizations
GOV	Government Organizations
MIL	Military Departments
ORG	Miscellaneous Organizations

Types of Servers

There are several types of servers. These are:

- master servers
- caching-only servers
- remote servers
- slave servers

These types of servers are described in more detail in the following four sections.

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 multiple domains, being primary for some domains and secondary for others.

Primary

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

Secondary

A secondary master server is a server 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.

Caching-Only Servers

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 that have the authority for the information needed. All servers keep data in their caches until the data expires, based on a time-to-live field attached to the data when it is received from another server.

Remote Servers

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 running the name server on the local machine. All of the queries are serviced by a name server that is running on another machine on the network.

Slave Server

A slave server is a server that always forwards queries it cannot satisfy locally to a fixed list of forwarding servers instead of interacting with the master name servers for the root and other domains. The queries to the forwarding servers are recursive queries. There may be one or more forwarding servers, and they are tried in turn until the list is exhausted. A slave and forwarder configuration is typically used when you do not wish all the servers at a given site to be interacting with the rest of the Internet servers. A typical scenario would involve a number of workstations and a departmental timesharing machine with Internet access. The workstations might be administratively prohibited from having Internet access. To give the workstations the appearance of access to the Internet domain system, the workstations could be slave servers to the timesharing machine, which would forward the queries and interact with other name servers to resolve the query before returning the answer. An added benefit of using the forwarding feature is that the central machine develops a much more complete cache of information that all the workstations can take advantage of. The use of slave mode and forwarding is discussed further under the description of the named bootfile commands.

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 as follows:

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. This group discusses future design decisions, operational problems, and other related topics. To request placement on this mailing list, send mail to the following address:

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

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 prefers to send mail via CSNET if possible (as opposed to BITNET or the Internet). 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.cs.net, or by phone at (617) 497-2777.

BITNET

If you are on the BITNET and need to set up a domain, contact INFO@BITNIC.

Boot File

The name server uses several files to load its database. The major file used is the boot file. This is the file that is first read when **named** starts up. This 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** starts up.

Domain

The boot file contains a line of code that designates the default domain. The line for the server looks like this:

```
domain      Berkeley.Edu
```

The name server uses this information when it receives a query for a name without a "." that is unknown. When it receives one of these queries, it appends the name in the second field to the query name. This is an obsolete facility, which will be removed from future releases.

Directory

The directory line specifies the directory in which the name server should run, allowing the other filenames in the boot file to use relative pathnames.

```
directory           /usr/local/lib/named
```

If you have more than a couple of named files to be maintained, you may wish to place the named files in a directory such as */usr/local/domain* and adjust the directory command properly. The main purposes of this command are to make sure **named** is in the proper directory when trying to include files by relative pathnames with $INCLUDE and to allow **named** to run in a location that is reasonable to dump core if it feels the urge.

Primary Master

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

```
primary             Berkeley.Edu    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.

Secondary Master

The line for a secondary server is similar to that for the primary, except that it lists addresses of other servers (usually primary servers) from which the zone data is obtained.

```
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 two network addresses specify 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. If a filename is present after the list of primary servers, data for the zone is dumped into that file as a backup. When the server is first started, the data are loaded from the backup file if possible, and a primary server is then consulted to check that the zone is still up-to-date.

Caching-Only Server

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

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

```
cache              .                root.cache
```

The period (.) specifies the current domain. All cache files listed are read in at named boot time and any values still valid are reinstated in the cache and the root name server information in the cache files are always used. For information on the cache file, see the later section, "Initializing the Cache."

Forwarders

Any server can make use of forwarders. A forwarder is another server capable of processing recursive queries to try to resolve queries on behalf of other systems. The **forwarders** command specifies forwarders by internet address as follows:

```
forwarders          128.32.0.10  128.32.0.4
```

There are two main reasons for wanting to do so. First, the other systems may not have full network access and may be prevented from sending any IP packets into the rest of the network and, therefore, must rely on a forwarder that does have access to the full net. The second reason is that the forwarder sees a union of all queries as they pass through the forwarder's server and, therefore, the forwarder builds up a very rich cache of data compared to the cache in a typical workstation name server. In effect, the forwarder becomes a meta-cache that all hosts can benefit from, thereby reducing the total number of queries from that site to the rest of the net.

Slave Mode

Slave mode is used if the use of forwarders is the only possible way to resolve queries because of lack of full net access or if you wish to prevent the name server from using other than the listed forwarders. Slave mode is activated by placing the simple command

slave

in the bootfile. If **slave** is used, then you must specify forwarders. When in slave mode, the server forwards each query to each of the forwarders until an answer is found or the list of forwarders is exhausted.

Remote Servers

To set up a host that uses a remote server instead of a local server to answer queries, create the file */etc/resolv.conf*. This file designates the name servers on the network that should be sent queries. 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**(SLIB) or **gethostbyaddr** is called.

Initializing the Cache

The name server needs to know the identities of the authoritative name servers for the root domain of the network. To do this, you have to prime the name server's cache with the address of these higher authorities. This is done in a file called *root.cache*. The location of this file is specified in the boot file */etc/named.boot*.

There are three standard files used to specify the data for a domain. These files are:

named.local
hosts
host.rev.

The *named.local* 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.

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

The *hosts.rev* file specifies the IN-ADDR.ARPA domain. This is a special domain for allowing address-to-name mapping. Because 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 an octet 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.

Standard Resource Records

The records in the name server data files are called resource records. The following is a general description of a resource record:

```
{name}    {ttl}    addr-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 and it must always start in column 1. For some resource records, the name can be left blank. In such cases, the name of the previous resource record is used. The second field is an optional time-to-live field. This specifies how long this data is stored in the database. When this field is left blank, the default time-to-live is specified in the Start of Authority resource record discussed later in this chapter. 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 the resource record. 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), \X 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, \DDD 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 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 can cause problems where you do not want this to happen. The following is a good rule of thumb: if the name is not in the domain for which you are creating the data file, end the name with a ".".

Separating Data into Multiple Files

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. Here is an example:

```
$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.

Changing an Origin in a Data File

Use the $ORIGIN command to change 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. For example, */etc/named.hosts* might contain lines of the form:

```
$ORIGIN CC.Berkeley.EDU
[assorted domain data...]
$ORIGIN EE.Berkeley.EDU
[assorted domain data...]
```

The Start of Authority Resource Record (SOA)

The Start of Authority record designates the start of a zone. An SOA record includes the following fields:

- Name
- Origin
- Person in charge
- Serial number
- Refresh
- Retry
- Expire
- Minimum

"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. "Serial number" is the version number of this data file; this number should be incremented whenever a change is made to the data. (Note that 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 time 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 time-to-live field on resource records. There should only be one SOA record per zone. Here is an example of an SOA record:

```
name  {ttl}  addr-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 Name Server Resource Record (NS)

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. Here is an example of a name server record:

```
{name}    {ttl}    addr-class    NS    Name servers name
                   IN            NS    ucbarpa.Berkeley.Edu.
```

The address class is IN (Internet addresses), and the record type is name server (NS). The record uses the default ttl (time-to-live) value. Here, the record-specific data is the identity of the name server.

The Address Resource Record (A)

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. Here is an example of an address record for a machine named *ucbarpa* with two network addresses:

```
{name}    {ttl}    addr-class    A    address
ucbarpa            IN            A    128.32.0.4
                   IN            A    10.0.0.78
```

The Host Information Resource Record (HINFO)

The host information resource record (HINFO) is for host-specific data. It 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 information and the operating-system information. If you want to include a space in the machine name, you must quote the name. Host information is not specific to any address class, so ANY may be used for the address class. There should be one HINFO record for each host. Here is an example:

```
{name}    {ttl}    addr-class    HINFO    Hardware      OS
                   ANY           HINFO    VAX-11/780    UNIX
```

Note that the current release ignores any records that appear after an HINFO record. Thus, you can use only one HINFO record within the file, and it should be the last record in the file.

The Well-Known Services Resource Record (WKS)

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 comes from the list of services specified in */etc/services*. There should be only one WKS record per protocol per address. Here is an example of a WKS record:

```
{name}  {ttl} addr-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
                                                     name server)
```

The Canonical Name Resource Record (CNAME)

The canonical name resource record (CNAME) specifies an alias for a canonical name. An alias should be the only record associated with the alias name; all other resource records should be associated with the canonical name and not with the alias. Any resource records that include a domain name as their value (for example, NS or MX) should list the canonical name, not the alias. Here is an example of a CNAME record:

```
aliases     {ttl}   addr-class   CNAME   Canonical name
ucbmonet            IN           CNAME   monet
```

The Domain Name Pointer Resource Record (PTR)

A domain name pointer record (PTR) allows special names to point to some other location in the domain. The following 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.
```

In this record, the name field is the network number of the host in reverse order. You only need to specify enough octets to make the name unique. For example, if all hosts are on network 127.174.14, then only the last octet needs to be specified. If hosts are on networks 128.174.14 and 127.174.23, then the last two octets need to be specified. PTR names should be unique to the zone. Here is an example of a PTR record:

```
name    {ttl}   addr-class   PTR   real name
7.0             IN           PTR   monet.Berkeley.Edu.
```

The Mailbox Resource Record (MB)

The mailbox resource record has a record type of MB. It 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. Mail box names should be unique to the zone. Here is an example of an MB record:

```
name      {ttl}   addr-class   MB   Machine
miriam            IN           MB   vineyd.DEC.COM.
```

The Mail Rename Resource Record (MR)

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. Here is an example of a mail rename record:

```
name            {ttl}   addr-class   MR   corresponding MB
Postmistress            IN           MR   miriam
```

The Mailbox Information Resource Record (MINFO)

The mail information record MINFO creates a mail group for a mailing list. This resource record is usually associated with a mail group, but it can be used with a mail box 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. Here is an example of this record:

```
name    {ttl}   addr-class   MINFO   requests       maintainer
BIND            IN           MINFO   BIND-REQUEST   kjd.Berkeley.Edu.
```

The Mail Group Member Resource Record (MG)

The mail group record (MG) lists members of a mail group.

```
{mail group name}    {ttl}    addr-class    MG    member name
                              IN            MG    Bloom
```

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.
```

The Mail Exchanger Resource Record (MX)

```
name            {ttl}  addr-class  MX  preference value  mailer exchanger
Munnari.OZ.AU.  IN                 MX  0                 Seismo.CSS.GOV.
*.IL.           IN                 MX  0                 RELAY.CS.NET.
```

Mail exchanger records (MX) are used to identify a machine that knows how to deliver mail to a machine that is not directly connected to the network. In the first 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 cannot 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 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.

Some Sample Files

The following sections contain sample files for the name server. This covers example boot files for the different types of server and example domain database files.

Caching-Only Server

```
;
; Boot file for Caching Only 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
```

Primary Master Server

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

; type       domain                  source file or host
;
directory    /usr/local/lib/named
primary      Berkeley.Edu            ucbhosts
primary      32.128.in-addr.arpa     ucbhosts.rev
primary      0.0.127.in-addr.arpa    named.local
cache        .                       root.cache
```

Secondary Master Server

```
;
; Boot file for Secondary Name Server
;
; type      domain                source file or host
;
directory   /usr/local/lib/named
secondary   Berkeley.Edu          128.32.0.4 128.32.0.10 128.32.136.22 ucbhost.bak
secondary   32.128.in-addr.arpa   128.32.0.4 128.32.0.10 128.32.136.22 ucbhosts.rev.bak
primary     0.0.127.in-addr.arpa  named.local
cache       .                     root.cache
```

The /etc/resolv.conf File

```
domain Berkeley.Edu
name server 128.32.0.4
name server 128.32.0.10
```

root.cache

```
;
; Initial cache data for root domain servers.
;
.                       99999999   IN   NS   SRI-NIC.ARPA.
                        99999999   IN   NS   NS.NASA.GOV.
                        99999999   IN   NS   TERP.UMD.EDU.
                        99999999   IN   NS   A.ISI.EDU.
                        99999999   IN   NS   BRL-AOS.ARPA.
                        99999999   IN   NS   GUNTER-ADAM.ARPA.
                        99999999   IN   NS   C.NYSER.NET.
; Prep the cache (hotwire the addresses).
SRI-NIC.ARPA.           99999999   IN   A    10.0.0.51
SRI-NIC.ARPA.           99999999   IN   A    26.0.0.73
NS.NASA.GOV.            99999999   IN   A    128.102.16.10
BRL-AOS.ARPA.           99999999   IN   A    128.20.1.2
A.ISI.EDU.              99999999   IN   A    26.3.0.103
BRL-AOS.ARPA.           99999999   IN   A    192.5.25.82
GUNTER-ADAM.ARPA.       99999999   IN   A    26.1.0.13
C.NYSER.NET.            99999999   IN   A    128.213.5.17
TERP.UMD.EDU.           99999999   IN   A    10.1.0.17
```

named.local

```
@    IN   SOA   ucbvax.Berkeley.Edu. kjd.ucbvax.Berkeley.Edu. (
                1       ; Serial
                10800     ; Refresh
                1800      ; Retry
                3600000 ; Expire
                86400 )   ; Minimum
     IN   NS    ucbvax.Berkeley.Edu.
1    IN   PTR   localhost.
```

hosts

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

@                IN   SOA     ucbvax.Berkeley.Edu. kjdmonet.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
                 IN   HINFO   VAX-11/780 UNIX
arpa             IN   CNAME   ucbarpa
ernie            IN   A       128.32.6
                 IN   HINFO   VAX-11/780 UNIX
ucbernie         IN   CNAME   ernie
monet            IN   A       128.32.7
                 IN   A       128.32.130.6
                 IN   HINFO   VAX-11/750 UNIX
ucbmonet         IN   CNAME   monet
```

```
ucbvax          IN  A       10.2.0.78
                IN  A       128.32.10
                IN  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
                            name server )
vax             IN  CNAME   ucbvax
toybox          IN  A       128.32.131.119
                IN  HINFO   Pro350 RT11
toybox          IN  MX      0 monet.Berkeley.Edu
miriam          IN  MB      vineyd.DEC.COM.
postmistress    IN  MR      Miriam
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.
```

hosts.rev

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

@       IN  SOA   ucbvax.Berkeley.Edu. kjd.monet.Berkeley.Edu. (
                  1.1     ; Serial
                  10800   ; Refresh
                  1800    ; Retry
                  3600000 ; Expire
                  86400 ) ; 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.
```

Additional Sample Files

The following sections contain an additional set of sample files for the name server.

named.boot

```
;
; Name Server boot file for Domain sco.COM
;
; Type        Domain        Source file or Host
;
domain        sco.COM
primary       sco.COM       /etc/named.data/sco-hosts
cache         .             /etc/named.data/root.cache
secondary     sco.COM       /etc/named.data/sco-host.s.rev
primary       sco.COM       /etc/named.data/named.local
```

root.cache

```
;
; Initial cached data for root domain servers.
;
;.                99999999 IN NS    USC-ISIB.ARPA.
;                 99999999 IN NS    BRL-AOS.ARPA.
;                 99999999 IN NS    SRI-NIC.ARPA.
;
; Insert your own name servers here
;
;                 99999999 IN NS    scovert.sco.COM
;
; Prep the cache (hotwire the addresses)
;
tandy.sco.COM.          99999999 IN A     192.9.200.2
;viscous.sco.COM. 99999999 IN A     128.0.21.6
;
; Root servers go here
;
tandy.sco.COM.          99999999 IN A     192.9.200.2
;SRI-NIC.ARPA.          99999999 IN A     10.0.0.51
;USC-ISIB.ARPA.         99999999 IN A     10.3.0.52
;BRL-AOS.ARPA.          99999999 IN A     128.20.1.2
;BRL-AOS.ARPA.          99999999 IN A     192.5.22.82
```

named.local

```
;
; Don't forget to increment the serial number in
; named.soa
;

$INCLUDE /etc/named/sco.soa
192.9.200.2 IN PTR      localhost.
```

sco-host.s.rev

```
;
; Don't forget to increment the serial number in
; named.soa
;

$INCLUDE /etc/named/sco.soa

192.9.200.1 IN    PTR   merlin
192.9.200.2 IN    PTR   tandy
192.9.200.3 IN    PTR   tvi
```

sco.soa

```
;
; Don't forget to increment the serial number when you
; change this.  SCCS or RCS might be a good idea here.
;

@           IN SOA  tandy.sco.COM.   root.tandy.sco.COM. (
                  1.0     ; Serial
                  3600    ; Refresh
                  300     ; Retry
                  3600000 ; Expire
                  3600 )  ; Minimum
            IN NS   tandy.sco.COM.
```

Domain Management

This section contains information for starting, controlling, and debugging **named**(ADMN), the Internet domain name server.

Starting the Name Server

The host name should be set to the full domain style name (that is, monet.Berkeley.EDU.) using **hostname**(TC). The name server is started automatically if the configuration file */etc/named.boot* is present. Do not attempt to run **named** from **inetd**(ADMN). This continuously restarts the name server and defeats the purpose of having a cache.

/etc/named.pid

When **named** is successfully started, 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 can be changed by defining PIDFILE to the new name when compiling **named**.

/etc/hosts

The **gethostbyname** library call can detect whether **named** is running. If it is determined that **named** is not running, it looks in */etc/hosts* to resolve an address. This option was added to allow **ifconfig**(ADMN) to configure the machine's 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 copy files from another machine with **rcp** when the system is in single-user mode. The format of */etc/hosts* has not changed. See **hosts**(SFF) for more information. Because the process of reading */etc/hosts* is slow, it is not advisable to use this option when the system is in multiuser mode.

Reload

There are several signals that can be sent to the **named** process to have it do tasks without restarting the process. The SIGHUP signal 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.

Debugging

When **named** is running incorrectly, look first in */usr/adm/syslog* 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 can be changed by defining DUMPFILE to the new name when compiling **named**.

Note

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 can 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 */usr/lib/libsocket.a*.

Chapter 5

Synchronizing Network Clocks

Introduction

The clock synchronization service is composed of a collection of time daemons (**timed**(ADMN)) running on the machines in a local-area network. The algorithms implemented by the service are based on a master-slave scheme. The time daemons communicate with each other using the Time Synchronization Protocol (TSP), which is built on the DARPA UDP protocol.

A time daemon has a two-fold function. First, it supports the synchronization of the clocks of the various hosts in a local-area network. Second, it starts (or takes part in) the election that occurs among slave time daemons when, for any reason, the master disappears. The synchronization mechanism and the election procedure employed by the program **timed** are described in the manual page **timed**(ADMN). This chapter is mainly concerned with the administrative and technical issues of running **timed** at a particular site. The next section is a brief overview of how the time daemon works. A master time daemon measures the time differences between the clock of the machine on which it is running and those of all other machines on its network. The master computes the network time as the average of the times provided by nonfaulty clocks. (A clock is considered to be faulty when its value is more than a small specified interval apart from the majority of the clocks of the other machines.) The master time daemon then sends to each slave time daemon the correction that should be performed on the clock of its machine. This process is repeated periodically.

Because the correction is expressed as a time difference rather than an absolute time, transmission delays do not interfere with the accuracy of the synchronization. When a machine comes up and joins the network, it starts a slave time daemon that asks the master for the correct time and resets the machine's clock before any user activity can begin. The time daemons are thus able to maintain a single network time in spite of the drift of clocks away from each other. The present implementation is capable of keeping processor clocks synchronized to within 20 milliseconds, but some hardware is not adjustable at less than 1 second intervals.

To ensure that the service provided is continuous and reliable, it is necessary to implement an election algorithm to elect a new master should the machine running the current master crash, the master terminate (for example, because of a runtime error), or the network be partitioned.

Under this algorithm, slaves are able to realize when the master has stopped functioning and to elect a new master from among themselves. It is important to note that the failure of the master results only in a gradual divergence of clock values; thus, the election need not occur immediately.

The machines that are gateways between distinct local-area networks require particular care. A time daemon on such machines may act as a "submaster." This artifact depends on the current inability of transmission protocols to broadcast a message on a network other than the one to which the broadcasting machine is connected. The submaster appears as a slave on one network and as a master on one or more of the other networks to which it is connected.

A submaster classifies each network as one of three types. A slave network is a network on which the submaster acts as a slave. There can only be one slave network. A master network is a network on which the submaster acts as a master. An ignored network is any other network that already has a valid master. The submaster tries periodically to become master on an ignored network, but gives up immediately if a master already exists.

Guidelines

While the synchronization algorithm is quite general, the election algorithm, which requires a broadcast mechanism, puts constraints on the kind of network on which time daemons can run. The time daemon works only on networks with broadcast capability augmented with point-to-point links. Machines that are only connected to point-to-point, non-broadcast networks cannot use the time daemon.

If submasters are excluded, there is normally only one master time daemon in a local-area internetwork. During an election, only one of the slave time daemons becomes the new master. Not all machines are suitable as masters; some do not have sufficiently accurate timing mechanisms or cannot afford the extra overhead. Therefore, a subset of machines must be designated as potential master time daemons. A master time daemon requires CPU resources proportional to the number of slaves (in general, more than a slave time daemon), and so it may be advisable to limit master time daemons to machines with more powerful processors or lighter loads. Also, machines with inaccurate clocks should not be used as masters. This is a purely administrative decision; an organization may well allow all of its machines to run master time daemons.

At the administrative level, a time daemon on a machine with multiple network interfaces may be told to ignore all but one network or to ignore one network. This is done with the **timed -n** *network* and **-i** *network* options, respectively, at startup time. Typically, the time daemon would be instructed to ignore all but the networks belonging to the local administrative control.

There are some limitations to the current implementation of the time daemon. It is expected that these limitations will be removed in future releases. The constant NHOSTS in */usr/src/etc/timed/globals.h* limits the maximum number of machines that can be directly controlled by one master time daemon. The maximum is (NHOSTS - 1). Currently, the maximum is 99. The constant must be changed and the program recompiled if a site wishes to run **timed** on a larger network.

In addition, there is a pathological situation to be avoided at all costs. This situation can occur when time daemons run on multiply-connected local-area networks. In this case, time daemons running on gateway machines are submasters, and they act on some of those networks as master time daemons. Consider machines A and B that are both gateways between networks X and Y. If time daemons were started on both A and B without constraints, it would be possible for submaster time daemon A to be a slave on network X and the master on network Y, while submaster

time daemon B would be a slave on network Y and the master on network X. This loop of master time daemons does not function properly or guarantee a unique time on both networks, and it causes the submasters to use large amounts of system resources in the form of network bandwidth and CPU time. In fact, this kind of loop can also be generated with more than two master time daemons, when several local-area networks are interconnected.

