SCO™ TCP/IP

Derived from

LACHMAN™ SYSTEM V STREAMS TCP

Administrator'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: 12/1/89

Contents

Chapter 1

Network Administration

Introduction

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

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

Kernel Configuration

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

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

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

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

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

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

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

All drivers must have references in the following files:

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

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

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

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

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

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

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

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

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

Runtime Configuration of STREAMS Drivers

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

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

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

Cloning Drivers with One Major Number per Interface

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

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

To add a second interface, add the following line:

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

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

Cloning Drivers Using unit select or DL_ATTACH

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

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

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

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

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

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

Non-Cloning Drivers

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

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

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

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

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

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

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

Setting Interface Parameters

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

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

Local Subnetworks

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

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

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

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

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

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

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

Internet Broadcast Addresses

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

Routing

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

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

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

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

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

netstat(TC) program displays routing table contents as well as various routing-oriented statistics. The following example displays the contents of the routing tables:

```
netstat -r
```

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

```
netstat -r -s
```

Using UNIX System Machines as Gateways

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

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

Network Servers

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

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

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

Network Databases

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

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

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

The /etc/hosts.equiv File

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

There are two ways to establish user equivalence:

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

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

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

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

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

```
Admin   Test1
```

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

```
Admin   Test1
```

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

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

The /etc/ftpusers File

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

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

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

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

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

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

```
uucp
root
```

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

Network Tuning and Troubleshooting

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

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

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

STREAMS Tuning

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

Active Connections Display

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

netstat -a

Active Internet connections (including servers) are as follows:

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

Descriptions of the Display Headings

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

Interfaces

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

netstat -i

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

Descriptions of the Display Headings

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

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

Routing Tables

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

netstat -r

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

Descriptions of the Display Headings

The information displayed for each route is as follows.

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

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

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

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

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

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

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

Statistics Display

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

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

netstat -s

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

(Continued on next page.)

(Continued)

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

(Continued on next page.)

(*Continued*)

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

Chapter 2

Introduction to sendmail

Introduction

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

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

Communicating with sendmail

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

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

These methods are discussed in the sections that follow.

User Interface Program

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

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

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

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

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

SMTP over Pipes

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

SMTP over a Berkeley-Style Socket

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

Overview of sendmail Operation

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

Argument Processing and Address Parsing

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

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

Collecting Messages

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

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

Delivering Messages

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

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

Queueing for Retransmission

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

Return to Sender

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

Editing the Message Header

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

The Configuration File

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

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

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

Sendmail Implementation

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

Sendmail and Arguments

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

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

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

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

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

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

Mailing to Files and Programs

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

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

Aliasing, Forwarding and Including Mail

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

- by aliasing
- by forwarding
- by inclusion

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

Aliasing

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

Forwarding

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

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

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

will use a different incoming mailer.

Including

The syntax for including a file is:

```
:include: pathname
```

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

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

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

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

Collecting Messages

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

- the message header
- the message body

The two parts are separated by a blank line.

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

field-name: field-value

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

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

Delivering Messages

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

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

Queued Messages

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

Configuration

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

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

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

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

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

Macros

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

Header Declarations

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

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

Mailer Declarations

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

Rules for Rewriting an Address

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

```
ucsfcgl!tef
```

might be mapped into:

```
tef@ucsfcgl.UUCP
```

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

Setting Options

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

Comparing sendmail with Other Mail Programs

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

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

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

Comparing sendmail with delivermail

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

- Configuration information is not compiled in. This change simplifies many of the problems of moving to other machines. It also simplifies debugging of new mailers.

- Address parsing is more flexible. For example, **delivermail** only supported one gateway to any network, whereas **sendmail** can be sensitive to host names and reroute to different gateways.

- The forward and include features of **sendmail** eliminate the requirement that the system alias file be writable by any user (or that an update program be written, or that the system administration make all changes).

- The **sendmail** program supports message batching across networks when a message is being sent to multiple recipients.

