SCO™ TCP/IP

Derived from

LACHMAN™ SYSTEM V STREAMS TCP

User's Guide

The Santa Cruz Operation™

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: 12/4/89

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 printfile
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.

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.

---

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.

Index