Options

The options for the **timed** command are:

-n *network*	Considers the named network.
-i *network*	Ignores the named network.
-t	Places tracing information in */usr/adm/timed.log*.
-M	Allows this time daemon to become a master. A time daemon run without this option is forced into the state of slave during an election.

Daily Operation

The **timedc(ADMN)** command is used to control the operation of the time daemon. It can be used to do the following:

- measure the differences between machines' clocks
- find the location where the master **timed** is running
- cause election timers on several machines to expire at the same time
- enable or disable tracing of messages received by **timed**

See the manual pages on **timed**(ADMN) and **timedc**(ADMN) for more detailed information.

The **rdate(ADMN)** command can be used to set the network date.

Glossary

ALIAS. An alternate host name, created as a convenience for addressing a host on a local network whose unique primary name is long and/or complicated.

ARP. Address Resolution Protocol is used by Ethernet for address mapping.

ARPA. Now called DARPA, stands for Defense Advanced Research Projects Agency. ARPANET is the network based on the work sponsored by this agency. See also DDN.

BIND. Berkeley Internet Name Domain. Also: bind. To fix an association between a name and an object. In networking, used to explicitly assign a network address to a socket.

BRIDGE. A simplified gateway used to connect local networks that use the same internal protocols and exhibit the same interface to attach stations.

BROADCAST NETWORK. A system in which messages are sent to all hosts simultaneously, rather than from point to point. Each node then "grabs" the transmissions intended for them.

BSD. Berkeley Software Distribution.

BUS. A set of one or more parallel signals implemented in hardware in a standard manner so that multiple devices can access it and communicate over it.

CACHE. To store temporarily in memory to improve access performance. Also, that which is stored temporarily in memory.

CACHING-ONLY SERVER. A server that is not authoritative for any domain. This server services queries and asks other servers that have the authority for the information needed.

CCITT. The Comite Consultatif Internationale de Telegraphie et Telophonie. A communications organization that sets international usage standards. In English: International Telegraph and Telephone Consultative Committee. See X.25.

CLIENT. A computer or executing program that sends a request to a server, and waits for a response. The term "client" is generally used in the context of NFS.

CLONING DEVICE. A cloning device provides for dynamic allocation of resources by means of a single pathname.

CONNECTION. A connection is a logical communication path.

CONNECTIONLESS. A packet delivery system in which packets sent from one machine to another may follow different paths. It is called unreliable because delivery is not guaranteed. Packets may be delivered out of sequence, duplicated, or lost. However, connectionless delivery may be desirable due to its low transport overhead.

CONSUMER. A computer or executing program that receives and uses information. A subset of client. The term "consumer" is generally used in the context of LM/X.

DAEMON. A daemon is a system service. It is a program that is active in the background but not connected to a terminal. Also: demon.

DARPA. Department of Defense Advanced Research Project Agency, formerly called ARPA. This agency sponsored the network architecture research project upon which ARPANET is based. ARPANET is a large governmental internetwork, called the Internet, part of which is the Defense Data Network (DDN).

DATAGRAM. Basic transfer unit of IP. Consists of a header, containing Internet source and destination addresses, and data. Also called a packet. Datagram implies that delivery will be connectionless. Also: dgram.

DATA LINK LEVEL. Data link level is the communications protocol for the physical media-link used to transport the data.

DDN. Defense Data Network. A set of communications capabilities that links together computer systems within the Department of Defense (DoD). The DDN allows users of these computer systems to send mail and files between systems and to access other computers on the network in interactive terminal sessions. The DDN is part of the DARPA Internet. See also Internet.

DESTINATION. The destination address, an internet header field.

DESTINATION ADDRESS. Network and host identifiers.

DNS. Domain Naming System.

DOMAIN. A naming category in DNS, a hierarchical naming scheme. A domain is a set of machines usually grouped by geographic location, organization, or activity (for example, EDU for educational machines, COM for machines in commercial use).

ETHERNET. Originally, a heavily shielded, half-inch diameter coaxial cable developed by Xerox Corporation, Digital Equipment Corporation, and Intel Corporation, for use in local area networks.

FLOW CONTROL. Flow control is the function and process of regulating the traffic and amount of data between flowing nodes so that neither node is sent more data than it can handle at a given time.

GATEWAY. A protocol translator device connecting two local networks, or a local to a long-haul network. Gateways can be thought of as communication paths for the exchange of data between networks.

HOST. A host is a computer that acts as client and/or server. It is, specifically, a source or destination of messages from the point of view of the communication network.

IAB. Internet Activities Board.

ICMP. Internet Control Message Protocol. ICMP is used by a gateway or destination host to communicate with a source host, for example, to report an error in datagram processing. ICMP uses the basic support of IP as if ICMP were a higher level protocol. However, ICMP is actually an integral part of IP, and must be implemented by every IP module.

IEN. Internet Engineering Notes.

INTERNET. When capitalized, Internet refers specifically to the internet built by DARPA. Otherwise it refers to any internet.

INTERNET ADDRESS. A 32-bit universal identifier assigned to each host on the Internet.

INTERNETWORKING. The connection of networks using different hardware and/or software protocols by means of devices called gateways, for the purpose of forwarding data from one network to another. Internetworking allows several networks to function cooperatively as a single, virtual network.

LAYER. A conceptual model in protocol software in which each machine in a network can be thought of as being stacked in tiers, in which each tier, or level, handles one aspect of the process of transferring data.

LOOPBACK INTERFACE. Used for diagnostic purposes, loopback interface is software, without any associative hardware, that receives information and sends it right back to its point of origin.

MASTER SERVER. A master server is the authority for a particular domain and maintains all data corresponding to it.

NETWORK INTERFACE. Device drivers and associated hardware that allow TCP/IP software to communicate with a particular network.

NETWORK MASK. A bit mask that specifies the portion of an Internet address that is to be considered the network part for that network.

PORT. A port, or port number, is a 16-bit address used by TCP/IP to identify a socket on a particular machine.

PRIMARY MASTER SERVER. A server that loads its data from a file on disk. In a multiple master situation, this server may also delegate authority to other servers in its domain.

PROCESS. A process is a program in execution. A source or destination of data from the point of view of the Transmission Control Protocol (TCP), or other host-to-host protocol.

PROTOCOL. A set of rules for communications, including standards for message format.

RFC. Request For Comments. A document containing proposals, ideas, observations, as well as general information and accepted Internet protocol standards. RFC is usually followed by a number, which refers to a particular edition or iteration of the notes, and is available across the Internet.

ROOT. root is the login name of the super-user. The super-user is the user who has the widest form of machine privileges.

ROUTING TABLE. A collection of configuration information that allows for the dynamic and adaptive transfer of data from point to point, automatically, via the best available path.

SECONDARY MASTER SERVER. A server that is delegated authority and receives its data for a domain from a primary master server. A secondary master server functions as a master server or backup when the primary master server is unavailable.

SERVER. Any program that accepts requests over the network, performs a service, and returns the result to the machine making the request.

SLAVE SERVER. A server that always forwards queries it cannot satisfy locally to a fixed list of forwarding servers. In slave mode the server forwards each query to each of the forwarders until an answer is found or the list of forwarders is exhausted.

SOCKET. A socket provides a point of access to network software that allows use of the network.

TCP. Transmission Control Protocol is a transport level, connection-oriented protocol that provides reliable end-to-end message transmission over an internetwork.

UDP. User Datagram Protocol. A connectionless mode, user-level transport protocol for transaction-oriented applications. UDP datagrams include a protocol port number, enabling the sender to specify a particular application on the remote machine.

X.25. X.25 is a circuit-switched network protocol used commonly in Europe and less so in the United States. X.25 is based on a three-layer, peer-communications protocol standard defined by the International Telegraph and Telephone Consultative Committee (CCITT).

SCO® TCP/IP

Derived from

LACHMAN™ SYSTEM V STREAMS TCP

Administrator's Reference

The Santa Cruz Operation, Inc.

SCO TCP/IP was developed by Lachman Associates.
SCO TCP/IP is derived from LACHMAN™ SYSTEM V STREAMS TCP, a joint development of Lachman Associates and Convergent Technologies.

This document was typeset with an IMAGEN® 8/300 Laser Printer.

SCO Document Number: 11-25-89-1.1.0D
Printed: Tue May 1 14:55:53 PDT 1990

Contents

Network Commands (**ADMN**)

telnetd	DARPA TELNET protocol server
tftpd	DARPA Trivial File Transfer Protocol server
timed	time server daemon
timedc	timed control program
trace	routing tools
trpt	transliterate protocol trace

intro

introduction to network maintenance and operation commands

Description

This section contains information related to network operation and maintenance. It describes a variety of commands: *slink*, to bring up the transport; *ifconfig*, and *slattach*, to configure network interfaces; *ping*, to test status of remote hosts; *trpt*, to display packet-tracing information; to invoke network services; and and other network administration functions.

arp

address resolution display and control

Syntax

arp hostname
arp -a [namelist] [corefile]
arp -d hostname
arp -s hostname ether_addr [**temp**] [**pub**] [**trail**]
arp -f filename

Description

The *arp* program displays and modifies the Internet-to-Ethernet address-translation table, which is normally maintained by the address-resolution protocol (*arp*(ADMP)).

When *hostname* is the only argument, *arp* displays the current ARP entry for *hostname*. The host may be specified by name or number, using Internet dot notation. [See *hosts*(ADMN) and *inet*(ADMP).]

Options are interpreted as follows:

-a [*namelist*] [*corefile*]
Display all of the current ARP entries by reading the table from the file **corefile** (default **/dev/kmem**) based on the kernel file **namelist** (default **/unix**).

-d Delete an entry for the host whose name is *hostname*. (This can be performed only by the super user.)

-s *hostname ether_addr* [**temp**] [**pub**] [**trail**]
Create an ARP entry for the host whose name is *hostname* with the Ethernet address *ether_addr*. The Ethernet address is given as six colon-separated, two-digit hexadecimal numbers. The entry will be permanent unless the argument **temp** is specified on the command line. If **pub** is specified, the entry will be published: that is, this system will act as an ARP server, responding to requests for *hostname* even though the host address is not an address of the local host. If **trail** is specified, trailer encapsulations are to be used with this host. *N.B.* Trailers are a link-dependent issue. Currently, no known LLI-compliant ethernet driver suppports trailers, and it is unwise to advertise them, unless it is certain that the link layer can handle them.

-f *filename*
Read the file **filename** and set multiple entries in the ARP tables. Entries in the file should be of the form:

hostname ether_addr [**temp**] [**pub**] [**trail**]

with argument meanings as given above.

See Also

inet(SLIB), arp(ADMP), ifconfig(ADMN).

drvconf

configure TCP/IP ethernet drivers

Syntax

/etc/drvconf

Description

/etc/drvconf is used to configure TCP/IP to use a particular ethernet driver. It prompts with a list of possible drivers and asks the user to select one. The TCP/IP configuration files */etc/strcf* and */etc/tcp* are then modified to use the appropriate driver. The driver must be installed on the system when *drvconf* is run.

See Also

strcf(SFF), tcp(ADMN), idmknod(ADMN).

Bugs

As distributed, this command only supports drivers for the AT/386.

fingerd

remote user information server

Syntax

/etc/fingerd

Description

fingerd is a server that provides a network interface to the *finger*(TC) program (or, on some other systems, the *name* program). This interface allows *finger* to display information about remote users.

fingerd listens for TCP connections on the *finger* port. (See *services(SFF).)* For each connection, *fingerd* reads a single input line (terminated by a <CRLF>), passes the line to *finger*, and copies the output of *finger* to the user on the client machine.

fingerd is started by the super-server *inetd*, and therefore must have an entry in *inetd* 's configuration file **/etc/inetd.conf.** [See *inetd*(ADMN) and *inetd.conf*(SFF).]

For it to work, *fingerd* needs to have a **/usr/local/bin** directory created and then linked to **/usr/bin/finger**.

See Also

finger(TC), inetd(ADMN), inetd.conf(SFF), services(SFF), RFC 742.

Warning

Connecting to *fingerd* using TELNET (see *telnet*(TC)) can have unpredictable consequences and is not recommended.

ftpd

DARPA Internet File Transfer Protocol server

Syntax

/etc/ftpd [**-d**] [**-l**] [**-t***timeout*]

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(SFF)*.

ftpd is started by the super-server *inetd*, and therefore must have an entry in *inetd*'s configuration file **/etc/inetd.conf.** [See *inetd(ADMN)* and *inetd.conf(SFF).]*

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*.

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
MODE	specify data transfer *mode*
NLST	give name list of files in directory (ls)
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

RETR	retrieve a file
RMD	remove a directory
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
STOU	store a file with a unique name
STRU	specify data transfer *structure*
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

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

The FTP server will abort an active file transfer only when the ABOR command is preceded by a Telnet Interrupt Process (IP) signal and a Telnet Synch signal in the command Telnet stream, as described in Internet RFC 959.

ftpd interprets file names according to the globbing conventions used by *sh*(C). This allows users to utilize the metacharacters *?[]{}~.

ftpd authenticates users according to three rules.

1) The user name must be in the password data base **/etc/passwd** and not have a null password. In this case, a password must be provided by the client before any file operations can be performed.

2) The user name must not appear in the file **/etc/ftpusers**.

3) 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. To make sure system security is not breached, it is recommended that the ftp subtree be constructed with care; the following rules are recommended. (Note: ~ftp means "the home directory of user ftp")

~ftp)
Make it so the home directory is owned by ftp and unwritable by anyone.

~ftp/bin)
Make it so this directory is owned by the superuser and unwritable by anyone. The program *ls*(C) must be present to support the list commands. This program should have mode 111.

~ftp/etc)
Make it so this directory owned by the superuser and unwritable by anyone. The files *passwd*(SFF) and *group*(SFF) must be present for the *ls* command to work properly. These files should be mode 444.

~ftp/pub)
Make this directory mode 777 and owned by ftp. Users should then place files that are to be accessible via the anonymous account in this directory.

See Also

ftp(TC), syslog(SLIB)

Notes

The anonymous account is inherently dangerous and should avoided when possible.

The server must run as the superuser to create sockets with privileged port numbers. It maintains an effective user id of the logged in user, reverting to the superuser only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

Files

/etc/ftpusers - restricted user list
/etc/passwd - the user database
/etc/group - the group database
/usr/adm/syslog - the system log file

The following files are needed for anonymous ftp:

~ftp/etc/passwd - used by ~ftp/bin/ls
~ftp/etc/group - used by ~ftp/bin/ls
~ftp/bin/ls - to support the LIST and NLST commands

In addition, if your /bin/ls is linked with shared libraries, you will need to copy /shlib/libc_s to ~ftp/shlib/libc_s. If your implementation is using the SIOCSOCKSYS ioctl, you will need to run the *mdnod*(ADMN) command on ~ftp/dev/socksys.

hostname

host name resolution description

Description

Hostnames are domains, where a domain is a hierarchical, dot-separated list of subdomains; for example, the machine laiter, in the Lachman subdomain of the COM subdomain of the ARPANET would be represented as

laiter.Lachman.COM

(with no trailing dot).

Hostnames are often used with network client and server programs, which must generally translate the name to an address for use. (This function is generally performed by the library routine *gethostbyname* (SSC).) Hostnames are resolved by the internet name resolver in the following fashion.

If the name consists of a single component, i.e. contains no dot, and if the environment variable "HOSTALIASES" is set to the name of a file, that file is searched for an string matching the input hostname. The file should consist of lines made up of two white-space separated strings, the first of which is the hostname alias, and the second of which is the complete hostname to be substituted for that alias. If a case-sensitive match is found between the hostname to be resolved and the first field of a line in the file, the substituted name is looked up with no further processing.

If the input name ends with a trailing dot, the trailing dot is removed, and the remaining name is looked up with no further processing.

If the input name does not end with a trailing dot, it is looked up in the local domain and its parent domains until either a match is found or fewer than 2 components of the local domain remain. For example, in the domain CHI.Lachman.COM, the name flaime.STG will be checked first as flaime.STG.CHI.Lachman.COM and then as flaime.STG.Lachman.COM. Flaime.STG.COM will not be tried, as the there is only one component remaining from the local domain.

See Also

gethostent(SFF),resolver(ADMN), mailaddr(ADMN),
named(ADMN).
RFC883.

ifconfig

configure network interface parameters

Syntax

/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; it defines the network address of each interface present on a machine. *ifconfig* is run at system start-up time via *tcp(1M)*. *ifconfig* may be run at other times to redefine an interface's address or other operating parameters. (For example, *slattach* (ADMN) also runs *ifconfig* .)

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 a separate naming scheme, it is necessary to specify the address_family, which may change the interpretation of the remaining parameters. Currently, only the Internet address family is supported: thus, the only valid value for address_family is inet.

For the DARPA-Internet family, the address is either a host name or a DARPA Internet address expressed in the Internet standard "dot notation". (Host name translation is performed either by the name server or by an entry in **/etc/hosts.** [See *named*(ADMN) and *hosts*(ADMN).] Internet "dot notation" is described in *hosts*(ADMN) and *inet*(ADMP). Other address families may use different notations.)

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.

detach Remove an interface from the system. This command is applicable to transient interfaces only, such as serial line interfaces.

trailers Request the use of a trailer link level encapsulation when sending (default). If a network interface supports trailers, the system will, when possible, encapsulate outgoing messages in a manner that minimizes the number of memory-to-memory copy operations performed by the receiver. On networks that support the Address Resolution Protocol (see *arp*(ADMP); currently, only 10 Mb/s Ethernet), this flag indicates that the system should request that other systems use trailers when sending to this host. Similarly, trailer encapsulations will be sent to other hosts that have made such requests. This is currently used by Internet protocols only.

-trailers Disable the use of a trailer-link-level encapsulation.

arp Enable the use of the Address Resolution Protocol in mapping between network level addresses and link-level addresses (default). This is currently implemented for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses. This option is not applicable in the STREAMS environment. Use of arp for an interface is specified in **/etc/strcf**. The arp driver will be opened when the STREAMS stack is built.

-arp Disable the use of the Address Resolution Protocol.

metric *n* Set the routing metric of the interface to *n*, default 0. The routing metric is used by the routing protocol. 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 sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation Internet address, or with a pseudo-network name listed in the network table *networks(SFF)*. The mask contains 1's for the bit positions in the 32-bit address, which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion.

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.

onepacket Enable the *one-packet* mode of operation (used for interfaces that cannot handle back-to-back packets) The keyword **onepacket** must be followed by two numeric parameters, giving the small packet size and threshold, respectively. If small packet detection is not desired, these values should be zero. See *tcp*(ADMP) for an explanation on one-packet mode.

-onepacket Disable one-packet mode.

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.

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.

Files

/etc/slattach
calls *ifconfig* to start serial lines

See Also

arp(ADMN), tcp(ADMN), netstat(TC), hosts(SFF), networks(SFF), strcf(ADMN), arp(ADMP), tcp(ADMP).

inetd

internet super server

Syntax

/etc/inetd [**-d**] [configuration file]

Description

inetd listens on multiple ports for incoming connection requests. When it receives a request, it spawns the appropriate server. The use of a superserver allows other servers to be spawned only when needed and to terminate when they have satisfied a particular request.

The mechanism is as follows: *inetd* is started by the superuser (usually during init 2, if **/etc/tcp** is linked to **/etc/rc2.d/S*nn*tcp**.). To obtain information about the servers it needs to spawn, *inetd* reads its configuration file (by default, */etc/inetd.conf* (SFF)) and issues a call to *getservbyname* . [See *getservent* (SLIB).] (Note that **/etc/services** and **/etc/protocols** must be properly configured.) *inetd* then creates a socket for each server and binds each socket to the port for that server. It does a *listen* (SSC) on all connection-based sockets (that is, stream rather than datagram), and waits, using *select* (SSC), for a connection or datagram.

- When a connection request is received on a listening (stream) socket, *inetd* does an *accept* (SSC), thereby creating a new socket. (*inetd* continues to listen on the original socket for new requests). *inetd* forks, dups, and execs the appropriate server, passing it any server program arguments specified in *inetd*'s configuration file. The invoked server has I/O to **stdin**, **stdout**, and **stderr** done to the new socket; this connects the server to the client process. (Some built-in, internal services are performed via function calls rather than child processes.)

- When there is data waiting on a datagram socket, *inetd* forks, dups, and execs the appropriate server, passing it any server program arguments; unlike a connection-based server, a datagram server has I/O to **stdin**, **stdout**, and **stderr** done to the original socket. If the datagram socket is marked as *wait* (which corresponds to an entry in *inetd*'s configuration file), the invoked server must process the message before *inetd* considers the socket available for new connections. If the datagram socket is marked as *nowait*, *inetd* continues to process incoming messages on that port. *tftpd* is an exceptional case: although its entry in *inetd*'s configuration file must be *wait* (to avoid contention for the port), *inetd* is able to continue processing new messages on the port.

The following servers may be started by *inetd*: *fingerd*, *ftpd*, *rexecd*, *rlogind*, *rshd*, *telnetd*, and *tftpd*. *inet* must also start several internal services: these are described in *inetd.conf*(SFF). Do **not** arrange for *inetd* to start *rwhod*, or any NFS server.

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.

The **-d** option turns on socket-level debugging and prints debugging information to **stdout**.

Files

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

See Also

fingerd(ADMN), ftpd(ADMN), rexecd(ADMN), rlogind(ADMN), rshd(ADMN), telnetd(ADMN), tftpd(ADMN), inetd.conf(SFF), protocols(SFF), services(SFF).

ldsocket

load socket configuration

Syntax

ldsocket [**-v**] [**-c** file]

Description

ldsocket initializes the System V STREAMS TCP/IP Berkeley networking compatability interface, which is an alternate stream head supporting the *socket*(SSC) system call family. *ldsocket* loads the kernel with associations between the protocol family, type and number triplets passed to the *socket* system call, and the STREAMS devices supporting those protocols. *ldsocket* reads the file **/etc/sockcf** to obtain configuration information, and must be run before the Berkeley networking interface can be used.

The following options may be specified on the *ldsocket* command line:

-c *file* Use *file* instead of **/etc/sockcf**.

-v Use verbose mode (in which a message is written to **stderr** for each protocol loaded).

Files

/etc/sockcf

See Also

sockcf(SFF), intro(ADMP), socket(SSC).

lmail

handle local mail delivery from sendmail

Syntax

lmail user ...

Description

lmail interprets incoming mail received from *sendmail*(ADMN), and delivers it to the specified user on the local machine. It locks the user's mailbox using the *mail*(TC) locking mechanism.

See Also

mail(TC), sendmail(ADMN).

mailaddr

mail addressing description

Description

Mail addresses are based on the ARPANET protocol listed at the end of this manual page. These addresses are in the general format

user@domain

where a domain is a hierarchical dot separated list of subdomains. For example, the address

stevea@laiter.lachman.com

is normally interpreted from right to left: the message should go to the Lachman gateway, after which it should go to the local host laiter. When the message reaches laiter it is delivered to the user "stevea".

Unlike some other forms of addressing, this does not imply any routing. Thus, although this address is specified as an RFC822 address, it might travel by an alternate route if that were more convenient or efficient. For example, at Lachman, the associated message would probably go directly to laiter over the Ethernet rather than going via the Lachman mail gateway.

Abbreviation.

Under certain circumstances it may not be necessary to type the entire domain name. In general, anything following the first dot may be omitted if it is the same as the domain from which you are sending the message. For example, a user on "laisagna.Lachman.COM" could send to "stevea@laiter" without adding the "Lachman.COM" since it is the same on both sending and receiving hosts.

Certain other abbreviations may be permitted as special cases. For example, at Lachman, Internet hosts may be referenced without adding the "Lachman.COM" as long as their names do not conflict with a local host name.

Compatibility.

Certain old address formats are converted to the new format to provide compatibility with the previous mail system. In particular,

user@host.ARPA

is allowed and

host:user

is converted to

user@host

to be consistent with the *rcp*(1) command.

Also, the syntax

host!user

is converted to:

user@host.UUCP

This is normally converted back to the "host!user" form before being sent on for compatibility with older UUCP hosts.

The current implementation is not able to route messages automatically through the UUCP network. Until that time you must explicitly tell the mail system which hosts to send your message through to get to your final destination.

Case Distinctions.

Domain names (i.e., anything after the "@" sign) may be given in any mixture of upper and lower case with the exception of UUCP hostnames. Most hosts accept any combination of case in user names, with the notable exception of MULTICS sites.

Route-addrs.

Under some circumstances it may be necessary to route a message through several hosts to get it to the final destination. Normally this routing is done automatically, but sometimes it is desirable to route the message manually. Addresses which show these relays are termed "route-addrs." These use the syntax:

<@hosta,@hostb:user@hostc>

This specifies that the message should be sent to hosta, from there to hostb, and finally to hostc. This path is forced even if there is a more efficient path to hostc.

Route-addrs occur frequently on return addresses, since these are generally augmented by the software at each host. It is generally possible to ignore all but the "user@domain" part of the address to determine the actual sender.

Postmaster.

Every site is required to have a user or user alias designated "postmaster" to which problems with the mail system may be addressed.

Other Networks.

Some other networks can be reached by giving the name of the network as the last component of the domain. *This is not a standard feature* and may not be supported at all sites. For example, messages to CSNET or BITNET sites can often be sent to "user@host.CSNET" or "user@host.BITNET" respectively.

Bugs

The RFC822 group syntax ("group:user1,user2,user3;") is not supported except in the special case of "group:;" because of a conflict with old berknet-style addresses.

Route-Address syntax is ugly.

UUCP- and RFC822-style addresses do not coexist politely.

See Also

mailx(TC), sendmail(ADMN). RFC822.

mconnect

connect to SMTP mail server socket

Syntax

mconnect [**-p** *port*] [**-r**] [*hostname*]

Description

Mconnect opens a connection to the mail server on a given host, so that it can be tested independently of all other mail software. If no host is given, the connection is made to the local host. Servers expect to speak the Simple Mail Transfer Protocol (SMTP) on this connection. Exit by typing the "quit" command. Typing end-of-file will cause an end of file to be sent to the server. An interrupt closes the connection immediately and exits.

Options

-p Specify the port number instead of the default SMTP port (number 25) as the next argument.

-r "Raw" mode: disable the default line buffering and input handling. This gives you a similar effect as *telnet* to port number 25, not very useful.

Files

/usr/lib/sendmail.hf	Help file for SMTP commands

See Also

sendmail(ADMN).
RFC821.

mkhosts

make node name commands

Syntax

/etc/mkhosts

Description

mkhosts makes the simplified forms of the *rcmd*(TC) and *rlogin*(TC) commands. For each node listed in **/etc/hosts**, *mkhosts* creates a link to **/usr/bin/rcmd** in **/usr/hosts**. Each link's name is the same as the node's official name in **/etc/hosts**.

See Also

rcmd (TC), rlogin(TC).

named

Internet domain name server

Syntax

named [**-d** *debuglevel*] [**-p** *port#*] [**-b** *bootfile*]

Description

named is the Internet domain name server. (See RFC1035 for more details on the Internet name-domain system.) Without any arguments, *named* will read the default boot file **/etc/named.boot**, read any initial data, and listen for queries.

Options are:

-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**.

-b Use an alternate boot file. This is optional and allows you to specify a file with a leading dash.

Any additional argument is taken as the name of the boot file. The boot file contains information about where the name server is to get its initial data. If multiple boot files are specified, only the last is used. Lines in the boot file cannot be continued on subsequent lines. The following is a small example:

```
;
;
;           boot file for name server
;
directory   /usr/local/lib/named
; type      domain                  source host/file                backup file

cache       .                                                       root.cache
primary     Berkeley.EDU            berkeley.edu.zone
primary     32.128.IN-ADDR.ARPA     ucbhosts.rev
secondary   CC.Berkeley.EDU         128.32.137.8 128.32.137.3       cc.zone.bak
secondary   6.32.128.IN-ADDR.ARPA   128.32.137.8 128.32.137.3       cc.rev.bak
primary     0.0.127.IN-ADDR.ARPA                                    localhost.rev
forwarders  10.0.0.78 10.2.0.78
; slave
```

The "directory" line causes the server to change its working directory to the directory specified. This can be important for the correct processing of $INCLUDE files in primary zone files.

The "cache" line specifies that data in "root.cache" is to be placed in the backup cache. Its main use is to specify data such as locations of root domain servers. This cache is not used during normal operation, but is used as "hints" to find the current root servers. The file "root.cache" is in the same format as "berkeley.edu.zone". There can be more than one "cache" file specified. The cache files are processed in such a way as to preserve the time-to-live's of data dumped out. Data for the root nameservers is kept artificially valid if necessary.

The first "primary" line states that the file "berkeley.edu.zone" contains authoritative data for the "Berkeley.EDU" zone. The file "berkeley.edu.zone" contains data in the master file format described in RFC1035. All domain names are relative to the origin, in this case, "Berkeley.EDU" (see below for a more detailed description). The second "primary" line states that the file "ucbhosts.rev" contains authoritative data for the domain "32.128.IN-ADDR.ARPA," which is used to translate addresses in network 128.32 to hostnames. Each master file should begin with an SOA record for the zone (see below).

The first "secondary" line specifies that all authoritative data under "CC.Berkeley.EDU" is to be transferred from the name server at 128.32.137.8. If the transfer fails it will try 128.32.137.3 and continue trying the addresses, up to 10, listed on this line. The secondary copy is also authoritative for the specified domain. The first non-dotted-quad address on this line will be taken as a filename in which to backup the transferred zone. The name server will load the zone from this backup file if it exists when it boots, providing a complete copy even if the master servers are unreachable. Whenever a new copy of the domain is received by automatic zone transfer from one of the master servers, this file will be updated. The second "secondary" line states that the address-to-hostname mapping for the subnet 128.32.136 should be obtained from the same list of master servers as the previous zone.

The "forwarders" line specifies the addresses of sitewide servers that will accept recursive queries from other servers. If the boot file specifies one or more forwarders, then the server will send all queries for data not in the cache to the forwarders first. Each forwarder will be asked in turn until an answer is returned or the list is exhausted. If no answer is forthcoming from a forwarder, the server will continue as it would have without the forwarders line unless it is in "slave" mode. The forwarding facility is useful to cause a large sitewide cache to be generated on a master, and to reduce traffic over links to outside servers. It can also be used to allow servers to run that do not have access directly to the Internet, but wish to act as though they do.

The "slave" line (shown commented out) is used to put the server in slave mode. In this mode, the server will only make queries to forwarders. This option is normally used on machine that wish to run a server but for physical or administrative reasons cannot be given access to the Internet, but have access to a host that does have access.

The "sortlist" line can be used to indicate networks that are to be preferred over other, unlisted networks. Queries for host addresses from hosts on the same network as the server will receive responses with local network addresses listed first, then addresses on the sort list, then other addresses. This line is only acted on at initial startup. When reloading the nameserver with a SIGHUP, this line will be ignored.

The master file consists of control information and a list of resource records for objects in the zone of the forms:

$INCLUDE <filename> <opt_domain>
$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. *opt_domain* field is used to define an origin for the data in an included file. It is equivalent to placing a $ORIGIN statement before the first line of the included file. The field is optional. Neither the *opt_domain* field nor $ORIGIN statements in the included file modify the current origin for this file. The *opt_ttl* field is an optional integer number for the time-to-live field. It defaults to zero, meaning the minimum value specified in the SOA record for the zone. 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 contains 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)

CNAME the canonical name for an alias (domain)

SOA marks the start of a zone of authority (domain of originating host, domain address of maintainer, a serial number and the following parameters in seconds: refresh, retry, expire and minimum TTL (see RFC1035))

MB a mailbox domain name (domain)

MG a mail group member (domain)

MR a mail rename domain name (domain)

MX a mail exchange record

NULL a null resource record (no format or data)

WKS a well-known 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)

Resource records normally end at the end of a line, but may be continued across lines between opening and closing parentheses. Comments are introduced by semicolons and continue to the end of the line.

Each master zone file should begin with an SOA record for the zone. An example SOA record is as follows:

```
@   IN   SOA  ucbvax.Berkeley.EDU.  rwh.ucbvax.Berkeley.EDU. (
                        2.89   ; serial
                        10800; refresh
                        3600   ; retry
                        3600000    ; expire
                        86400 )    ; minimum
```

The SOA lists a serial number, which should be changed each time the master file is changed. Secondary servers check the serial number at intervals specified by the refresh time in seconds; if the serial number changes, a zone transfer will be done to load the new data. If a master server cannot be contacted when a refresh is due, the retry time specifies the interval at which refreshes should be attempted until successful. If a master server cannot be contacted within the interval given by the expire time, all data from the zone is discarded by secondary servers. The minimum value is the time-to-live used by records in the file with no explicit time-to-live value.

Notes

The boot file directives "domain" and "suffixes" have been obsoleted by a more useful resolver based implementation of suffixing for partially qualified domain names. The prior mechanisms could fail under a number of situations, especially when then local nameserver did not have complete information.

The following signals have the specified effect when sent to the server process using the *kill* (C) command.

SIGHUP Causes server to read named.boot and reload database.

SIGINT Dumps current data base and cache to **/usr/tmp/named_dump.db**.

SIGIOT Dumps statistics data into /usr/tmp/named.stats if the server is compiled -DSTATS. Statistics data is appended to the file.

SIGSYS Dumps the profiling data in /usr/tmp if the server is compiled with profiling (server forks, chdirs and exits).

SIGTERM Dumps the primary and secondary database files. Used to save modified data on shutdown if the server is compiled with dynamic updating enabled.

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
/usr/tmp/named.stats	nameserver statistics data

See Also

kill(C), gethostent(SLIB), signal(S), sigset(S), resolver(SFF), resolver(ADMN), hostname(ADMP).
RFC974, RFC1034, RFC1035, *Name Server Operations Guide for BIND*.

netlogin

network login program

Syntax

netlogin [**-p**] [**-r** remotehost] [name] [env-var]

Description

Netlogin is a derivative of the *login*(TC) command. It provides facilities that *telnetd*(ADMN) and *rlogind(ADMN)* need, such as preserving the environment, and support for automatically logging users in. *Netlogin* takes the following options:

-p Preserve the environment. This is used by *telnetd* to pass information obtained via terminal type negotiation.

-r *remotehost*
Process automatic login from *remotehost.* Used by *rlogind* to allow a user with the proper permissions to bypass the password prompt when logging in.

See Also

login(TC), rlogind(ADMN), telnetd(ADMN), rhosts(SFF).

ping

send ICMP ECHO_REQUEST packets to network hosts

Syntax

/etc/ping [**-r**] [**-v**] host [packetsize] [count]

Description

ping is a troubleshooting tool for tracking a single-point hardware or software failure in the Internet. It uses 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 an ICMP header, followed by a **struct timeval** and 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.

-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. The *ping* tool 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 are out (with a *count* specified), or if the program is terminated with a SIGINT, then 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

netstat(TC), ifconfig(ADMN).

rdate

notify time server that date has changed

Syntax

rdate

Description

rdate notifies *timed*(ADMN) that the system date has changed. If the local time server is a master, it will notify all of the slaves that the time has changed. If it is a slave, it will ask the master to update the time.

rdate should be run whenever the super user sets the date with *date(C)*. A shell script such as the following could be used to do both automatically.

```
:
: mv /bin/date /bin/s5date
: install as /bin/date
:
PATH=/bin:/usr/bin
s5date $*
rdate
```

See Also

date(C), adjtime(SSC), gettimeofday(SLIB), icmp(ADMP), timed(ADMN), timedc(ADMN).

rexecd

remote execution server

Syntax

/etc/rexecd

Description

rexecd is the server for the *rexec* (SLIB) 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* (SFF). When a service request is received, the following protocol is initiated:

1) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is non-zero, 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, unencrypted 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) Then, *rexecd* 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 with a diagnostic message 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*.

rexecd is started by the super-server *inetd*, and therefore must have an entry in *inetd*' s configuration file **/etc/inetd.conf**.

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 password 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.

See Also

rexec(SLIB), inetd(ADMN), inetd.conf(SFF), services(SFF).

Notes

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.

rlogind

remote login server

Syntax

/etc/rlogind

Description

rlogind is a network server that supports remote logins by programs such as *rlogin*(TC). It is started by the superserver *inetd* and, therefore, must have an entry in *inetd*'s configuration file **/etc/inetd.conf.** [See *inetd*(ADMN) and *inetd.conf*(SFF).]

rlogind enforces an authentication procedure based on equivalence of user names (see *rhosts*(SFF)). This procedure assumes all hosts on the network are equally secure.

See Also

inetd(ADMN), rlogin(TC), inetd.conf(SFF), rhosts(SFF), services(SFF).

rmail

handle remote mail received via uucp

Syntax

rmail user ...

Description

rmail interprets incoming mail received via *uucp*(C), collapsing "From" lines in the form generated by *mail*(TC) into a single line of the form *return-path!sender*, and passing the processed mail on to *sendmail*(ADMN).

rmail is explicitly designed for use with *uucp* and *sendmail*.

See Also

mail(TC), uucp(C), sendmail(ADMN).

route

manually manipulate the routing tables

Syntax

/etc/route [**-f**] [**-n**] [command destination gateway [metric]]

Description

route is a program used to manipulate manually the network routing tables. It is normally not needed, since the routing daemon *routed* manages the system routing table and therefore handles this function.

route accepts two commands: *add*, to add a route; and *delete*, to delete a route.

All commands have the following syntax:

/etc/route *command destination gateway* [*metric*]

where *destination* is a host or network for which the route is "to", *gateway* is the gateway to which packets should be addressed, and *metric* is an optional count indicating the number of hops to the *destination*. If no metric is specified, *route* assumes a value of 0. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. If the *destination* has a local address part of INADDR_ANY, the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. Note: If the route is to a destination connected via a gateway, *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first in the host-name database; see *hosts(SFF)*. If this lookup fails, the name is then looked for in the network name database; see *networks(SFF)*.

route uses a raw socket and the SIOCADDRT and SIOCDELRT *ioctl*'s to do its work. Therefore, only the super user 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.

Diagnostics

add [host | network]
The specified route is being added to the tables. The values printed are from the routing table entry supplied in the *ioctl* call.

"delete *host*: **gateway** *host* **flags** *hex-flags*"
As above, but when deleting an entry.

"*host host* **done**"
When the **-f** flag is specified, each routing table entry deleted is indicated with a message of this form.

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

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

See Also

routed(ADMN), intro(ADMN), hosts(SFF), networks(SFF).

routed

network routing daemon

Syntax

/etc/routed [**-d**] [**-g**] [**-s**] [**-t**] [logfile]

Description

routed manages the Internet routing tables using a variant of the Xerox NS Routing Information Protocol. *routed* is invoked by the superuser (usually during init 2).

In normal operation, *routed* listens on the *udp(ADMP)* socket for the *route* service (see *services(SFF))* for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When *routed* is started, it uses the SIOCGIFCONF *ioctl* to find those directly connected interfaces configured into the system and marked "up". (The software loopback interface is ignored.) If multiple interfaces are present, it is assumed that the host will forward packets between networks. Then, *routed* transmits a request packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for request and response packets from other hosts.

When a request packet is received, *routed* formulates a reply based on the information maintained in its internal tables. The response packet generated contains a list of known routes, each marked with a *hop count* metric. (A count of 16 or greater is considered infinite.) The metric associated with each route returned provides a metric relative to the sender.

Response packets received by *routed* are used to update the routing tables if one of the following conditions is satisfied:

(1)
No routing table entry exists for the destination network or host, and the metric indicates the destination is reachable (that is, the hop count is not infinite).

(2)
The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.

(3)
The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.

(4)
The new route describes a shorter path to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, *routed* records the change in its internal tables and updates the kernel-routing table. The change is reflected in the next response packet sent.

In addition to processing incoming packets, *routed* also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, its metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to ensure that the invalidation is propagated throughout the local internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly-connected hosts and networks. The response is sent to the broadcast address on nets capable of the broadcast function, to the destination address on point-to-point links, and to the router's own address on other networks. The normal routing tables are bypassed when sending gratuitous responses. The reception of responses on each network is used to determine that the network and interface are functioning correctly. If no response is received on an interface, another route may be chosen to route around the interface, or the route may be dropped if no alternative is available.

routed supports several options:

-d Enable additional debugging information to be logged, such as bad packets received.

-g This flag is used on internetwork routers to offer a route to the default destination. This is typically used on a gateway to the Internet, or on a gateway that uses another routing protocol whose routes are not reported to other local routers.

-s Supplying this option forces *routed* to supply routing information whether it is acting as an internetwork router or not. This is the default if multiple network interfaces are present, or if a point-to-point link is in use.

-q This is the opposite of the **-s** option.

-t If the **-t** option is specified, all packets sent or received are printed on the standard output. In addition, *routed* will not divorce itself from the controlling terminal, and so interrupts from the keyboard will kill the process.

Any other argument supplied is interpreted as the name of file in which *routed*'s actions should be logged. This log contains information about any changes to the routing tables and, if the log is not tracing all packets, a history of recent messages sent and received that are related to the changed route.

In addition to the facilities described above, *routed* supports the notion of distant passive and active gateways. When *routed* is started up, it reads the file **/etc/gateways** to find gateways that may not be located using only information from the SIOCGIFCONF *ioctl*. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (that is, they should have a *routed* process running on the machine). Passive gateways are maintained in the routing tables forever, and information regarding their existence is included in any routing information transmitted. Active gateways are treated equally with network interfaces. Routing information is distributed to the gateway and, if no routing information is received for a period of time, the associated route is deleted. External gateways are also passive, but are not placed in the kernel routing table nor are they included in routing updates. The function of external entries is to inform *routed* that another routing process will install such a route, and that alternate routes to that destination should not be installed. Such entries are only required when both routers may learn of routes to the same destination.

The **/etc/gateways** is comprised of a series of lines, each in the following format:

<net | host> *name1* **gateway** *name2* **metric** *value* **<passive | active | external>**

The **net** or **host** keyword indicates whether the route is to a network or specific host.

name1 is the name of the destination network or host. This may be a symbolic name located in **/etc/networks** or **/etc/hosts** (or, if started after *named(ADMN)*, known to the name server), or an Internet address specified in "dot" notation; see *hosts(SFF)* and *inet(ADMP)*.

name2 is the name or address of the gateway to which messages should be forwarded.

value is a metric indicating the hop count to the destination host or network.

One of the keywords **passive, active** and **external** indicates whether the gateway should be treated as passive or active (as described above), or the gateway is external to the scope of the *routed* protocol.

Files

/etc/gateways for distant gateways

See Also

udp(ADMP).

Notes

The kernel's ICMP routing tables may not correspond to those of *routed* when ICMP redirects change or add routes.

rshd

remote shell server

Syntax

/etc/rshd

Description

rshd is the network server for programs such as *rcmd*(TC) and *rcp*(TC) which need to execute a noninteractive shell on remote machines. *rshd* is started by the superserver *inetd*, and therefore must have an entry in *inetd*'s configuration file **/etc/inetd.conf**. [See *inetd*(ADMN) and *inetd.conf*(SFF)].

rshd enforces an authentication procedure based on equivalence of user names (see *rhosts*(SFF)). This procedure assumes all nodes on the network are equally secure.

See Also

inetd(ADMN), rcmd(TC), rcp(TC), inetd.conf(SFF), rhosts(SFF).

rwhod

system status server

Syntax

/etc/rwhod

Description

rwhod is the server which maintains the database used by the *rwho*(TC) and *ruptime*(TC) programs. Its operation is predicated on the ability to broadcast messages on a network.

rwhod operates as both a producer and a 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*(SFF). 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_fill[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 are as calculated by the *uptime*(C) program, and 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 the *gethostname*(SLIB) system call, 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*(M) 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's 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 5 minutes. *rwhod* performs an *nlist*(S) on **/unix** every 30 minutes to guard against the possibility that this file is not the system image currently operating.

See Also

rwho(TC), ruptime(TC).

Notes

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.

Some mechanism for cleaning dead machine data out of the spool directory is needed.

sendmail

send mail over the internet

Syntax

/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; for instance, 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. Every input line must end with a CR-LF, and each message 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. *sendmail* will fork and run in background listening on TCP port 25 for incoming SMTP connections. This is normally run from */etc/rc*.

-bi Initialize the alias database. This works only if *sendmail* was built with a DBM library. Otherwise, this option does nothing.

-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 (that is, 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 Don't do aliasing.

-o*x value* Set option *x* to the specified *value*. Options are described below.

-q[*time*] Process 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 and 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, and so on.

There is 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 *TCP/IP Administrator's 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 queueing.

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 - that is, actual delivery is done the next time the queue is run.

D Try to rebuild the alias database automatically 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 (so that 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 mode '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.

F*mode* The mode to use when creating temporary files.

f Save UNIX-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.

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 (that is, 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, so 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 (for instance, because a host is down) for this amount of time, failed messages will be returned to the sender. The default is three 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 which to pipe the mail. It may be necessary to quote the name to keep *sendmail* from suppressing the blanks 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.

The *sendmail* command 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	Catchall, meaning 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. This works only if *sendmail* was built with a DBM library. Otherwise, this option does nothing. If invoked as *mailq, sendmail* will print the contents of the mail queue.

Files

Except for **/usr/lib/sendmail.cf**, these pathnames are all specified in **/usr/lib/sendmail.cf**. Thus, these values are only approximations.

/usr/lib/aliases	raw data for 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

mail(TC), aliases(SFF), mailaddr(SFF);
RFC819, RFC821, RFC822;
The chapter "Introduction to sendmail" in the *TCP/IP Administrator's Guide*;
The chapter "Installing and Operating Sendmail" in the *TCP/IP Administrator's Guide*.

slattach, sldetach

attach and detach serial lines as network interfaces

Syntax

/etc/slattach devname source destination [baudrate]

/etc/sldetach interface-name

Description

slattach is used to assign a serial (tty) line to a network interface using the DARPA Internet Protocol, and to define the source and destination network addresses. The *devname* parameter is the name of the device the serial line is attached to, that is, /dev/tty001. The source and destination are either host names present in the host name data base (see *hosts(SFF)*), or DARPA Internet addresses expressed in the Internet standard "dot notation." The optional *baudrate* parameter is used to set the speed of the connection; if not specified, the default of 9600 is used.

Only the superuser may attach or detach a network interface.

There should not be a *getty*(M) on the line.

sldetach is used to remove the serial line that is being used for IP from the network tables and allow it to be used as a normal terminal again. *interface-name* is the name that is shown by *netstat*(TC).

Examples

```
/etc/slattach tty001 tom-src genstar
/etc/slattach /dev/tty001 hugo dahl 4800
/etc/sldetach sl01
```

Files

/etc/hosts
/dev/*
/usr/spool/locks/slippid.*

Diagnostics

Various messages indicating:
- the specified interface does not exist
- the requested address is unknown
- the user is not the superuser

See Also

hosts(SFF), netstat(TC), ifconfig(ADMN).

slink

streams linker

Syntax

slink [**-v**] [**-f**] [**-c** file] [func [arg1 arg2 ...]]

Description

slink is a STREAMS configuration utility that is used to link together the various STREAMS modules and drivers required for STREAMS TCP/IP. Input to *slink* is in the form of a script specifying the STREAMS operations to be performed. Input is normally taken from the file **/etc/strcf**.

The following options may be specified on the *slink* command line:

-c *file* Use *file* instead of **/etc/strcf**.

-v Verbose mode (that is, each operation is logged to **stderr**).

-f Do not fork (that is, *slink* will remain in foreground).

The configuration file contains a list of functions, each of which is composed of a list of commands. Each command is a call to one of the functions defined in the configuration file or to one of a set of built-in functions. Among the built-in functions are the basic STREAMS operations *open*, *link*, and *push*, along with several TCP/IP-specific functions.

slink processing consists of parsing the input file, then calling the user-defined function **boot**, which is normally used to set up the standard configuration at boot time. If a function is specified on the *slink* command line, that function will be called instead of **boot**. Following the execution of the specified function, *slink* goes into the background and remains idle, holding open whatever file descriptors have been opened by the configuration commands.

A function definition has the following form:

```
function-name {
        command1
        command2
        ...
}
```

The syntax for commands is:

```
function arg1 arg2 arg3 ...
```

or:

```
var = function arg1 arg2 arg3 ...
```

The placement of newlines is important: a newline must follow the left and right braces and every command. Extra newlines are allowed, that is, where one newline is required, more than one may be used. A backslash ('\') followed immediately by a newline is considered equivalent to a space, so it may be used to continue a command on a new line. The use of other white space characters (spaces and tabs) is at the discretion of the user, except that there must be white space separating the function name and the arguments of a command.

Comments are delimited by '#' and newline, and are considered equivalent to a newline.

Function and variable names may be any string of characters taken from A-Z, a-z, 0-9, and '_', except that the first character cannot be a digit. Function names and variable names occupy separate name spaces. All functions are global and may be forward-referenced. All variables are local to the functions in which they occur.

Variables are defined when they appear to the left of an equal sign ('=') on a command line, such as:

```
tcp = open /dev/inet/tcp
```

The variable acquires the value returned by the command. In the above example, the value of the variable *tcp* will be the file descriptor returned by the *open* call.

Arguments to a command may be variables, parameters, or strings.

A variable that appears as an argument must have been assigned a value on a previous command line in that function.

Parameters take the form of a dollar sign ('$') followed by one or two decimal digits, and are replaced with the corresponding argument from the function call. If a given parameter was not specified in the function call, an error results (for instance, if a command references $3 and only two arguments were passed to the function, an execution error will occur).

Strings are sequences of characters optionally enclosed in double quotes ('"'). Quotes may be used to prevent a string from being interpreted as a variable name or a parameter, and to allow the inclusion of spaces, tabs, and the special characters '{', '}', '=', and '#'. The backslash ('\') may also be used to quote the characters '{', '}', '=', '#', '"', and '\' individually.

The following built-in functions are provided by *slink*:

open *path*
: Open the device specified by pathname *path*. Returns a file descriptor referencing the open stream.

link *fd1 fd2*
: Link the stream referenced by *fd2* beneath the stream referenced by *fd1*. Returns the link identifier associated with the link. *Note*: The *fd2* function cannot be used after this operation.

push *fd module*
: Push the module identified by *module* onto the stream referenced by *fd*.

sifname *fd link name*
: Send a SIOCSIFNAME (set interface name) ioctl down the stream referenced by *fd* for the link associated with link identifier *link* specifying the name given in *name*.

unitsel *fd unit*
: Send a IF_UNITSEL (unit select) ioctl down the stream referenced by *fd* specifying the unit given in *unit*.

dlattach *fd unit*
: Send a DL_ATTACH_REQ message down the stream referenced by *fd* specifying the unit given in *unit*.

initqp *path qname lowat hiwat ...*
: Send an INITQPARMS (initialize queue parameters) ioctl to the driver corresponding to pathname *path*. *qname* specifies the queue for which the low and high water marks will be set, and must be one of:

hd	stream head
rq	read queue
wq	write queue
muxrq	multiplexor read queue
muxwq	multiplexor write queue

The *lowat* and *hiwat* functions specify the new low and high water marks for the queue. Both *lowat* and *hiwat* must be present. To change only one of these parameters, the other may be replaced with a dash ('-'). Up to five *qname lowat hiwat* triplets may be present.

strcat *str1 str2*	Concatenate strings *str1* and *str2* and return the resulting string.
return *val*	Set the return value for the current function to *val*. *Note*: executing a **return** command does not terminate execution of the current function.

Files

/etc/strcf

See Also

strcf(SFF), intro(ADMP).

talkd

remote user communication server

Syntax

/etc/talkd

Description

Talkd is the server that notifies a user that somebody else wants to initiate a conversation. It acts as a repository of invitations, responding to requests by clients wishing to rendezvous to hold a conversation. In normal operation, a *talk* client initiates a 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 client. If the lookup fails, the caller then sends an ANNOUNCE message causing the server to broadcast an announcement on the callee's login ports requesting contact. When the callee responds, the local server uses the recorded invitation to respond with the appropriate rendezvous address and the caller and callee client programs establish a stream connection through which the conversation takes place.

See Also

talk(TC), write(TC)

/etc/tcp

TCP start/stop script

Syntax

/etc/tcp start
/etc/tcp stop

Description

/etc/tcp is used to start or stop the STREAMS TCP software. TCP will start automatically at system startup time if **/etc/rc.d/6/***name* contains a script including the command **/etc/tcp start**. TCP does not stop automatically at system shutdown time. The command **/etc/tcp stop** will stop TCP. See *init*(M) for further information.

/etc/tcp must be customized for a particular installation before it can be used. The following items must be edited:

Domain name	The environment variable *DOMAIN* must be set to the name of your domain.
Interface configuration	*ifconfig* commands must be used to set the internet address (and any other desired options) for each of your interfaces. The *ifconfig* line for the loopback interface should not require modification. See *ifconfig*(ADMN) for further information.

The following items may need to be edited:

PATH	The supplied path may require modification if commands run by */etc/tcp* are in other directories.
PROCS	The *PROCS* variable contains a space-separated list of names of processes to kill when executing the **stop** function. If additional daemons are used, their names can be added to this list.
Network initialization	Certain network hardware may require the execution of an initialization command before use. Any such commands should be included in this section.

Daemons

The standard internetworking daemons are started at this point. Any additional daemons or other commands may be included in this section. Any of the standard daemons that are not desired may be removed or commented out.

telnetd

DARPA TELNET protocol server

Syntax

/etc/telnetd

Description

telnetd is a server that supports the DARPA standard **TELNET** virtual terminal protocol. *telnetd* is invoked by the internet server (see *inetd*(ADMN)), normally for requests to connect to the **TELNET** port as indicated by the **/etc/services** file (see *services*(SFF)).

telnetd operates by allocating a pseudo-terminal device for a client, then creating a login process that 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 a willingness 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 ICANON mode, and with TAB3 and ICRNL enabled. (See *termio*(M).)

telnetd is willing to do: *echo*, *binary*, *suppress go ahead*, and *timing mark*. *telnetd* is willing to have the remote client do: *binary*, *terminal type*, and *suppress go ahead*.

See Also

telnet(TC)

Notes

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 4.2 BSD *telnet*, *telnetd* performs some dubious protocol exchanges to try to discover if the remote client is, in fact, a 4.2 BSD *telnet*.

Binary mode has no common interpretation except between similar operating systems (Unix, in this case).

The terminal type name received from the remote client is converted to lowercase.

The packet interface to the pseudo terminal should be implemented for intelligent flushing of input and output queues.

telnetd never sends **TELNET** *go ahead* commands.

tftpd

DARPA Trivial File Transfer Protocol server

Syntax

/etc/tftpd

Description

tftpd is a server that supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the *tftp* service description; see *services* (SFF). This port number may be overridden (for debugging purposes) by specifying a port number on the command line.

The use of *tftp* 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. 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.

tftpd is spawned by the superserver *inetd* and, therefore, must have an entry in *inetd*'s configuration file, **/etc/inetd.conf**. [See *inetd* (ADMN) and *inetd.conf* (SFF).] Note that the *tftpd* entry in this file must be "wait": this is to prevent subsequent *selects* from being successful before the first *tftpd* process does its *receive*. *tftpd* takes care to prevent multiple *tftpd* processes from being spawned to service the same request. (*inetd* is able to continue processing new messages on the port.)

See Also

inetd(ADMN), tftp(TC), inetd.conf(SFF), services(SFF).

Warnings

This server is known only to be self-consistent (that is, it operates with the user TFTP program *tftp* (TC)).

The search permissions of the directories leading to the files accessed are not checked if *tftp* runs as root. The default configuration runs *tftpd* as user "sync."

timed

time server daemon

Syntax

/etc/timed [-t] [-M] [-n network] [-i network]

Description

timed is the time server daemon and is normally invoked at boot time from the STREAMS TCP/IP start-up script. It synchronizes the host's time with that of other machines in a local area network running *timed*(ADMN). 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*(ADMN) 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*(SSC) to perform the needed corrections on the host's clock.

It also communicates with *rdate*(ADMN) in order to set the date globally, and with *timedc*(ADMN), 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(ADMN)*. *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 to which the host is connected (see *networks(SFF)*), 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(SFF)*), 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.

Files

/usr/adm/timed.log	tracing file for timed
/usr/adm/timed.masterlog	log file for master timed

See Also

date(C), adjtime(SSC), gettimeofday(SLIB), icmp(ADMP), rdate(ADMN), timedc(ADMN).

timedc

timed control program

Syntax

timedc [command [argument ...]]

Description

timedc is used to control the operation of the *timed* 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* 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

date(C), adjtime(SSC), icmp(ADMP), rdate(ADMN), timed(ADMN).

Diagnostics

?Ambiguous command	abbreviation matches more than one command
?Invalid command	no match found
?Privileged command	command can be executed by root only

trace, query

routing tools

Syntax

trace [**on**|**off**] machines... **query** [**-n**] hosts...

Description

trace sends a RIP_TRACE_ON or RIP_TRACE_OFF command to the specified machines. *Machine must be specified as an IP address*.

query is used to request routing information from the specified host. Any packets received in response to a query will be displayed.

These commands are useful for debugging *routed(ADMN)*.

See Also

routed(ADMN), udp(ADMP).
RFC1058

Bugs

RFC 1058 states that TRACE_ON and TRACE_OFF are not supposed to be supported any more.

trpt

transliterate protocol trace

Syntax

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*(SSC)), 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*(TC). 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

getsockopt(SSC), netstat(TC)

Diagnostics

The message "no namelist" when the system image doesn't contain the proper symbols to find the trace buffer; other messages 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.

Contents

intro

introduction to special files and protocols

#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_str.h>
#include <netinet/strioc.h>

Description

This section describes various special files and protocols that refer to specific System V STREAMS TCP/IP networking protocol drivers. Features common to a set of protocols are documented as a protocol family.

Protocol Family Entries

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*(2) 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**(2). A specific protocol may be accessed by creating a socket of the appropriate type and protocol family, by requesting the protocol explicitly when creating a socket, by executing the appropriate TLI primitives, or by opening the associated STREAMS device.

Protocol Entries

The system currently supports the DARPA Internet protocols. Raw socket interfaces are provided to the IP protocol layer of the DARPA Internet and to the ICMP protocol. Consult the appropriate manual pages in this section for more information.

Routing Ioctls

The network facilities provided 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 data base with the aid of two socket-specific *ioctl*(2) commands, SIOCADDRT and SIOCDELRT. The commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by super-user.

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 as follows:

```
#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) */
```

Routing table entries are of three general types: those for a specific host, those for all hosts on a specific network, and those 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 (that is, the packet is forwarded).

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (*rt_refcnt* is non-zero), the 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 ENOSR 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.

Socket Ioctls

There are a few *ioctls* which have significance for the socket layer only. The *ioctl* call has the general form:

```
ioctl(so, code, arg)
```

SIOCPROTO
Enter a socket type into the kernel protocol switch table. The arguments used to create the socket used by this *ioctl* may be zero. The new socket type is downloaded by setting *arg* to a pointer to a specification block with the following structure:

```
struct socknewproto {
        int         family; /* address family (AF_INET, etc.) */
        int         type;   /* protocol type
                               (SOCK_STREAM, etc.) */
        int         proto;  /* per family proto number */
        dev_t       dev;    /* major/minor to use
                               (must be a clone) */
        int         flags;  /* protosw flags */
};
```

The flags currently supported are specified in the *<net/protosw.h>* header file as:

```
        /* exchange atomic messages only */
#define PR_ATOMIC       0x01
        /* addresses given with messages */
#define PR_ADDR         0x02
        /* connection required by protocol */
#define PR_CONNREQUIRED 0x04
#define PR_RIGHTS       0x10    /* passes capabilities */
#define PR_BINDPROTO    0x20    /* pass protocol */
```

SIOCXPROTO
Purge the protocol switch table. The arguments used to create the socket used by this *ioctl* may be zero.

SIOCSPGRP
Set the process group for a socket to enable signaling (SIGUSR1) of that process group when out-of-band data arrives. The argument, *arg*, is a pointer to an **int** and, if positive, is treated as a process ID; otherwise, (if negative) is treated as a process group ID.

SIOCGPGRP
Get the process group ID associated with a particular socket. If the value returned to the **int** location pointed to by *arg* is negative, it should be interpreted as a process group ID; otherwise, it should be interpreted as a process ID.

SIOCCATMARK
Used to ascertain whether or not the socket read pointer is currently at the point (mark) in the data stream where out-of-band data was sent. If a 1 is returned to the *int* location pointed to by *arg*, the next read will return data after the mark. Otherwise (assuming out-of-band data has arrived), the next read will provide data sent by the client prior to transmission of the out-of-band signal.

FIONREAD
Returns (to the **int** location pointed to by *arg*) the number of bytes currently waiting to be read on the socket.

FIONBIO
Toggles the socket into blocking/non-blocking mode. If the **int** location pointed to by *arg* contains a non-zero value, subsequent socket operations that would cause the process to block waiting on a specific event will return abnormally with *errno* set to EWOULDBLOCK; otherwise, the process will block.

Queue Ioctls

Each STREAMS device has default queue high and low water marks, that can be changed by the super-user with the INITQPARMS specification in an **ioctl**(2). The ioctl is done on a driver or module, with the argument being an array of structures of type:

```
struct iocqp {
        ushort iqp_type;
        ushort iqp_value;
}
```

iqp_value specifies the value for the queue parameter according to **iqp_type**, which may be one of: **IQP_RQ**(read queue), **IQP_WQ**(write queue), **IQP_MUXRQ**(mux read queue), **IQP_MUXWQ**(mux write queue), or **IQP_HDRQ**(stream head queue), each OR'ed with either **IQP_LOWAT**(value is for low water mark of queue), or **IQP_HIWAT**(value is for high water mark of queue).

Interface Ioctls

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, although 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 "communications domain" [see *protocols*(4)]. Unless specified otherwise, the request takes an *ifrequest* structure as its parameter. This structure has the form

```
struct  ifreq {
        char    ifr_name[16]; /* name of interface (e.g. ec0) */
        union {
                struct  sockaddr ifru_addr;
                struct  sockaddr ifru_dstaddr;
                struct  sockaddr ifru_broadaddr;
                short   ifru_flags;
                int     ifru_metric;
                struct onepacket ifru_onepacket;
        } ifr_ifru;
#define ifr_addr        ifr_ifru.ifru_addr        /* address */
                /* other end of p-to-p link */
#define ifr_dstaddr     ifr_ifru.ifru_dstaddr
                /* broadcast address */
#define ifr_broadaddr   ifr_ifru.ifru_broadaddr
#define ifr_flags       ifr_ifru.ifru_flags       /* flags */
                /* routing metric */
#define ifr_metric      ifr_ifru.ifru_metric
                /* one-packet mode params */
#define ifr_onepacket   ifr_ifru.ifru_onepacket
};
```

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.

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.

SIOCSIFONEP
Set one-packet mode parameters. The *ifr_onepacket* field of the *ifreq* structure is used for this request. This structure is defined as follows:

```
struct onepacket {
        int     spsize;     /* small packet size */
        int     spthresh;   /* small packet threshold */
};
```

One-packet mode is enabled by setting the IFF_ONEPACKET flag (see SIOCSIFFLAGS above). See *tcp*(7) for an explanation of one-packet mode.

SIOCGIFONEP
Get one-packet mode parameters.

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.

```
    /* Structure used in SIOCGIFCONF request.
     * Used to retrieve interface configuration
     * for machine (useful for programs which
     * must know all networks accessible).
     */
struct  ifconf {
        /* size of associated buffer */
        int     ifc_len;
```

```
        union {
                caddr_t ifcu_buf;
                struct  ifreq *ifcu_req;
        } ifc_ifcu;
/* buffer address */
#define ifc_buf ifc_ifcu.ifcu_buf
/* array of structures returned */
#define ifc_req ifc_ifcu.ifcu_req
};
```

Streams Ioctl Interface

Socket *ioctl* calls can also be issued using STREAMS file descriptors. The standard *strioctl* structure is used, with the *ic_cmd* field containing the socket *ioctl* code (from **<sys/socket.h>**) and the *ic_db* field pointing to the data structure appropriate for that *ioctl*, for all socket *ioctl*s except SIOCGIFCONF. For the SIOCGIFCONF *ioctl*, an *ifconf* structure is not used. Rather, the *ic_db* field points to the buffer to receive the *ifreq* structures.

TLI Options Management

Options may be set and retrieved in a manner similar to *getsockopt* (2) and *setsockopt* (2) using *t_optmgmt* (3N). Options are communicated using an options buffer, which contains a list of options. Each option consists of an option header and an option value. The *opthdr* structure gives the format of the option header:

```
struct opthdr {
        long level;     /* protocol level affected */
        long name;      /* option to modify */
        long len;       /* length of option value (in bytes) */
};
```

The option value must be a multiple of sizeof(long) bytes in length, and must immediately follow the option header. Following the option value is the header of the next option, if present.

To get the values of options, set the *flags* field of the *t_optmgmt* structure to T_CHECK. It is not necessary to set the *len* fields in the option headers to the expected lengths of the option values, nor is it necessary to provide space between option headers for the option values to be stored (the *len* fields should be set to zero and the option headers should be adjacent). A new options buffer will be formatted and returned to the user. Note that T_CHECK may have failed even if *t_optmgmt* returns zero. The user must check the *flags* field of the returned *t_optmgmt* structure. If this field contains T_FAILURE, one or more of the options were invalid.

To set options, set the *flags* field of the *t_optmgmt* structure to T_NEGOTIATE.

To retrieve the default values of all options, set the *flags* field of the *t_optmgmt* structure to T_DEFAULT. For this operation, no input buffer should be specified.

Note

System V STREAMS TCP/IP man pages frequently cite appropriate RFCs (Requests for Comments). RFCs can be obtained from the DDN Network Information Center, SRI International, Menlo Park, CA 94025.

See Also

ioctl(SSC), socket(SSC), t_optmgmt(NSL), tcp(ADMP).

arp

Address Resolution Protocol

Description

ARP is a protocol used to map dynamically between DARPA Internet and 10Mb/s Ethernet addresses. It is used by all the 10Mb/s Ethernet interface drivers running the Internet protocols.

ARP caches Internet-Ethernet address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message which 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 answered; only the most recently "transmitted" packet is kept. The ARP protocol is implemented by a STREAMS driver to do the protocol negotiation, and by a separate STREAMS module to do the address translation.

To facilitate communications with systems that do not use ARP, *ioctl* s are provided to enter and delete entries in the Internet-to-Ethernet tables. Usage:

```
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
struct arpreq arpreq;

ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```

Each *ioctl* takes the same structure as an argument. SIOCSARP sets an ARP entry, SIOCGARP gets an ARP entry, and SIOCDARP deletes an ARP entry. These *ioctls* may be applied to any socket descriptor *s*, but only by the superuser. The *arpreq* structure is as follows:

```
   /* ARP ioctl request */
struct arpreq {
   struct sockaddr      arp_pa;/* protocol address */
   struct sockaddr      arp_ha;/* hardware address */
   int                  arp_flags;/* flags */
};
   /*  arp_flags field values */
#defineATF_COM       0x02/* completed entry
                            (arp_ha valid) */
#defineATF_PERM      0x04/* permanent entry */
```

```
#defineATF_PUBL      0x08/* publish
                          (respond for other host) */
#defineATF_USETRAILERS0x10/* send trailer packets
                          to host */
```

The address family for the *arp_pa sockaddr* , must be AF_INET; for the *arp_ha sockaddr* it must be AF_UNSPEC. The only flag bits which may be written are ATF_PERM, ATF_PUBL and ATF_USETRAILERS. 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 8 (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.

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

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

Diagnostics

duplicate IP address!! sent from ethernet address: %x:%x:%x:%x:%x:%x. ARP has discovered another host on the local network that responds to mapping requests for its own Internet address.

Files

/dev/inet/arp

See Also

arp(ADMP), ifconfig(ADMN), inet(ADMP).

e3A

3C501 Ethernet Driver

Description

The e3A driver provides an LLI interface to a 3Com 3C501 ethernet card. As with other network interfaces, e3A interface must have network addresses assigned for each address family with which it is to be used. (Currently, only the Internet address family is supported.) These addresses may be set or changed with the SIOCSIFADDR ioctl.

Files

/dev/e3A[0-3]

See Also

intro(ADMP), inet(ADMP).

e3B

3C503 Ethernet Driver

Description

The e3B driver provides an LLI interface to a 3Com 3C503 ethernet card. As with other network interfaces, e3B interface must have network addresses assigned for each address family with which it is to be used. (Currently, only the Internet address family is supported.) These addresses may be set or changed with the SIOCSIFADDR ioctl.

Files

/dev/e3B[0-3]

See Also

intro(ADMP), inet(ADMP).

eli

EMD convergence module

Description

Eli acts as a convergence module between the EMD Ethernet Driver, and another STREAMS driver or module. *Eli* provides an LLI compatible interface, which is expected by *ip*(ADMP). *Eli* must be pushed on the STREAM between *ip* and *emd*.

It is expected that since the 10base5 driver is now available as a product, EMD will no longer be used, and *eli* will become obsolete.

See Also

strcf(SFF), ip(ADMP).

icmp

Internet Control Message Protocol

Syntax

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

s = socket(AF_INET, SOCK_RAW, proto);

Description

ICMP is the error and control message (or device) protocol used by IP and the Internet protocol family. It may be accessed through a "raw socket" for network monitoring and diagnostic functions. The *proto* parameter to the socket call to create an ICMP socket is obtained from *getprotobyname* . [See *getprotoent*(SLIB).] ICMP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls; the *connect*(SSC) call may also be used to fix the destination for future packets (in which case the *read*(S) or *recv*(SSC) and *write*(S) or *send*(SSC) system calls may be used).

Outgoing packets automatically have an IP header prepended to them (based on the destination address). Incoming packets are received with the IP header and options intact.

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 already connected;

[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket has not been connected;

[ENOSR] 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.

Files

/dev/inet/icmp

See Also

send(SSC), recv(SSC), intro(ADMP), inet(ADMP), ip(ADMP).

inet

Internet protocol family

Syntax

#include <sys/types.h>
#include <netinet/in.h>

Description

The Internet protocol family is a set of protocols using the *Internet Protocol* (IP) network layer and the Internet address format. The Internet family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types; the SOCK_RAW interface provides access to the IP protocol.

Addressing

Internet addresses are four-byte quantities, stored in network standard format. The include file < **sys/in.h** > defines this address as a discriminated union.

Sockets bound to the Internet protocol family use the following addressing structure:

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

When using sockets, the *sin_family* is specified in host order, and the sin_port and *sin_addr* fields are specified in network order.

Sockets may be created with the local address INADDR_ANY to affect wildcard matching on incoming messages. The address in a *connect*(SSC) or *sendto* [see *send*(SSC)] 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.

When using the Transport Layer Interface (TLI), transport providers such as *tcp*(ADMP) support addresses whose lengths vary from eight to sixteen bytes. The eight byte form is the same as a *sockaddr_in* without the *sin_zero* field. The sixteen byte form is identical to

sockaddr_in. Additionally, when using TLI, the *sin_family* field is accepted in either host or network order.

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 abstraction; UDP is used to support the SOCK_DGRAM abstraction. A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The ICMP message protocol is accessible from a raw socket.

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*(S) commands on a datagram socket in the Internet "communications domain"; they have the same form as the SIOCIFADDR command. [See *intro*(ADMP).]

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

ioctl(S), socket(SSC), intro(ADMP), intro(SFF), icmp(ADMP), ip(ADMP), tcp(ADMP), udp(ADMP).

Note

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.

ip

Internet Protocol

Syntax

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

s = socket(AF_INET, SOCK_RAW, proto);

Description

IP is the network layer protocol used by the Internet protocol family. Options may be set at the IP level when using higher-level protocols that are based on IP (such as TCP and UDP). It may also be accessed through a "raw socket" or device when developing new protocols or special purpose applications.

A single generic option IP_OPTIONS, is supported at the IP level, and may be used to provide IP options to be transmitted in the IP header of each outgoing packet. Options are set with *setsockopt* and examined with *getsockopt* . [See *getsockopt* (SSC).] The format of IP options to be sent is that specified by the IP protocol specification, with one exception: the list of addresses for Source Route options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.

Raw IP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls; the *connect* (SSC) call may also be used to fix the destination for future packets (in which case, the *read* (S) or *recv* (SSC), and *write* (S) or *send* (SSC) system calls may be used).

If *proto* is 0, the default protocol IPPROTO_RAW is used for outgoing packets, and only incoming packets destined for that protocol are received. If *proto* is non-zero, that protocol number will be used on outgoing packets and to filter incoming packets. *Proto* must be specified in *sockcf(SFF)*.

Outgoing packets automatically have an IP header prepended to them (based on the destination address given and the protocol number the socket is created with). Incoming packets are received with IP header and options intact.

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, or when trying to send a datagram with the destination address specified and the socket already connected;

[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket has not been connected;

[ENOSR] 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.

The following errors specific to IP may occur when setting or getting IP options:

[EINVAL] An unknown socket option name was given.

[EINVAL] The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

Files

/dev/inet/ip
/dev/inet/rip

See Also

getsockopt(SSC), send(SSC), recv(SSC), sockcf(SFF), intro(ADMP), icmp(ADMP), inet(ADMP).

llcloop

software loopback network interface

Syntax

```
#include <sys/socket.h>
#include <netinet/in.h>
struct sockaddr_in sin;

s = socket(AF_INET, SOCK_XXX, 0);
.
.
.
sin.sin_addr.s_addr = htonl (INADDR_ANY);
bind(s, (char *)&sin, sizeof(sin));
```

Description

The *llcloop* interface is a software loopback mechanism which may be used for performance analysis, software testing, and/or local communication. As with other network interfaces, the loopback interface must have network addresses assigned for each address family with which it is to be used. (Currently, only the Internet address family is supported.) These addresses may be set or changed with the SIOCSIFADDR ioctl. The loopback interface should be the first one configured, otherwise nameserver lookups for hostnames of other interfaces may fail.

Files

/dev/llcloop

See Also

intro(ADMP), inet(ADMP).

slip

serial line IP network interface

Description

The slip interface is a driver that allows IP datagrams to be sent over normal serial lines. This is useful for connecting machines that do not have Ethernet hardware. As with other network interfaces, the slip interface must have network addresses assigned for each address family with which it is to be used. (Currently, only the Internet address family is supported.) These addresses may be set or changed with the SIOCSIFADDR ioctl.

See Also

ifconfig(ADMN), slattach(ADMN), sldetach(ADMN), intro(ADMP), inet(ADMP).

sock

Socket Interface Driver

Description

The socket driver is used to provide socket emulation to applications. Sockets are an alternate entry point into transport providers, such as *tcp*(ADMP). The socket driver is a character device that acts as an alternate stream head, augmenting the functions of the standard stream head. It also provides support for miscelleanous functions such as *select*(SSC).

FILES

/dev/socksys

SEE ALSO

ifconfig(ADMN), intro(SSC), slattach(ADMN), sldetach(ADMN), intro(ADMP), inet(ADMP)

tcp

Internet Transmission Control Protocol

Syntax

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

s = socket(AF_INET, SOCK_STREAM, 0);

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. TCP uses the standard Internet address format and, in addition, 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.

Sockets using the *tcp* protocol are either "active" or "passive." Active sockets initiate connections to passive sockets. By default, TCP sockets are created active; to create a passive socket, the *listen*(SSC) system call must be used after binding the socket with the *bind*(SSC) system call. Only passive sockets may use the *accept*(SSC) call to accept incoming connections. Only active sockets may use the *connect*(SSC) call to initiate connections.

Passive sockets may "underspecify" their location to match incoming connection requests from multiple networks. This technique, called "wildcard addressing," allows a single server to provide service to clients on multiple networks. To create a socket that 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 socket's address is fixed by the peer entity's location. The address assigned the socket 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 one socket option that is set with *setsockopt* and tested with *getsockopt* . [See *getsockopt*(SSC).] 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. For a small number of clients, such as window systems that send a stream of mouse events that 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. The option level for the *setsockopt* call is the protocol number for TCP, available from *getprotobyname* . [See *getprotoent* (SLIB).]

- Options at the IP transport level may be used with TCP; see *ip* (ADMP). Incoming connection requests that are source-routed are noted, and the reverse source route is used in responding.

TCP is also available as a TLI connection-oriented protocol via the special file **/dev/inet/tcp**. TCP options are supported via the TLI options mechanism.

TCP provides a facility, *one-packet mode*, that attempts to improve performance over Ethernet interfaces that cannot handle back-to-back packets. One-packet mode may be set by *ifconfig* (1M) for such an interface. On a connection that uses an interface for which one-packet mode has been set, TCP attempts to prevent the remote machine from sending back-to-back packets by setting the window size for the connection to the maximum segment size for the interface.

Certain TCP implementations have an internal limit on packet size that is less than or equal to half the advertised maximum segment size. When connected to such a machine, setting the window size to the maximum segment size would still allow the sender to send two packets at a time. To prevent this, a "small packet size" and a "small packet threshold" may be specified when setting one-packet mode. If, on a connection over an interface with one-packet mode enabled, TCP receives a number of consecutive packets of the small packet size equal to the small packet threshold, the window size is set to the small packet size.

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;

[ENOSR] 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 connection establishment (usually because no process is listening to the port);

[EADDRINUSE] when an attempt is made to create a socket with a port which has already been allocated;

[EADDRNOTAVAIL]
when an attempt is made to create a socket with a network address for which no network interface exists.

Files

/dev/inet/tcp

See Also

ifconfig(ADMN), getsockopt(SSC), socket(SSC), intro(ADMP), inet(ADMP), ip(ADMP).

udp

Internet User Datagram Protocol

Syntax

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

s = socket(AF_INET, SOCK_DGRAM, 0);

Description

UDP is a simple, unreliable datagram protocol that is used to support the SOCK_DGRAM abstraction for the Internet protocol family. UDP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls; the *connect*(SSC) call may also be used to fix the destination for future packets (in which case, the *recv*(SSC), or *read*(S) and *send*(SSC), or *write*(S) system/library calls may be used). In addition, UDP is available as TLI connectionless transport via the special file **/dev/inet/udp**.

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 (that is, 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.

Options at the IP transport level may be used with UDP; see *ip*(ADMP).

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, or when trying to send a datagram with the destination address specified and the socket already connected;

[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket has not been connected;

[ENOSR] 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.

Files

/dev/inet/udp

See Also

getsockopt(SSC), recv(SSC), send(SSC), socket(SSC), intro(ADMP), inet(ADMP), ip(ADMP), RFC768.

vty

pseudo terminal slave driver

ttyp

pseudo terminal master driver

Description

The ttyp and vty drivers together provide 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 processes an interface identical to that described in *termio*(ADMP). However, whereas all other devices which provide the interface described in *termio*(ADMP) 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.

The following *ioctl* call applies 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 a la ^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*(TC) and *rlogind*(ADMN) 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.

Files

/dev/ptyp[0-f][0-f]	master pseudo terminals
/dev/ttyp[0-f][0-f]	slave pseudo terminals

See Also

termio(ADMP).

Contents

Formats of Files Used by Networking Commands (**SFF**)

intro

introduction to formats of files used by networking commands

Description

This section outlines the formats of various files. The C struct declarations for the file formats are given where applicable. Usually, these structures can be found in header files under the directories **/usr/include**, **/usr/include/net**, **/usr/include/netinet**, or **/usr/include/sys**.

References of the type named(ADMN) refer to entries found in Section ADMN of the TCPIP Network Administrator's Reference.

aliases

aliases file for sendmail

Syntax

/usr/lib/aliases

Description

This file describes user id aliases used by */usr/lib/sendmail*. It is formatted as a series of lines of the form

name: name_1, name2, name_3, . . .

The *name* is the name to alias, and the *name_n* are the aliases for that name. Lines beginning with white space are continuation lines. Lines beginning with '#' are comments.

Aliasing occurs only on local names. Loops can not occur, since no message will be sent to any person more than once.

After aliasing has been done, local and valid recipients who have a ".forward" file in their home directory have messages forwarded to the list of users defined in that file.

See Also

sendmail(ADMN)

hosts

list of hosts on network

Description

The file **/etc/hosts** is a list of hosts that share the network, including the local host. It is referred to by programs that need to translate between host names and DARPA Internet addresses when the name server is not being used [See *named*(ADMN).] Each line in the file describes a single host on the network and consists of three fields separated by any number of blanks or tabs:

address name aliases ...

where

address is the DARPA Internet address. Unless another type of address is required by some host on the network, *address* should be a Class A address, which takes the form *net.node*, where *net* is the network number from **/etc/networks** (see *networks* (4)), that must between 0 and 127; and *node* is a value which must be unique for each host and be between 0 and 16777215.

name is the official name of the host. If the host is a computer system running UNIX, it must claim this host name by executing *hostname*(TC) when it is initializing itself.

aliases... is a list of alternate names for the host. Aliases can be used in network commands in place of the official name.

It is suggested that you specify the *hostname* and the *node name* [see *hostname*(TC) and *uname*(C)] as aliases for one another for each machine listed in the **/etc/hosts** file.

The routines which search this file ignore comments (portions of lines beginning with #) and blank lines.

An internet address can actually take one of four forms:

A *A* is a simple 32-bit integer.

A.B *A* is an eight-bit quantity occupying the high-order byte and *B* is a 24-bit quantity occupying the remaining bytes. This form is suitable for a Class A address of the form *net.node*.

A.B.C *A* is an eight-bit quantity occupying the high-order byte; *B* is an eight-bit quantity occupying the next byte; and *C* is a 16-bit quantity occupying the remaining bytes. This form is suitable for a Class B address of the form **128.***net.node*.

A.B.C.D The four parts each occupy a byte in the address.

Example

```
#       Engineering network

192.35.53.1         laizy.Lachman.COM laizy
192.35.53.2         laidback.Lachman.COM laidback
192.35.53.85        laiter.Lachman.COM laiter# Sun-3/50 [stevea]
```

Files

/etc/hosts

See Also

hostname(TC), uname(C), networks(SFF), inet(ADMP).

hosts.equiv

list of trusted hosts

Description

Hosts.equiv resides in directory **/etc** and contains a list of trusted hosts. When an *rlogin*(1) or *rcmd*(1) request from such a host is made, and the initiator of the request is in */etc/passwd*, then no further validity checking is done. That is, *rlogin* does not prompt for a password, and *rsh* completes successfully. So a remote user is "equivalenced" to a local user with the same user ID when the remote user is in **hosts.equiv**.

The format of **hosts.equiv** is a list of names, as in this example:

```
host1
host2
```

A line consisting of a simple host name means that anyone logging in from that host is trusted. The **.rhosts** file has the same format as **hosts.equiv**. When user *XXX* executes *rlogin* or *rcmd*, the **.rhosts** file from *XXX*'s home directory is conceptually concatenated onto the end of **hosts.equiv** for permission checking. In the special case when the user is the super-user then only the **/.rhosts** file is checked.

It is also possible to have two entries (separated by a single space) on a line of these files. In this case, if the remote host is equivalenced by the first entry, then the user named by the second entry is allowed to log in as anyone, that is, specify any name to the **-l** flag (provided that name is in the **/etc/passwd** file, of course). Thus

```
laidbak ez
```

allows *ez* to log in from laidbak as anyone. The usual usage would be to put this entry in the **.rhosts** file in the home directory for *derek* . Then *ez* may log in as *derek* when coming from laidbak.

Files

/etc/hosts.equiv
$HOME/.rhost

See Also

rlogin(TC), rcmd(TC)

inetd.conf

configuration file for inetd (internet "super-server").

Description

inetd.conf is the configuration file for the *inetd*(SFF) System V STREAMS TCP/IP internetworking "super-server".

The file consists of a series of single-line entries, each entry corresponding to a service to be invoked by *inetd*. These services are connection-based, datagram, or "internal".

Internal services are those supported by the *inetd* program: 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.

Each service, including internal services, must have a valid entry in **/etc/services**(ADMN). In the case of an internal service, its name must correspond to the official name of the service: that is, the first entry in **/etc/services**.

Each entry has a series of space- or tab-separated fields. (No field, except for the last one, may be omitted.) The fields are as follows:

service name
Name of a valid service in **/etc/services**, as described above.

socket type
One of "stream", "dgram", or "raw", depending on whether the socket type is stream, datagram, or raw [see *socket*(SSC)].

protocol
Name of a valid protocol (for example, "tcp") specified in **/etc/protocols**(ADMN).

wait/nowait
Specifies whether the socket can be made available for new connections while there is still data waiting on the socket. The value is always "nowait" unless it is a datagram socket. If it is a datagram socket, the value is usually "wait", although "nowait" is possible in some cases. (Note that *tftpd* is an exception in that it must have "wait" specified, and yet the socket can continue to process messages on the port.)

user
Name of the user as whom the server should run. This allows servers to be run with less permission than root.

server program
Except in the case of internal services, full pathname of the server program to be invoked by *inetd* when a request is waiting on a socket. For an internal service, the value is "internal".

server program arguments
Arguments to the server program, starting with *argv*[0], which is the name of the program. For an internal service, the value is "internal".

Comments are denoted by a "#" at the beginning of a line.

The distribution **inetd.conf** file contains prototype entries; refer to these entries when editing the file.

Example

```
                .
                .
                .
ftp     stream tcp   nowait  root    /etc/ftpd       ftpd
telnet  stream tcp   nowait  root    /etc/telnetd    telnetd
login   stream tcp   nowait  root    /etc/rlogind    rlogind
exec    stream tcp   nowait  root    /etc/rexecd     rexecd
finger  stream tcp   nowait  sync    /etc/fingerd    fingerd
echo    stream tcp   nowait  root    internal
discard stream tcp   nowait  root    internal
chargen         stream          tcp     nowait          rootinternal
daytime         stream          tcp     nowait          rootinternal
time    stream tcp   nowait  root    internal
echo    dgram  udp   wait    root    internal
discard dgram  udp   wait    root    internal
chargen         dgram           udp     wait            rootinternal
daytime         dgram           udp     wait            rootinternal
time    dgram  udp   wait    root    internal
                .
                .
                .
```

See Also

fingerd(ADMN), ftpd(ADMN), inetd(ADMN), rexecd(ADMN), rlogin(ADMN), rshd(ADMN), telnetd(ADMN), tftpd(ADMN), protocols(SFF), services(SFF).

localhosts

configuration file for sendmail

Description

Localhosts is a file that lists hosts that are to be treated as equivalent by *sendmail*(ADMN). In the distributed configuration files, an equivalent host is in class S. *Sendmail* also looks at **/etc/hosts.equiv**.

The format of *localhosts* is very simple. It consists of a list of hostnames, one per line. There is no support for comments.

Example

```
laidbak
laiter
laisagna
```

Files

/usr/lib/mail/localhosts

See ALso

hosts.equiv(SFF), sendmail(ADMN), sendmail(SFF), uucpindomain(SFF).
Sendmail Installation and Operations Guide.

netrc

login file for remote networks

Description

If the **.netrc** file exists, it will be used by *ftp*(TC) for automatic login on the remote host. For each remote host, the file contains a one-line entry that describes the login data for the user on that host.

An entry may consist of up to three blank-separated fields introduced by keywords. The keyword is followed by the literal data needed for login. The following keywords are available:

machine The hostname of the machine.

login The user login name for that host.

password (Optional) The user's password on that host. **NOTE:** The literal password must be given in clear text; it is not encrypted.

If the **.netrc** file includes the password feature, permissions on the file must be set to prohibit reading by group and others; the file will not otherwise take effect.

Example

The following example entry allows automatic login on the "admin" host by a user named "superuser" whose password is "open".

```
machine admin login superuser password open
```

Files

$HOME/.netrc

See Also

ftp(TC).

Warning

For security reasons, use of the password feature is not recommended.

networks

names and numbers for the internet

Description

The file **/etc/networks** lists networks on the internet. Each line describes a single network and consists of the following blank separated fields:

name number aliases ...

where

name is the official name of the network. All hosts on the internet should use the same official name for a given network.

number is the network number, which serves as part of the DARPA Internet address for each host on the internet. All hosts on the internet must use the same number for a given network.

aliases ... is a blank-separated list of local aliases for the network.

The routines which search this file ignore comments (portions of lines beginning with #) and blank lines.

Example

```
# Building 1 Internet
Lachman-Net 192.35.52     #General
LAI-TCP-Net  192.35.53    #TCP Development
```

See Also

hosts(SFF).

Files

/etc/networks

protocols

list of Internet protocols

Description

The file **/etc/protocols** lists known DARPA Internet protocols. Each line describes a single protocol and consists of the following blank separated fields:

name number aliases ...

where

name is the official name of the protocol.

number is the protocol number.

aliases ... is a blank-separated list of local aliases for the protocol.

The routines which search this file ignore comments (portions of lines beginning with #) and blank lines.

Protocol names and numbers are specified by the DDN Network Information Center. Do not change this file.

Files

/etc/protocols

See Also

socket(SSC), slink(ADMN), ldsocket(ADMN).

resolver

resolver configuration file

Syntax

/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 is designed to be human readable and 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 3) name servers may be listed; if more than one name server is specified, the resolver library queries each one 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; then repeat trying all the name servers until a maximum number of retries are made.

domain
: followed by an 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 domain returned by *gethostname* (SLIB) is used (everything after the first '.'). Finally, if the host name does not contain a domain part, the root domain is assumed.

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.

Example

```
domain   Lachman.COM
nameserver      192.35.52.1
nameserver      192.35.52.2
```

Files

/etc/resolv.conf

See Also

named(ADMN), resolver(SFF), hosts(ADMN), byteorder(SLIB), rexec(SLIB).
Name Server Operations Guide for BIND

rhosts

remote equivalent users

Description

These files grant permission for remote users to use local user names without knowing the corresponding user passwords. This is known as making the remote user "equivalent" to the local user, and is convenient, for example, when one person owns user names on more than one host.

If a user's home directory contains a file named **.rhosts**, remote users specified in the file are equivalent to the local user. Each user specification in the file consists of the remote user host name and user name, separated by a space. (If an asterisk is substituted for either name, any name will match.) For security reasons, **.rhosts** must belong to the user granting the equivalence or to root.

The file **/etc/hosts.equiv** is a list of remote hosts with matching-name equivalence. The file lists remote hosts one per line. On each host listed in **/etc/hosts.equiv**, a remote user with the same name as a local user is equivalent to the local user. In effect, the users are the same if the names are the same.

Files

$HOME/.rhosts
/etc/hosts.equiv

See Also

rcmd(TC), rcp(TC), rlogin(TC).

Warnings

When a system is listed in **/etc/hosts.equiv**, its security must be as good as local security. One insecure system mentioned in **/etc/hosts.equiv** can compromise the security of an entire network.

sendmail.cf

configuration file for sendmail

Description

Sendmail.cf is the configuration file for the sendmail mail router. A full description of this file can be found in chapter nine of the *STREAMS TCP User's Guide.*

Files

/usr/lib/sendmail.cf

See Also

sendmail(ADMN), localhosts(SFF), uucpindomain(SFF).
Sendmail Installation and Operations Guide.

services

list of Internet services

Description

The file **/etc/services** lists known DARPA Internet services. Each line describes a single service and consists of the following blank separated fields:

name number/protocol aliases ...

where:

name is the official name of the service.

number is the service number.

protocol is the name of the protocol used by the service. (See *protocols*(SFF).)

aliases ... is a blank-separated list of local aliases for the service.

The routines which search this file ignore comments (portions of lines beginning with #) and blank lines.

Service names and numbers are specified by the DDN Network Information Center. Do not change this file unless you are familiar with DARPA Internet internals.

Files

/etc/services

See Also

inetd(ADMN), inetd.conf(SFF).

/etc/strcf

STREAMS Configuration File for STREAMS TCP/IP

Description

/etc/strcf contains the script that is executed by *slink*(SFF) to perform the STREAMS configuration operations required for STREAMS TCP/IP.

The standard **/etc/strcf** file contains several functions that perform various configuration operations, along with a sample **boot** function. Normally, only the **boot** function must be modified to customize the configuration for a given installation. In some cases, however, it may be necessary to change existing functions or add new functions.

The following functions perform basic linking operations:

Function **tp** is used to set up the link between a transport provider, such as TCP, and IP.

```
#
# tp - configure transport provider (i.e. tcp, udp, icmp)
# usage: tp devname
#
tp {
        p = open $1
        ip = open /dev/inet/ip
        link p ip
}
```

Function **linkint** links the specified streams and does a **sifname** operation with the given name.

```
#
# linkint - link interface to ip or arp
# usage: linkint top bottom ifname
#
linkint {
        x = link $1 $2
        sifname $1 x $3
}
```

Function **aplinkint** performs the same function as **linkint** for an interface that uses the **arpproc** module.

```
#
# aplinkint - like linkint, but arpproc is pushed on dev
# usage: aplinkint top bottom ifname
#
aplinkint {
        push $2 arpproc
        linkint $1 $2 $3
}
```

The following functions are used to configure different types of Ethernet interfaces:

Function **uenet** is used to configure an Ethernet interface for a cloning device driver that uses the *unit select* ioctl to select the desired interface. The interface name is constructed by concatenating the supplied prefix and the unit number.

```
#
# uenet - configure ethernet-type interface for cloning driver using
#       unit select
# usage: uenet ip-fd devname ifprefix unit
#
uenet {
        ifname = strcat $3 $4
        dev = open $2
        unitsel dev $4
        aplinkint $1 dev ifname
        dev = open $2
        unitsel dev $4
        arp = open /dev/inet/arp
        linkint arp dev ifname
}
```

Function **denet** performs the same function as **uenet**, except that *DL_ATTACH* is used instead of *unit select*.

```
#
# denet - configure ethernet-type interface for cloning driver using
#       DL_ATTACH
# usage: denet ip-fd devname ifprefix unit
#
denet {
        ifname = strcat $3 $4
        dev = open $2
        dlattach dev $4
        aplinkint $1 dev ifname
        dev = open devname
        dlattach dev $4
        arp = open /dev/inet/arp
        linkint arp dev ifname
}
```

Function **cenet** is used to configure an Ethernet interface for a cloning device driver that uses a different major number for each interface. The device name is formed by concatenating the supplied device name prefix and the unit number. The interface name is formed in a similar manner using the interface name prefix.

```
#
# cenet - configure ethernet-type interface for cloning driver with
#       one major per interface
# usage: cenet ip-fd devprefix ifprefix unit
#
cenet {
        devname = strcat $2 $4
        ifname = strcat $3 $4
        dev = open devname
        aplinkint $1 dev ifname
        dev = open devname
        arp = open /dev/inet/arp
        linkint arp dev ifname
}
```

Function
senet
is used to configure an Ethernet interface for a non-cloning device driver. Two different device nodes must be specified for IP and ARP.

```
#
# senet - configure ethernet-type interface for non-cloning driver
# usage: senet ip-fd ipdevname arpdevname ifname
#
senet {
        dev = open $2
        aplinkint $1 dev $4
        dev = open $3
        arp = open /dev/inet/arp
        linkint arp dev $4
}
```

Function **senetc** is like **senet**, except that it allows the specification of a convergence module to be used with the ethernet driver (e.g. for the 3B2 emd driver).

```
#
# senetc - configure ethernet-type interface for non-cloning driver
#       using convergence module
# usage: senetc ip-fd convergence ipdevname arpdevname ifname
#
senetc {
        dev = open $3
        push dev $2
        aplinkint $1 dev $5
        dev = open $4
        push dev $2
        arp = open /dev/inet/arp
        linkint arp dev $5
}
```

Function **loopback** is used to configure the loopback interface.

```
#
# loopback - configure loopback device
# usage: loopback ip-fd
#
loopback {
        dev = open /dev/llcloop
        linkint $1 dev lo0
}
```

Function **slip** is used to configure a SLIP interface. This function is not normally executed at boot time. Rather, the *slattach*(ADMN) command runs *slink* specifying **slip** on the command line.

```
#
# slip - configure slip interface
# usage: slip unit
#
slip {
        ip = open /dev/inet/ip
        s = open /dev/slip
        ifname = strcat sl $1
        unitsel s $1
        linkint ip s ifname
}
```

Function **boot** is called by default when *slink* is executed. Normally, only the *interfaces* section and possibly the *queue params* section will have to be customized for a given installation. Examples are provided for the various Ethernet driver types.

```
#
# boot - boot time configuration
#
boot {
        #
        # queue params
        #
        initqp /dev/inet/udp rq 8192 40960
        initqp /dev/inet/ip muxrq 8192 40960 rq 8192 40960
        #
        # transport
        #
        tp /dev/inet/tcp
        tp /dev/inet/udp
        tp /dev/inet/icmp
        #
        # interfaces
        #
        ip = open /dev/inet/ip
        senetc ip eli /dev/emd0 /dev/emd1 en0
#       uenet ip /dev/abc en 0
#       denet ip /dev/def en 0
#       cenet ip /dev/ghi en 0
#       senet ip /dev/jkl0 /dev/jkl1 en0
        loopback ip
}
```

Files

/etc/strcf

See Also

slink(ADMN), intro(ADMP).

uucpindomain

configuration file for sendmail

Description

Uucpindomain is a file that lists hosts that are connected by UUCP, but should be treated as if they were in the local domain by *sendmail*(1M). In the distributed configuration files, this type of host is in class **L**.

The format of *uucpindomain* is very simple. It consists of a list of hostnames, one per line. There is no support for comments.

Example

```
huey
duey
louie
```

Files

/usr/lib/mail/uucpindomain

See Also

hosts.equiv(SFF), localhosts(SFF), sendmail(ADMN), sendmail(SFF).
Sendmail Installation and Operations Guide.

Index

E

F

G

H

I

K

L

M

N

P

Q

R

S

T

U

W

SCO® NFS™

Derived from

LACHMAN™ SYSTEM V NFS

Reference Guide

The Santa Cruz Operation, Inc.

SCO NFS was developed by Lachman Associates.
SCO NFS is derived from LACHMAN™ SYSTEM V NFS, a joint development of Lachman Associates and Sun Microsystems.

SCO Document Number: XN-9-30-89-1.1.0B

Commands

Network Administration **(NADM)**

Intro	Introduction to ONC maintenance and operation commands
fsirand	install random inode generation numbers
lckclnt	create lock manager client handles
lockd	Network lock daemon
mount	Mount and unmount file system and remote resources
mountd	NFS mount requested server
nfs	NFS start/stop script
nfsclnt	create NFS client handles
nfsd	NFS daemons
nfsstat	Network File System statistics
nmountall	mount, unmount multiple file systems
passmgmt	password files management
pcnfsd	pc-nfs authentication and print spooling daemon
portmap	DARPA port to RPC program number mapper
rexd	RPC-based remote execution server
rpcinfo	Report RPC information
showmount	Show all remote mounts
statd	Network status monitor

intro

introduction to ONC maintenance and operation commands

Description

This section contains information related to ONC operation and maintenance. It describes the commands used to start and stop NFS, mount and unmount NFS file systems, ping RPC based services and the various NFS/ONC daemons and utilities.

fsirand

install random inode generation numbers

Syntax

fsirand [**-p**] *special*

Description

fsirand installs random inode generation numbers on all the inodes on device *special*. This helps increase the security of filesystems exported by NFS.

fsirand must be used only on an unmounted filesystem that has been checked with *fsck*(ADM). The only exception is that it can be used on the root filesystem in single-user mode, if the system is immediately re-booted afterwords.

Options

-p Print out the generation numbers for all the inodes, but do not change the generation numbers.

See Also

fsck(ADM)

lckclnt

create lock manager client handles

Syntax

lckclnt [nclienthandles]

Description

lckclnt allocates connectionless transport endpoints which are used to create client handles. *Lock manager* client programs obtain a client handle for the duration of an RPC operation.

nclienthandles is the number of client handles allocated. This number limits the number of *lock manager* client operations that can be run concurrently, and should be based on the load expected on the client. If additional client handles are required, more *lckclnt* processes may be started. The number of client handles available to *lock manager* client programs is the sum of the number of client handles allocated by each *lckclnt* program. Killing an *lckclnt* process will reduce the available number of client handles by the amount that was initially allocated by that process.

Files

/dev/inet/udp UDP device node

See Also

kclt_create(NS)

lockd

Network lock daemon

Syntax

/etc/lockd [**-d** *debuglevel*] [**-t** *timeout*] [**-g** *graceperiod*] [**-h** *hashsize*] [**-l** *k2utimeout*]

Description

lockd processes lock requests that are either sent locally by the kernel or remotely by another lock daemon. *lockd* forwards lock requests for remote data to the server site's lock daemon through the RPC/XDR package. *lockd* then requests the status monitor daemon, *statd(NADM)*, for monitor service. The reply to the lock request will not be sent to the kernel until the status daemon and the server site's lock daemon have replied. If either the status monitor or server site's lock daemon is unavailable, the reply to a lock request for remote data is delayed until all daemons become available.

When a server recovers, it waits for a grace period for all client site *lockds* to submit reclaim requests. Client site *lockds*, on the other hand, are notified by the *statd* of the server recovery and promptly resubmit previously granted lock requests. If a *lockd* fails to secure a previously granted lock at the server site, the *lockd* sends SIGUSR2 to a process.

lockd should be invoked as early as possible during the transition from single user to multiuser, so that no other processes have the opportunity to get a standard System V lock. If there are active locks or sleeping locks in the standard System record locking code, *lockd* attempts to migrate them to the user level process. This is done so the active locking processes will not have their locks destroyed by the starting lock manager.

Options

-t *timeout* *lockd* uses *timeout* (seconds) as the interval instead of the default value (5 seconds) to retransmit lock request to the remote server.

-d*debuglevel*
lockd has extensive internal reporting capabilities. A level of 2 reports significant events. Level 4 reports internal state and all request traffic. Level 4 is considered verbose.

-g *graceperiod*
lockd uses *graceperiod* (seconds) as the grace period duration instead of the default value (45 seconds).

-h*hashsize* *lockd* uses *hashsize* has buckets internally instead of the default of 29.

-l*k2utimeout*
lockd uses *k2utimeout* seconds as the interval instead of the default value of 2 seconds to retransmit kernel lock manager requests. This is the timeout value used for local lock requests.

See Also

fcntl(S), lockf(S), signal(S), statd(NADM)

mount, umount

Mount and unmount file system and remote resources

Syntax

/etc/mount [[-r] [-f *fstyp*] *special directory*]
/etc/mount [-r] [-c] [-d *resource directory*]
/etc/umount special directory
/etc/umount [-d] *resource*

Description

File systems other than **root** (/) are considered *removable* in the sense that they can be either available to users or unavailable. *mount* announces to the system that *special*, a block special device or *resource*, a remote resource, is available to users from the mount point directory. *directory* must exist already; it becomes the name of the root of the newly mounted *special*.

mount, when entered with arguments, adds an entry to the table of mounted devices, **/etc/mnttab**. *umount* removes the entry. If invoked with no arguments, *mount* prints the entire mount table. If invoked with an incomplete argument list, *mount* searches **/etc/default/filesys** for the missing arguments:
special, **-d** *resource*, *directory*, or **-d** *directory*.

Options

-r A *special* or *resource* is to be mounted read-only. If *special* or *resource* is write-protected or read-only advertised, this flag must be used.

-d A *resource* is a remote resource that is to be mounted on *directory* or unmounted.

-c indicates that remote reads and writes should not be cached in the local buffer pool. **-c** is used in conjuncion with **-d**.

-f*fstyp* An *fstype* is the file system type to be mounted. If this argument is omitted, it defaults to the **root** *fstyp*. If *fstyp* is NFS, then NFS options may be added after the *fstyp* separated by commas. The available NFS options are:

soft
return error if the server does not respond.

rsize=*n*
set the read buffer size to *n* bytes. (default = 8192)

wsize=*n*
set the write buffer size to *n* bytes. (default = 8192)

timeo=*n*
set the initial NFS timeout to *n* tenths of a second.

retrans=*n*
set the number of NFS retransmissions to *n*.

port=*n*
set the server IP port number to *n*.

nosuid
ignore setuid and setgid bits during exec.

bg background this mount. This is recommended for automatic mounts done during system startup.

special The block special device that is to be mounted on *directory*. If *fstyp* is NFS, then *special* should be of the form **hostname:/pathname**.

resource The remote resource name that is to be mounted on a directory.

directory The directory mount point for *special* or *resource*. (The directory must already exist.)

umount announces to the system that the file system previously mounted *special* or *resource* is to be made unavailable. If invoked with an incomplete argument list, *umount* searches **/etc/default/filesys** for the missing arguments.

mount can be used by any user to list mounted file systems and resources. Only a super-user can mount and umount file systems.

Files

/etc/mnttab mount table
/etc/default/filesys file system table

Example

The following command mounts the root file system of the remote machine *test* onto the mount point *mnt* and specifies the file system type.

```
mount -f nfs,soft,rsize=1024, wsize=1024  test:/  /mnt
```

See Also

setmnt(ADM), mountd(NADM), nfsd(NADM), showmount(NADM), mount(NS), mnttab(NF)

Diagnostics

If the *mount*(NS) system call fails, *mount* prints an appropriate diagnostic. *mount* issues a warning if the file system to be mounted is currently mounted under another name. A remote resource mount will fail if the resource is not available.

umount fails if *special* or *resource* is not mounted or if it is busy. *special* or *resource* is busy if it contains an open file or some user's working directory.

Warnings

Physically removing a mounted file system diskette from the diskette drive before issuing the *umount* command damages the file system.

mountd

NFS mount requested server

Syntax

/etc/mountd

Description

mountd is an **RPC** server that answers file system mount requests. It reads the file **/etc/exports**, described in *exports*(NF), to determine which file systems are available to which machines and users. It also provides information as to which clients have file systems mounted. This information can be printed using the *showmount*(NADM) command.

See Also

exports(NF), services(ADMN), showmount(NADM)

/etc/nfs

NFS start/stop script

Syntax

/etc/nfs start
/etc/nfs stop

Description

/etc/nfs is used to start or stop the NFS software. NFS will start automatically at system startup time if */etc/nfs* is linked to **/etc/rc2.d/S***name* (name is installed as *72nfs* by default). Similarly, NFS will stop automatically at system shutdown time if */etc/nfs* is linked to **/etc/rc0.d/K***name* (name is installed as *66nfs* by default).

/etc/nfs must be customized for a particular installation before it can be used. The following items may need to be edited:

PATH — The supplied path may require modification if commands run by */etc/nfs* are in other directories.

PROCS — The "PROCS" variable contains a space-separated list of names of processes to kill when executing the **stop** function. If additional daemons are used, their names can be added to this list.

Daemons — The standard NFS daemons are started at this point. Any additional daemons or other commands may be included in this section. Any of the standard daemons that are not desired may be removed or commented out. By default, *biod* (see *nfsd*(NADM)),

A transport service for NFS may have to be initialized before NFS is started. Conversely, the service have to be turned off after NFS is brought down.

See Also

domainname(NC), portmap(NADM), mountd(NADM), nfsd(NADM), rexd(NADM), lockd(NADM), statd(NADM), sh(C).

nfsclnt

create NFS client handles

Syntax

nfsclnt [nclienthandles]

Description

nfsclnt allocates connectionless transport endpoints which are used to create client handles. *NFS* client programs obtain a client handle for the duration of an RPC operation.

nclienthandles is the number of client handles allocated. This number limits the number of *NFS* client operations that can be run concurrently, and should be based on the load expected on the client. If additional client handles are required, more *nfsclnt* processes may be started. The number of client handles available to *NFS* client programs is the sum of the number of client handles allocated by each *nfsclnt* program. Killing an *nfsclnt* process will reduce the available number of client handles by the amount that was initially allocated by that process.

Files

/dev/inet/udp UDP device node

See Also

kclt_create(NS)

nfsd, biod

NFS daemons

Syntax

/etc/nfsd [nservers]

/etc/biod [nservers]

Description

nfsd starts the NFS server daemons that handle client filesystem requests. *nservers* is the number of filesystem request daemons to start. This number should be based on the load expected on this server.

biod, run on an NFS client, starts *nservers* asynchronous block I/O daemons, which do read-ahead and write-behind of blocks from the client's buffer cache.

See Also

mountd(NADM), exports(NF)

nfsstat

Network File System statistics

Syntax

nfsstat [**-csnrz**]

Description

nfsstat displays statistical information about the Network File System (NFS) and Remote Procedure Call (RPC) interfaces to the kernel. It can also be used to reinitialize this information. If no options are given the default is

nfsstat -csnr

That is, print everything and reinitialize nothing.

Options

-c Display client information. Only the client side NFS and RPC information will be printed. Can be combined with the **-n** and **-r** options to print client NFS or client RPC information only.

-s Display server information. Works like the **-c** option above.

-n Display NFS information. NFS information for both the client and server side will be printed. Can be combined with the **-c** and **-s** options to print client or server NFS information only.

-r Display RPC information. Works like the **-n** option above.

-z Zero (reinitialize) statistics. Can be combined with any of the above options to zero particular sets of statistics after printing them. The user must have write permission on **/dev/kmem** for this option to work.

Files

/unix	system namelist
/dev/kmem	kernel memory

See Also

nfs(NADM)

nmountall, numountall

mount, unmount multiple file systems

Syntax

/etc/nmountall
/etc/numountall

Description

The *nmountall* command is used to mount NFS file systems according to entries in */etc/default/filesys* . It is strongly recommended that the NFS mount option, **bg**, be used for file systems which are automatically mounted during startup. This will prevent startup processing from hanging while trying to mount a file system from a very slow or dead server.

The *numountall* command causes all NFS mounted file systems to be unmounted. Processes which hold open files or have current directories on these file systems are killed by being sent a series of signals. The first signal sent is SIGHUP. One second later, SIGTERM, is sent. Finally, one second later, SIGKILL is sent.

These commands may be executed only by the super-user.

Files

The file system table format is as follows. Lines that being with the "#" character are considered comments and the fields are generally delimited with a blank space. Note that for an entry longer than one line, it is desirable to escape the new lines with backslashes to separate the fields. This has been done in our example for easier reading:

```
bdev=nfs2:/ \
# The first field is the remote directory name to be mounted.
#
        mountdir=/nfs/nfs2 \
# The second field is the mount point directory (which must already exist
#
        mount=prompt mountflags= \
# These fields direct NFS to prompt before mounting and specify
# no flags when the filesystem is mounted with mnt(NADM)
# or umnt(NADM).
#
        fsck=no fsckflags= rcfsck=no \
# These fields direct NFS not to clean and check the remote filesystem.
#
        desc="The Root Filesystem of machine NFS2" \
# The above field is a brief description of the mounted filesystem.
#
        rcmount=yes fstyp=NFS \
# The above fields specify that the filesystem should be
# mounted during NFS initialization and the filesystem type.
#
        nfsopts="soft,wsize=1024,rsize=1024,nosuid"
# These are NFS specific options,
# see mount(NS) for more information.
```

Note that empty lines in */etc/default/filesys* are ignored.

In practice, our example file-system-table entry might read:

```
bdev=nfs2:/  mountdir=/nfs/nfs2  \  mount=prompt
mountflags=  \  fsck=no  fsckflags=  rcfsck=no  \
        desc="The  Root  Filesystem  of  machine  NFS2"  \
        rcmount=yes  fstyp=NFS  \
        nfsopts="bg,soft,wsize=1024,rsize=1024,nosuid"
```

See Also

mount(NADM), umount(NADM), fuser(NADM),
and signal(S), filesys(F) in the *Programmer's Reference Manual* and the *User's Reference Manual* respectively.

Diagnostics

nmountall prints the mount commands that it will run before it runs them.

numountall prints the list of process-ids that it sent signals. The list of file systems which are being unmounted is also printed.

Notes

The information displayed in Column 3 will only appear if the file system was mounted as a read-only.

passmgmt

password files management

Syntax

passmgmt -a options name
passmgmt -m options name
passmgmt -d name

Description

The *passmgmt* command updates information in the password files. This command works with both **/etc/passwd** and **/etc/shadow**. If there is no shadow password file the changes done by *passmgmt* will go into **/etc/passwd**.

passmgmt -a adds an entry for user *name* to the login password files. This command does not create any directory for the new user and the new login remains locked (with the string LK in the *passwd* field) until the *passwd*(C) command is executed to set the password.

passmgmt -m modifies the entry for user *name* in the login password files. The name field in the **/etc/shadow** entry and all the fields (except the password field) in the **/etc/passwd** entry can be modified by this command. Only fields entered on the command line will be modified. If there is no **/etc/shadow** file, all modifications are made in **/etc/passwd**.

passmgmt -d deletes the entry for user *name* from the login password files. It will not remove any files that the user owns on the system; they must be removed manually.

name, the login name of the user, must be unique.

The following options are available:

-c *comment* A short description of the login. It is limited to a maximum of 128 characters and defaults to an empty field.

-h *homedir* Home directory of *name*. It is limited to a maximum of 256 characters and defaults to **/usr**/*name*.

-u *uid* UID of the *name*. This number must range from 0 to the maximum value for the system. It defaults to the next available UID greater than 100. Without the **-o** option, it enforces the uniqueness of a UID.

-o This option allows a UID to be non-unique. It is used only with the **-u** option.

-g *gid* GID of the *name*. This number must range from 0 to the maximum value for the system. The default is 1.

-s *shell* Login shell for *name*. It should be the full pathname of the program that will be executed when the user logs in. The maximum length of *shell* is 256 characters. The default is for this field to be empty and to be interpreted as **/bin/sh**.

-l *logname* This option changes the *name* to *logname* for the **-m** option only.

The total size of each login entry, whether existing or new, is limited to a maximum of 511 bytes in the password files.

Files

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

See Also

passwd(C)

Diagnostics

The *passmgmt* command exits with one of the following values:

0 SUCCESS.

1 Permission denied.

2 Invalid command syntax. Usage message of the **passmgmt** command will be displayed.

3 Invalid argument provided to option.

4 UID in use.

5 Inconsistent password files (e.g., *name* is in the **/etc/passwd** file and not in the **/etc/shadow** file, or vice versa).

6 Unexpected failure. Password files unchanged.

7 Unexpected failure. Password file(s) missing.

8 Password file(s) busy. Try again later.

9 *name* does not exist (if **-m** or **-d** is specified), already exists (if **-a** is specified), or *logname* already exists (if **-m -l** is specified).

NOTE

You cannot use a colon or <cr> because it will be interpreted as a field separator.

pcnfsd

pc-nfs authentication and print spooling daemon

Syntax

/etc/pcnfsd [**-d**] [**-s** *spooldir*]

Description

pcnfsd processes authentication and print spool requests from MS-DOS clients running Sun Microsystem's PC-NFS. Requests, and their responses, are forwarded through the RPC/XDR package. Upon receipt of an authentication request, *pcnfsd* consults the server's password file and verifies that the password sent from the MS-DOS client matches that on the local machine. An acceptance or rejection message is then sent back to the client. The account's uid/gid pair is also returned if authentication succeeds. If the account in question does not exist, then authentication fails.

Print spooling consists of two services: spooling initialization, and start print. When the server receives an initialization request, a path name for the print spool directory is assembled from *spooldir* and the client machine's name. This path is returned to the client. The client then NFS mounts this directory. When a spool file is ready to print, the start print request is sent to the server. The server then sends the file to the print spooling subsystem.

If the host handling *pcnfsd* service crashes, RPC timeout messages will be returned to the user when the above requests are generated.

Options

-d Turn on debugging mode. Status messages are returned to the console terminal.

-s *spooldir* *pcnfsd* uses *spooldir* instead of */usr/spool/lp/pcnfsd* as the base spooling directory. This option is only available when *pcnfsd* is run as a daemon.

See Also

lp(C), passwd(C), getpwent(S)

portmap

DARPA port to RPC program number mapper

Syntax

/etc/portmap

Description

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

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

See Also

rpcinfo(NADM)

Notes

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

rexd

RPC-based remote execution server

Syntax

/etc/rexd

Description

rexd is the RPC server for remote program execution. For non-interactive programs standard file descriptors are connected directly to TCP connections. Interactive programs involve pseudo-terminals, similar to the login sessions provided by *rlogin*(C). This daemon may use NFS to mount file systems specified in the remote execution request.

Files

/dev/ttyp*n*	pseudo-terminals used for interactive mode
/etc/passwd	authorized users
/tmp/rexd.log	if it exists, logs errors and events

See Also

mount(NADM), on(NC), rex(NS), exports(NF)

Diagnostics

Diagnostic messages are normally printed on the console, and returned to the requester.

Notes

Should be better access control.

rpcinfo

Report RPC information

Syntax

rpcinfo -p [host]
rpcinfo -u host program [version]
rpcinfo -t host program [version]

Description

rpcinfo makes an RPC call to an RPC server and reports what it finds.

Options

-p Probe the portmapper on *host*, and print a list of all registered RPC programs. If *host* is not specified, it defaults to the node name returned by *hostname*(NC).

-u Make an RPC call to procedure 0 of *program* on the specified *host* using UDP, and report whether a response was received.

-t Make an RPC call to procedure 0 of *program* on the specified *host* using TCP, and report whether a response was received.

-b Make an RPC broadcast to procedure 0 of the specified *program* and *version* using UDP and report all hosts that respond.

The *program* argument can be either a name or a number. If a *version* is specified, **rpcinfo** attempts to call that version of the specified *program*. Otherwise, **rpcinfo** attempts to find all the registered version numbers for the specified *program* by calling every registered version and also version 0, which is presumed not to exist. If it does exist, **rpcinfo** attempts to obtain this information by calling an extremely high version number instead.

Note that the version number is required for the **-b** option.

Examples

To show all the RPC services registered on a machine, use the command:

rpcinfo -p *machine_name*

Files

/etc/rpc names for RPC program numbers

See Also

rpc(NF), portmap(NADM)

showmount

Show all remote mounts

Syntax

/etc/showmount [**-a**] [**-d**] [**-e**] [host]

Description

showmount lists all the clients that have remotely mounted a filesystem from *host*. This information is maintained by the *mountd*(NADM) server on *host*, and is saved across crashes in the file **/etc/rmtab**. The default value for *host* is the node name returned by *hostname*(NC).

Options

-d List directories that have been remotely mounted by clients.

-a Print all remote mounts in the format

hostname:directory

where **hostname** is the name of the client, and **directory** is the root of the file system that has been mounted.

-e Print the list of exported file systems.

See Also

rmtab(NF), mountd(NADM), exports(NF)

Notes

If a client crashes, its entry will not be removed from the list until it reboots and unmounts the file system.

statd

Network status monitor

Syntax

/etc/statd [**-d** *debuglevel*]

Description

statd is an intermediate version of the status monitor. It interacts with *lockd*(NADM) to provide the crash and recovery functions for the locking services on NFS.

The *statd* preserves crash/recovery state in the */etc/sm* directory. The *record* file records the hostname of all currently monitored systems. The *recover* file records the hostname of all systems that have as yet not been notified of *statd*'s failure, and the *state* file records *statd*'s current version number.

Options

-d *debuglevel*
The *statd* command has extensive internal reporting capabilities. A level of 2 reports significant events. A level of 4 reports internal state and all status monitor request traffic. Level 4 is considered verbose.

Files

/etc/sm/record
/etc/sm/recover
/etc/sm/state

See Also

lockd(NADM), statmon(NF)

Notes

The crash of a site is only detected upon its recovery.

Commands

Network Commands (NC)

Intro	Introduction to ONC commands
hostname	Get name of current host
mnt	Mount and unmount a file system
on	Execute remote command
rpcgen	RPC Protocol Compiler

intro

introduction to NFS commands

Description

The commands in this section are tools for efficient use of NFS. You can use them to mount a filesystem, execute commands on remote systems, and compile RPC commands.

hostname

Set or print name of current host system

Syntax

hostname [nameofhost]

Description

The *hostname* command prints the name of the current host, as given before the "login" prompt. The super-user can set the hostname by giving an argument; this is usually done in the startup script **/etc/rc.local**.

See Also

gethostname(SLIB)

mnt, umnt

mount a filesystem

Syntax

/usr/bin/mnt [-urant] [directory **]**

/usr/bin/umnt *directory*

Description

mnt allows users other than the super-user to access the functionality of the *mount*(ADM) command to mount selected filesystems. The super-user can define how and when a filesystem mount is permitted via special entries in the */etc/default/filesys* file.

The filesystem requirements are the same as defined for *mount*(ADM).

umnt removes the mountable filesystem previously mounted in *directory* .

mnt is invoked from the */etc/rc* scripts with the **-r**, the **-n** and possibly the **-a** flag to mount filesystems when the system comes up multiuser. The **-a** flag is used when the system has autobooted. None of these flags should be specified during normal command line use.

The **-n** flag directs the system to mount all filesystems defined as *fstyp* "NFS" with *rcmount* set to "yes" in the */etc/default/filesys* file. Filesystems of this type should have *bdev* defined as follows:

bdev= hostname:/pathname

The *cdev* entry is not necessary if the filesystem is of type "NFS". *rcfsck* should be set to "no". As stated previously, *fstyp* must be "NFS" and *rcmount* must be set to "yes".

The **-t** flag displays the contents of /etc/default/filesys.

The **-u** flag forces *mnt* to behave like *umnt*.

Options

The following options can be defined in the */etc/default/filesys* entry for a filesystem:

bdev=/dev/*device* — Name of block device associated with the filesystem.

cdev=/dev/*device* — Name of character (raw) device associated with the filesystem.

mountdir=/*directory* — The directory the filesystem is to be mounted on.

desc=*name* — A string describing the filesystem.

passwd=*string* — An optional password prompted for at mount request time. Cannot be a simple string; must be in the format of /etc/passwd. (See **Notes**.)

fsck=**yes, no, dirty, prompt**
If yes/no, tells explicitly whether or not to run *fsck*. If dirty, *fsck* is run only if the filesystem requires cleaning. If prompt, the user is prompted for a choice. If no entry is given, the default value is dirty.

fsckflags=*flags* — Any flags to be passed to *fsck*.

rcfsck=**yes, no, dirty, prompt**
Similar to fsck entry, but only applies when the **-r** flag is passed.

maxcleans=*n* — The number of times to repeat cleaning of a dirty filesystem before giving up. If undefined, default is 4.

mount=**yes, no, prompt**
If yes or no, users are allowed or disallowed to mount the filesystem, respectively. If prompt, the user specifies whether the filesystem should be mounted.

rcmount=**yes, no, prompt**
If yes, the filesystem is mounted by */etc/rc2* when the system comes up multiuser. If no, the filesystem is never mounted by */etc/rc2*. With prompt, a query is displayed at boot time to mount the filesystem.

mountflags=*flags*	Any flags to be passed to *mount*.
prep=**yes**, **no**, **prompt**	Indicates whether any prepcmd entry should always be executed, never executed, or executed as specified by user.
prepcmd=*command*	An arbitrary shell command to be invoked immediately following password check and prior to running *fsck*.
init=**yes**, **no**, **prompt**	Indicates whether an initcmd entry should always be executed, never be executed, or executed as specified by user.
initcmd=*command*	An optional, arbitrary shell command to be invoked immediately following a successful mount.
fstyp=*type*	Defines the filesystem type. Available types are NFS, S51K, XENIX, and DOS.
nfsopts=*opts*	Defines NFS options for filesystems of type NFS. Available options are described in the **mount**(NADM) manual page.

Any entries containing spaces, tabs, or newlines must be contained in double quotes (").

The only mandatory entries in */etc/default/filesys* are **bdev** and **mountdir**. The **prepcmd** and **initcmd** options can be used to execute another command before or after mounting the filesystem. For example, **initcmd** could be defined to send mail to root whenever a given filesystem is mounted.

When invoked without arguments, *mnt* attempts to mount all filesystems that have the entries **mount=yes** or **mount=prompt**.

Examples

The following is a sample */etc/default/filesys* file:

```
bdev=/dev/root  cdev=/dev/rroot  mountdir=/  \
desc="The Root Filesystem"  rcmount=no  mount=no

bdev=/dev/u  cdev=/dev/ru  mountdir=/u  rcmount=yes  \
fsckflags=-y  desc="The User Filesystem"

bdev=/dev/x  cdev=/dev/rx  mountdir=/u  rcmount=no  \
mount=yes  fsckflags=-y  desc="The Extra Filesystem"
```

Of the examples above, only */x* is mountable by the user.

Files

/etc/default/filesys Filesystem data

See Also

mount(ADM), default(F)

Diagnostics

mnt will fail if the filesystem to be mounted is currently mounted under another name.

Busy filesystems cannot be unmounted with *umnt*. A filesystem is busy if it contains an open file or if a user's present working directory resides within the filesystem.

Notes

Some degree of validation is done on the filesystem, however it is generally unwise to mount corrupt filesystems.

In order to create a password for a filesystem, the system administrator must run the *passwd*(C) command using the **-f** option.

Value Added

mnt is an extension of AT&T System V provided by the Santa Cruz Operation.

on

Execute a command remotely

Syntax

on [**-i**] [**-n**] [**-d**] *host command* [*argument*] ...

Description

The *on* program is used to execute commands on another system, in an environment similar to that invoking the program. All environment variables are passed, and the current working directory is preserved. To preserve the working directory, the working file system must be either already mounted on the host or be exported to it. Relative path names will only work if they are within the current file system; absolute path names may cause problems.

Standard input is connected to standard input of the remote command, and standard output and standard error from the remote command are sent to the corresponding files for the *on* command.

Options

-i Interactive mode: use remote echoing and special character processing. This option is needed for programs that expect to be talking to a terminal. All terminal modes and window size changes are propagated.

-n No input: this option causes the remote program to get end-of-file when it reads from standard input, instead of passing standard input from the standard input of the *on* program. For example, **-n** is necessary when running commands in the background with job control.

-d Debug mode: print out some messages as work is being done.

See Also

rexd(NADM), exports(NF)

Diagnostics

unknown host	Host name not found
cannot connect to server	Host down or not running the server
can't find .	Problem finding the working directory
can't locate mount point	Problem finding current file system

Other error messages may be passed back from the server.

rpcgen

An RPC protocol compiler

Syntax

rpcgen *infile*
rpcgen -h [**-o** *outfile*] [*inputfile*]
rpcgen -c [**-o** *outfile*] [*infile*]
rpcgen -m [**-o** *outfile*] [*infile*]
rpcgen -l [**-o** *outfile*] [*infile*]
rpcgen [**-s** *transport*]* [**-o** *outfile]* [*infile*]

Description

rpcgen is a tool that generates C code to implement an RPC protocol. The input to *rpcgen* is a language similar to C known as RPCL (Remote Procedure Call Language). *rpcgen* is normally used as in the first syntax entry, where it takes an input file and generates four output files. If the *infile* is named *proto.x*, **rpcgen** generates a header file in *proto.h*, XDR routines in the file *proto_xdr.c*, server-side stubs in the file *proto_svc.c*, and client-side stubs in the file *proto_clnt.c*.

The other syntax examples shown above produce only one output file, rather than all four. The usage of each syntax example is described below.

The C-preprocessor, *cpp(CP)*, is run on all input files before they are actually interpreted by *rpcgen*, so all the cpp directives are legal within an *rpcgen* input file. For each type of output file, *rpcgen* defines a special cpp symbol for use by the *rpcgen* programmer:

RPC_HDR
    defined when compiling into header files
RPC_XDR
    defined when compiling into XDR routines
RPC_SVC
    defined when compiling into server-side stubs
RPC_CLNT
    defined when compiling into client-side stubs

In addition, *rpcgen* does a little preprocessing of its own. Any line beginning with '%' is passed directly into the output file, uninterpreted by *rpcgen*.

You can customize some of your XDR routines by leaving those data types undefined. For every data type that is undefined, *rpcgen* will assume that there exists a routine with the name **xdr_** prepended to the name of the undefined type.

It is highly recommended that you read the chapters on RPC and XDR in the *SCO NFS Programmer's Guide* before using this utility.

Options

-c Compile XDR routines.

-h Compile C data-definitions (a header file)

-o *outfile*
Specify the name of the output file. If none is specified, standard output is used (**-c**, **-h** and **-s** modes only).

-l Compile into a client-side stubs.

-s *transport*
Compile into server-side stubs, using the the given transport. The supported transports are **udp** and **tcp**. This option may be invoked more than once so as to compile a server that serves multiple transports.

-m
Compile into a server-side stubs, but do not produce a *main()* routine. This option is useful if you want to supply your own *main()*.

Usage

RPCL Syntax Summary

This summary of RPCL syntax, which is used for *rpcgen* input, is intended more for aiding comprehension than as an exact statement of the language.

Primitive Data Types

[unsigned] char
[unsigned] short
[unsigned] int
[unsigned] long
unsigned
float
double
void
bool

Except for the added boolean data-type **bool**, RPCL is identical to C.

rpcgen converts **bool** declarations to **int** declarations in the output header file (literally it is converted to a **bool_t**, which has been **#define**'d to be an **int**). Also, **void** declarations may appear only inside of **union** and **program** definitions. For those averse to typing the prefix **unsigned**, the abbreviations **uchar**, **ushort**, **uint** and **ulong** are available.

Declarations

RPCL allows only three kinds of declarations:

- *simple-declaration*

  type-name object-ident

  For example,

  ```
  long a;
  ```

- *pointer-declaration*

  type-name ***object-ident***

  For example,

  ```
  char *b;
  ```

- *vector-declaration*

  type-name object-ident[*size*]

  (*size* can be either an integer or a symbolic constant)

  For example,

  ```
  opaque c[10];
  ```

RPCL declarations contain both limitations and extensions with respect to **C**. The limitations are that you cannot declare multidimensional arrays or pointers-to-pointers in-line (You can still declare them though, using **typedef**). There are two extensions:

- Opaque data is declared as a vector as follows:

  opaque *object-ident* [*size*]

  In the protocol, this results in an object of *size* bytes. Note that this is *not* the same as a declaration of *size* characters, since XDR characters are 32-bits. Opaque declarations are compiled in the output header file into character array declarations of *size* bytes.

- Strings are special and are declared as a vector declaration:

  string *object-ident* [*max-size*]

  If *max-size* is unspecified, then there is essentially no limit to the maximum length of the string. String declarations get compiled into the following:

  char **object-ident*

Type Definitions

You need to use *rpcgen* to generate an XDR routine and/or header file that defines a type in an input file. For every *zetype* you define, *rpcgen* creates a corresponding XDR routine named *xdr_zetype* that is mandatory for creating RPC programs.

There are six ways to define a type:

type-definition:
typedef
enumeration-def
structure-def
variable-length-array-def
discriminated-union-def
program-def

The first three are very similar to their C namesakes. C does not have a formal type mechanism to define variable-length arrays and XDR unions are quite different from their C counterparts. Program definitions are not really type definitions, but they are useful nonetheless.

You may not nest XDR definitions. For example, the following will cause *rpcgen* to choke:

```
struct dontdoit {
        struct ididit {
                int oops;
        } sorry;
        enum ididitagain { OOPS, WHOOPS } iapologize;
};
```

Typedefs

An XDR **typedef** looks as follows:

typedef:
typedef *type-name object-ident* ;

The *object-ident* is the name of the new type, whereas the *type-name* part is the name of the type from which it is derived. For example,

```
typedef longa;
```

Enumeration Types

The syntax is:

enumeration-def:
enum *enum-ident* {
enum-list
};

enum-list:
enum-symbol-ident [= *assignment*]
enum-symbol-ident [= *assignment*] , *enum-list*

(*assignment* may be either an integer or a symbolic constant)

If there is no explicit assignment, then the implicit assignment is the value of the previous enumeration plus 1. If not explicitly assigned, the first enumeration receives the value 0.

Structures

structure-def:
struct *struct-ident* {
declaration-list
};

declaration-list:
declaration ;
declaration ; *declaration-list*

Variable-Length Arrays

variable-length-array-def:
array *array-ident* {
unsigned *length-identifer* ;
vector-declaration ;
};

A variable length array definition looks much like a structure definition. Here's an example:

```
array mp_int {
        unsigned len;
        short val[MAX_MP_LENGTH];
};
```

This is compiled into:

```
struct mp_int {
        unsigned len;
        short *val;
};
typedef struct mp_int mp_int;
```

Discriminated Unions

discriminated-union-def:
union *union-ident* **switch** (*discriminant-declaration*) **{**
case-list
[**default** : *declaration* ;]
};

case-list:
case *case-ident* : *declaration* ;
case *case-ident* : *declaration* ; *case-list*

discriminant-declaration:
declaration

The union definition looks like a cross between a C-union and a C-switch. For example:

```
union net_object switch (net_kind kind) {
case MACHINE:
        struct sockaddr_in sin;
case USER:
        int uid;
default:
        string whatisit;
};
```

Compiles into:

```
struct net_object {
        net_kind kind;
        union {
                struct sockaddr_in sin;
                int uid;
                char *whatisit;
        } net_object;
};
typedef struct net_object net_object;
```

Note that the name of the union component of the output struct is the same as the name of the type itself.

Program Definitions

program-def:
program *program-ident* **{**
version-list
} = *program-number* **;**

version-list:
version
version version-list
version:
version *version-ident* **{**
procedure-list
} = *version-number* **;**
procedure-list:
procedure-declaration
procedure-declaration procedure-list
procedure-declaration:
type-name procedure-ident **(** *type-name* **)** = *procedure-number* **;**

Program definitions look like nothing you've ever seen before, so we turn to an example to explain them. Suppose you wanted to create server that could get or set the date. Its declaration might look like this:

```
program DATE_PROG {
    version DATE_VERS {
        date DATE_GET(timezone) = 1;
        void DATE_SET(date) = 2;    /* Greenwich mean time */
    } = 1;
} = 100;
```

In the header file, this compiles into the following:

```
#define DATE_PROG 100
#define DATE_VERS 1
#define DATE_GET 1
#define DATE_SET 2
```

These **define**'s are intended for use by the client program to reference the remote procedures.

If you are using *rpcgen* to compile your server, then there are some important things for you to know. The server interfaces to your local procedures by expecting a C function with the same name as that in the program definition, but in all lowercase letters and followed by the version number. Here is the local procedure that implements DATE_GET:

```
date *  /* always returns a pointer to the results */
date_get_1(tz)
        timezone *tz; /* always takes a pointer to the arguments */
{
        static date d;  /* must be static! */

        /*
         * figure out the date
         * and store it in d
         */
        return(&d);
}
```

The name of the routine is the same as the **#define**'d name, but in all lowercase letters and followed by the version number. XDR will recursively free the argument after getting the results from your local procedure, so you should copy from the argument any data that you will need between calls. However, XDR neither allocates nor frees your results. You must take care of their storage yourself.

Make Inference Rules For Compiling XDR Headers

It is possible to set up suffix transformation rules in *make*(C) for compiling XDR routines and header files. The convention is that RPCL protocol files have the extension **.x**. The *make* rules to do this are:

```
.SUFFIXES: .x
.x.c:
        rpcgen -c $< -o $@

.x.h:
        rpcgen -h $< -o $@
```

Example

Consider the following program, **example** , which defines three data types:

```
const NFS_PORT        = 2059;
enum nfsstat {
       NFS_OK=0
};
struct gnumbers {
       long g_assets;
       long g_liabilities;
};
```

When you run *rpcgen* with no arguments, it generates a header file, **example.h** and an XDR file, **example_xdr.c.**

example.h

```
#define NFS_PORT 2059

enum nfsstat {
       NFS_OK = 0,
};
typedef enum nfsstat nfsstat;
bool_t xdr_nfsstat();


struct gnumbers {
       long g_assets;
       long g_liabilities;
};
typedef struct gnumbers gnumbers;
bool_t xdr_gnumbers();
```

example_xdr.c

```
#include <rpc/rpc.h>
#include "infile.h"


bool_t
xdr_nfsstat(xdrs, objp)
        XDR *xdrs;
        nfsstat *objp;
{
        if (!xdr_enum(xdrs, (enum_t *)objp)) {
                return (FALSE);
        }
        return (TRUE);
}


bool_t
xdr_gnumbers(xdrs, objp)
        XDR *xdrs;
        gnumbers *objp;
{
        if (!xdr_long(xdrs, &objp->g_assets)) {
                return (FALSE);
        }
        if (!xdr_long(xdrs, &objp->g_liabilities)) {
                return (FALSE);
        }
        return (TRUE);
}
```

See Also

SCO NFS Programmer's Guide

Notes

Name clashes can occur when using program definitions, since the apparent scoping does not really apply. Most of these can be avoided by giving unique names for programs, versions, procedures and types.

Nesting is not supported. As a workaround, structures can be declared at the top level and their names used inside other structures in order to achieve the same effect.

Commands

Network File Formats (**NF**)

Intro	Introduction to network file formats
exports	NFS file systems being exported
mnttab	Mounted file system table
rmtab	Remotely mounted file system table
rpc	RPC program number data base
statmon	Statd directory and file structures

intro

introduction to formats of files used by ONC commands

Description

This section outlines the formats of various files. The C struct declarations for the file formats are given where applicable. Usually, these structures can be found in header files under the directories **/usr/include/rpc**, **/usr/include/rpcsvc**, or **/usr/include/sys/fs/nfs**.

exports

NFS file systems being exported

Syntax

/etc/exports

Description

The file **/etc/exports** describes the file systems which are being exported to NFS clients. It is created by the system administrator using a text editor and processed by the *mount* request daemon *mountd*(NADM) each time a mount request is received.

The file consists of a list of file systems and the *netgroups*(NF) or machine names allowed to remote mount each file system. The file system names are left justified and followed by a list of names separated by white space. The names will be looked up in **/etc/netgroups** and then in **/etc/hosts**. A file system name with no name list following means export to everyone. A sharp sign (#) anywhere in the file indicates a comment extending to the end of the line it appears on. Lines beginning with white space are continuation lines.

Example

```
/usr     clients          # export to my clients
/usr/local                # export to the world
/usr2    paris peoria phoenix      # export to only these machines
```

Files

/etc/exports

Notes

The identification of the remote system is dependent on the local network transport mechanism employed.

See Also

mountd(NADM)

mnttab

Mounted file system table

Syntax

#include <mnttab.h>

Description

mnttab resides in directory **/etc** and contains a table of devices, mounted by the *mount*(NADM) command, in the following structure as defined by **<mnttab.h>**:

```
struct mnttab {
        char    mt_dev[32];
        char    mt_filsys[32];
        short   mt_ro_flg;
        time_t  mt_time;
        char    mt_fstyp[16];
        char    mt_mntopts[64];
};
```

Each entry is 150 bytes in length;

- the first 32 bytes are the null-padded name of the place where the *special file* is mounted;
- the next 32 bytes represent the null-padded root name of the mounted special file;
- the next 6 bytes contain the mounted *special file*'s read/write permissions and the date on which it was mounted;
- the following 16 bytes are the null-padded name of file system type;
- the remaining 64 bytes are the null-padded string of *mount* options.

The *mount* options are only used in the case of an NFS file system.

The maximum number of entries in *mnttab* is based on the system parameter **NMOUNT** located in **/etc/master.d/kernel,** which defines the number of allowable mounted special files.

See Also

mount(NADM), setmnt(ADM)

rmtab

Remotely mounted file system table

Description

rmtab resides in directory **/etc** and contains a record of all clients that have done remote mounts of file systems from this machine. Whenever a remote *mount* is done, an entry is made in the *rmtab* file of the machine serving up that file system. *umount* removes entries, if of a remotely mounted file system. The table is a series of lines of the form

hostname:directory

This table is used only to preserve information between crashes, and is read only by *mountd*(NADM) when it starts up. *mountd* keeps an in-core table, which it uses to handle requests from programs like *showmount*(NADM) and *shutdown*(ADM).

Files

/etc/rmtab

See Also

showmount(NADM), mountd(NADM), mount(NADM), shutdown(ADM)

Notes

Although the *rmtab* table is close to the truth, it is not always 100% accurate.

rpc

RPC program number data base

Syntax

/etc/rpc

Description

The *rpc* file contains user readable names that can be used in place of RPC program numbers. Each line has the following information:

name of server for the rpc program
rpc program number
aliases

Items are separated by any number of blanks and/or tab characters. A sharp sign (#) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Here is an example of the */etc/rpc* file from the Sun RPC Source distribution.

```
#
# rpc 87/12/02 3.9 RPCSRC
#
portmapper    100000    portmap sunrpc
rstat_svc                100001    rstatd rstat rup perfmeter
rusersd                  100002    rusers
nfs                      100003    nfsprog
mountd                   100005    mount showmount
walld                    100008    rwall shutdown
etherstatd    100010     etherstat
rquotad                  100011    rquotaprog quota rquota
sprayd                   100012    spray
3270_mapper100013
rje_mapper    100014
selection_svc 100015     selnsvc
database_svc 100016
rexd                     100017    rex
alis                     100018
sched                    100019
llockmgr                 100020
nlockmgr                 100021
x25.inr                  100022
```

statmon	100023	
status	100024	
bootparam	100026	
keyserv	100029	keyserver

Files

/etc/rpc

See Also

getrpcent(NS)

statmon, current, backup, state

Statd directory and file structures

Syntax

/etc/sm/record /etc/sm/recover, /etc/sm/state

Description

The **/etc/sm/record** and **/etc/sm/recover** plain text files generated by *statd*. Each hostname in **/etc/sm/record** represents the name of the machine to be monitored by the *statd* daemon. Each hostname in **/etc/sm/recover** represents the name of the machine to be notified by the *statd* daemon upon its recovery.

The **/etc/sm/state** file is generated by *statd* to record its version number. This version number is incremented each time a crash or recovery takes place.

See Also

statd(NADM), lockd(NADM)

Commands

Network System Services (NS)

Intro	Introduction to network system services
bindresvport	Bind a socket
dbm	Data base subroutines
getdomainname	Get/set name of current domain
gethostent	Get network host entry
getrpcent	Get RPC entry
getrpcport	Get RPC port number
kclt_create	Create client kernel handles
mount	Mount a file system
nfs_getfh	Get NFS file handle
nfs_svc	NFS daemons
rex	Remote execution protocol

intro

introduction to NFS system calls and error numbers

Syntax

#include <sys/errno.h>

Description

This section describes all of the socket system calls used in System V NFS. Some of these system call are accessible from the **RPC** library, *librpc*. The rest of the system calls were designed for specific purposes for specific programs. The system call interfaces are generally built into these programs. There are no new error numbers added for the support of the NFS system calls. Some of these system calls were not designed to return during normal operation. They were designed to give the kernel a user context to run in or to provide the kernel with a resource that is more easily allocated from the user level. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible return value. This is almost always -1; the individual descriptions specify the details.

As with normal arguments, all return codes and values from functions are of type integer unless otherwise noted. An error number is also made available in the external variable *errno*, which is not cleared on successful calls. Thus *errno* should be tested only after an error has occurred.

See the *intro (S)* manual page for standard error codes.

Files

/usr/lib/librpc.a

See Also

intro(S), perror(S)

List Of Functions

Name	*Appears on Page*	*Description*
kclt_create	kclt_creat(NS)	create kernel RPC client handles
nfs_getfh	nfs_getfh(NS)	return a file handle
nfs_svc	nfs_svc(NS)	NFS server daemon
async_daemon	nfs_svc(NS)	NFS block i/o daemon

bindresvport

bind a socket to a privileged IP port

Syntax

```
#include <sys/types.h>
#include <netinet/in.h>

bindresvport(sd, sin)
int sd;
struct sockaddr_in *sin;
```

Description

bindresvport is used to bind a socket descriptor to a privileged IP port, that is, a port number in the range 0-1023. The routine returns 0 if it is successful, otherwise -1 is returned and *errno* set to reflect the cause of the error.

Only root can bind to a privileged port; this call will fail for any other users.

dbminit, fetch, store, delete, firstkey, nextkey

Data base subroutines

Syntax

```
typedef struct {
        char *dptr;
        int dsize;
} datum;

dbminit(file)
char *file;

datum fetch(key)
datum key;

store(key, content)
datum key, content;

delete(key)
datum key;

datum firstkey()

datum nextkey(key)
datum key;

dbmclose()
```

Description

These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two file system accesses. The functions are obtained with the loader option **-ldbm**.

*key*s and *content*s are described by the *datum* typedef. A *datum* specifies a string of *dsize* bytes pointed to by *dptr*. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

Before a database can be accessed, it must be opened by *dbminit*. At the time of this call, the files *file***.dir** and *file***.pag** must exist. (An empty database is created by creating zero-length **.dir**and **.pag**files.)

Once open, the data stored under a key is accessed by *fetch* and data is placed under a key by *store*. A key (and its associated contents) is deleted by *delete*. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of *firstkey* and *nextkey*. *firstkey* will return the first key in the database. With any key *nextkey* will return the next key in the database. This code will traverse the data base:

for (key = firstkey(); key.dptr != NULL; key = nextkey(key))

A database may be closed by calling *dbmclose*. You must close a database before opening a new one.

Diagnostics

All functions that return an *int* indicate errors with negative values. A zero return indicates ok. Routines that return a *datum* indicate errors with a null (0) *dptr*.

Notes

The **.pag** file will contain holes so that its apparent size is about four times its actual content. Older systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (*cp*, *cat*, *tp*, *tar*, *ar*) without filling in the holes.

dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 1024 bytes). Moreover all key/content pairs that hash together must fit on a single block. *store* will return an error in the event that a disk block fills with inseparable data.

delete does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by *firstkey* and *nextkey* depends on a hashing function, not on anything interesting.

There are no interlocks and no reliable cache flushing; thus concurrent updating and reading is risky.

getdomainname, setdomainname

Get/set name of current domain

Syntax

```
getdomainname(name, namelen)
char *name;
int namelen;

setdomainname(name, namelen)
char *name;
int namelen;
```

Description

getdomainname returns the name of the domain for the current processor, as previously set by *setdomainname*. The parameter *namelen* specifies the size of the *name* array. The returned name is null-terminated unless insufficient space is provided.

setdomainname sets the domain of the host machine to be *name*, which has length *namelen*. This call is restricted to the super-user and is normally used only when the system is bootstrapped.

The purpose of domains is to enable two distinct networks that may have host names in common to merge. Each network would be distinguished by having a different domain name.

Return Value

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*.

Errors

The following errors may be returned by these calls:

[EFAULT]	The *name* parameter gave an invalid address.
[EPERM]	The caller was not the super-user. This error only applies to *setdomainname*.

Notes

Domain names are limited to 64 characters.

gethostent, gethostbyaddr, gethostbyname, sethostent, endhostent

Get network host entry

Syntax

#include <netdb.h>

struct hostent *gethostent()

struct hostent *gethostbyname(name)
char *name;

struct hostent *gethostbyaddr(addr, len, type)
char *addr; int len, type;

sethostent(stayopen)
int stayopen

endhostent()

Description

gethostent, *gethostbyname*, and *gethostbyaddr* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network host data base, **/etc/hosts**.

```
struct hostent {
        char    *h_name;        /* official host name *,
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* address type */
        int     h_length;       /* length of address */
        char    *h_addr;        /* address */
};
```

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 A pointer to the network address for the host. Host addresses are returned in network byte order.

gethostent reads the next line of the file, opening the file if necessary.

sethostent opens and rewinds the file. If the *stayopen* flag is non-zero, the host data base will not be closed after each call to *gethostent* (either directly, or indirectly through one of the other gethost calls).

endhostent closes the file.

gethostbyname and *gethostbyaddr* sequentially search from the beginning of the file until a matching host name or host address is found, or until EOF is encountered. Host addresses are supplied in network order.

Files

/etc/hosts

See Also

Diagnostics

Null pointer (0) returned on EOF or error.

Notes

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.

Use of this routine depends on the local network transport mechanism

getrpcent, getrpcbyname, getrpcbynumber

Get RPC entry

Syntax

#include <rpc/netdb.h>

struct rpcent *getrpcent()

struct rpcent *getrpcbyname(name)
char *name;

struct rpcent *getrpcbynumber(number)
int number;

setrpcent(stayopen)
int stayopen

endrpcent()

Description

getrpcent, *getrpcbyname*, and *getrpcbynumber* each return a pointer to an object with the following structure containing the broken-out fields of a line in the RPC program number data base, **/etc/rpc**.

```
struct rpcent {
    char *r_name;        /* name of server for this rpc program
    char **r_aliases;    /* alias list */
    long r_number;       /* rpc program number */
};
```

The members of this structure are:

r_name The name of the server for this RPC program.

r_aliases A zero-terminated list of alternate names for the RPC program.

r_number The RPC program number for this service.

The commands operate as follows:

getrpcent reads the next line of the file, opening the file if necessary.

setrpcent opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to *getrpcent* (either directly, or indirectly through one of the other getrpc calls).

endrpcent closes the file.

getrpcbyname and *getrpcbynumber* sequentially search from the beginning of the file until a matching RPC program name or program number is found, or until EOF is encountered.

Files

/etc/rpc
domainname/*rpc.bynumber*

See Also

rpc(NF), rpcinfo(NADM),

Diagnostics

Null pointer (0) returned on EOF or error.

Notes

All information is contained in a static area so it must be copied if it is to be saved.

getrpcport

Get RPC port number

Syntax

int getrpcport(host, prognum, versnum, proto)
char *host;
int prognum, versnum, proto;

Description

getrpcport returns the port number for version *versnum* of the RPC program *prognum* running on *host* and using protocol *proto*. It returns 0 if it cannot contact the portmapper, or if *prognum* is not registered. If *prognum* is registered but not with version *versnum*, it will return that port number.

kclt_create

Create kernel RPC client handles

Syntax

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

kclt_create(nfd, fds, trans, tsdu, pgm, vers)
int nfd;
int *fds;
int trans;
int tsdu;
ulong pgm;
ulong vers;
```

Description

kclt_create is used to create client handles for kernel RPC clients to use. Currently, there are two kernel RPC clients: NFS and the lock manager.

nfd is the number of file descriptors in the array pointed by *fds*. *nfd* controls the number of client handles which will be created using these file descriptors in conjunction with the rest of the arguments. *trans* is used as "transport identifier". It is intended to be used to differentiate between different transport and protocol implementations running concurrently on the host. Values for *trans* have not been defined yet, so *trans* should be set to 1. *tsdu* is the maximum transport service data unit that the transport can handle. *pgm* and *vers* are used when initializing the new client handles. The RPC call header is preserialized in the client handle for performance. *pgm* and *vers* are part of the RPC call header. In addition, *trans*, *pgm*, and *vers* are used internally to identify client handles for allocation.

See Also

nfsclnt(NADM)

mount

Mount a file system

Syntax

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

int mount (spec, dir, mflag, fstyp,
char *spec, *dir;
int mflag, fstyp;
caddr_t dataptr;
int datalen;
```

Description

mount requests that a removable file system contained on the block special file identified by *spec* be mounted on the directory identified by *dir*. *spec* and *dir* are pointers to path names. *fstyp* is the file system type number. The *sysfs*(S) system call can be used to determine the file system type number. If the MS_FSS flag bit of *mflag* is off, the file system type will default to root file system type. If the bit is on, then *fstyp* is used to indicate the file system type. Additionally, if the MS_DATA flag is on in *mflag* then *dataptr* and *datalen* are used to pass mount parameters to the system. If MS_DATA is off, or if either of *dataptr* and *datalen* is zero, it means that there is no additional data. In the normal case of a local mount, *dataptr* should be NULL. When mounting an NFS file system, *dataptr* should point to a structure that describes the NFS mount options.

Upon successful completion, references to the file *dir* will refer to the root directory on the mounted file system.

The low-order bit of *mflag* is used to control write permission on the mounted file system; if **1**, writing is forbidden, otherwise writing is permitted according to individual file accessibility.

mount may be invoked only by the super-user. It is intended for use only by the *mount*(NADM) utility.

mount will fail if one or more of the following are true:

[EPERM]	The effective user ID is not super-user.
[ENOENT]	Any of the named files does not exist.

[ENOTDIR] A component of a path prefix is not a directory.

[EREMOTE] *spec* is remote and can not be mounted.

[ENOLINK] *path* points to a remote machine and the link to that machine is no longer active.

[EMULTIHOP] Components of *path* require hopping to multiple remote machines.

[ENOTBLK] *spec* is not a block special device.

[ENXIO] The device associated with *spec* does not exist.

[ENOTDIR] *dir* is not a directory.

[EFAULT] *spec* or *dir* points outside the allocated address space of the process.

[EBUSY] *dir* is currently mounted on, is someone's current working directory, or is otherwise busy.

[EBUSY] The device associated with *spec* is currently mounted.

[EBUSY] There are no more mount table entries.

[EROFS] *spec* is write protected and *mflag* requests write permission.

[ENOSPC] The file system state in the super-block is not FsOKAY and *mflag* requests write permission.

[EINVAL] The super block has a bad magic number or the *fstyp* is invalid or *mflag* is not valid.

See Also

sysfs(S), fs(F), mount(NADM)

Diagnostics

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

nfs_getfh

Get NFS file handle

Syntax

```
#include <sys/fs/nfs.h>

nfs_getfh(fdes, fhp)
int fdes;
fhandle_t *fhp;
```

Description

nfs_getfh returns a file handle for the file open as file descriptor *fdes*. It is only used by the NFS mount daemon, and should not be used by users.

See Also

mountd(NADM)

nfs_svc, async_daemon

NFS daemons

Syntax

```
nfs_svc(tep, addr, tsdu, buf)
int tep;
int addr;
int tsdu;
char *buf;

async_daemon()
⟨PrevPg⟩.ft B
```

Description

These two system calls allow kernel processes to have a user context.

nfs_svc starts an NFS daemon listening on the transport endpoint *tep*. This transport endpoint is typically the file descriptor returned from a call to *t_open*(). Currently, the transport endpoint (in 4.2BSD terminology) must be AF_INET, and SOCK_DGRAM (protocol UDP/IP), but this is completely dependent on the local network transport implementation. In addition, the transport endpoint should be bound to the NFS internet port, 2049. *addr* is the maximum size of a remote address that the transport can handle. *buf* should be the address of a buffer of size **NFS_MAXDATA** (currently 8192 bytes) inside of the user level process. This buffer is used by the local file system *getdents*(S) implementation. The system call will return only if the process is killed.

async_daemon implements the NFS daemon that handles asynchronous I/O for an NFS client. The system call never returns.

rex

Remote execution protocol

Syntax

#include <rpcsvc/rex.h>

Description

This server will execute commands remotely. The working directory and environment of the command can be specified, and the standard input and output of the command can be arbitrarily redirected. An option is provided for interactive I/O for programs that expect to be running on terminals. Note that this service is only provided with the TCP transport.

RPC Info

program number:
REXPROG

xdr routines:

```
int xdr_rex_start(xdrs, start);
        XDR *xdrs;
        struct rex_start *start;

int xdr_rex_result(xdrs, result);
        XDR *xdrs;
        struct rex_result *result;

int xdr_rex_ttymode(xdrs, mode);
        XDR *xdrs;
        struct rex_ttymode *mode;

int xdr_rex_ttysize(xdrs, size);
        XDR *xdrs;
        struct ttysize *size;
```

procs:

REXPROC_START
: Takes rex_start structure, starts a command executing, and returns a rex_result structure.

REXPROC_WAIT
: Takes no arguments, waits for a command to finish executing, and returns a rex_result structure.

REXPROC_MODES
: Takes a rex_ttymode structure, and sends the tty modes.

REXPROC_WINCH
: Takes a ttysize structure, and sends window size information.

versions:

REXVERS_ORIG
: Original version

structures:

```
struct B_sgttyb {
        char    bsg_ispeed;     /* input speed     */
        char    bsg_ospeed;     /* output speed    */
        char    bsg_erase;      /* erase character */
        char    bsg_kill;       /* kill character  */
        short   bsg_flags;
};

struct tchars {
        char    t_intrc;        /* interrupt    */
        char    t_quitc;        /* quit         */
        char    t_startc;       /* start output */
        char    t_stopc;        /* stop output  */
        char    t_eofc;         /* end-of-file  */
        char    t_brkc;         /* input delimiter
                                   (like nl) */
};

struct ltchars {
        char    t_suspc;        /* stop process signal */
        char    t_dsuspc;       /* delayed stop process
                                   signal */
        char    t_rprntc;       /* reprint line */
        char    t_flushc;       /* flush output (toggles) */
        char    t_werasc;       /* word erase */
        char    t_lnextc;       /* literal next character */
};
```

```
#define REX_INTERACTIVE  1  /* Interactive mode */
struct rex_start {
char    **rst_cmd;      /* list of command and args     */
char    *rst_host;      /* working directory host name  */
char    *rst_fsname;    /* directory file system name   */
char    *rst_dirwithin; /* directory within file system */
char    **rst_env;      /* list of environment          */
ushort  rst_port0;      /* port for stdin               */
ushort  rst_port1;      /* port for stdin               */
ushort  rst_port2;      /* port for stdin               */
ulong   rst_flags;      /* options - see #defines above */
};

struct rex_result {
int     rlt_stat;       /* integer status code         */
char    *rlt_message;   /* string message for humans   */
};

struct rex_ttymode {
struct B_sgttyb basic; /* Berkeley unix tty flags  */
struct tchars more;     /* interrupt, kill etc.     */
struct ltchars yetmore;/* special Berkeley chars   */
ulong andmore;          /* and Berkeley modes       */
};

struct ttysize {
int     ts_lines;       /* no. lines on terminal    */
int     ts_cols;        /* no. columns on terminal  */
};
```

See Also

on(NC), rexd(NADM)