- A mail queue is provided in **sendmail.** Mail that cannot be delivered immediately but can potentially be delivered later is stored in this queue for a later retry. The queue also provides a buffer against system crashes; after the message has been collected, it may be reliably redelivered even if the system crashes during the initial delivery.

- The **sendmail** program uses the networking support of 4.2BSD, which provides a direct interface to networks such as ARPANET or Ethernet using SMTP (the Simple Mail Transfer Protocol) over a TCP/IP connection.

Comparing sendmail with MMDF

The Multichannel Memorandum Distribution Facility (MMDF) spans a wider problem set than **sendmail**. For example, the domain of MMDF includes a "phone network" mailer, whereas **sendmail** calls on pre-existing mailers in most cases.

MMDF and **sendmail** both support aliasing, customized mailers, message batching, automatic forwarding to gateways, queueing, and retransmission. MMDF supports two-stage timeout, which **sendmail** does not support. (MMDF uses two-stage timeout when routing mail through machines to users. If a message can't be forwarded to a particular machine or to a particular user on a machine, a warning is sent back to the mail message sender. This is stage 1. At some future time (configurable by the administrator), the message is relayed again. If it fails, a failure message is returned to the sender, and MMDF makes no further attempts to resend the original message. This is stage 2.)

The configuration for MMDF is compiled into the code.

Since MMDF does not consider backwards compatibility as a design goal, the address parsing is simpler but much less flexible.

It is somewhat harder to integrate a new channel into MMDF. In particular, MMDF must know the location and format of host tables for all channels, and each channel must speak a special protocol. This allows MMDF to do additional verification (such as verifying host names) at submission time.

MMDF strictly separates the submission and delivery phases. Although **sendmail** has the concept of each of these stages, they are integrated into one program, whereas in MMDF they are split into two programs.

Sendmail and the Message-Processing Module

The Message Processing Module (MPM) matches **sendmail** closely in terms of its basic architecture. However, like MMDF, the MPM includes the network interface software as part of its domain.

MPM also postulates a duplex channel to the receiver, as does MMDF. This allows simpler handling of errors by the mailer than is possible in **sendmail**. When a message queued by **sendmail** is sent, any errors must be returned to the sender by the mailer itself. Both MPM and MMDF mailers can return an immediate error response, and a single error processor can create an appropriate response.

MPM prefers passing the message as a structured object, with {type, length, value} triples. Such a convention requires a much higher degree of cooperation between mailers than is required by **sendmail**. MPM also assumes a universally agreed-upon internet name space (with each address in the form of a net-host-user tuple), which **sendmail** does not.

Chapter 3

Installing and Operating sendmail

Introduction

The **sendmail** program implements a general purpose internetwork mail-routing facility under the UNIX operating system. It is not tied to any one transport protocol. Its function may be likened to a crossbar switch, relaying messages from one domain into another. Included as part of this process, it can do a limited amount of message-header editing to put the message into a format that is appropriate for the receiving domain. All of this is done under the control of a configuration file.

Due to the requirements of flexibility for **sendmail**, the configuration file can seem somewhat unapproachable. However, there are only a few basic configurations for most sites, for which standard configuration files have been supplied. Most other configurations can be built by adjusting existing configuration files incrementally.

Although **sendmail** is intended to run without the need for monitoring, it has a number of features that may be used to monitor or adjust the operation under unusual circumstances.

Basic Installation

There are two basic steps to installing **sendmail.** They are:

- building the configuration table
- installing the software

The configuration table is a file that **sendmail** reads when it starts up. This file describes the mailers that sendmail knows about, how to parse addresses, how to rewrite the message header, and the settings of various options. Although the configuration table is quite complex, a configuration can usually be built by adjusting an existing off-the-shelf configuration. The second step is performing the actual installation. This means creating the necessary files and so on.

The remainder of this section describes the installation of **sendmail.** The description assumes that you can use one of the existing configurations and that the standard installation parameters are acceptable.

Off-the-Shelf Configurations

Several sample configuration files are included with this release. They are found in the directory */usr/local/lib/sendmail.* The makefile in the **cf** directory makes two files, **node.cf** and **relay.cf. Node.cf** is the configuration to use on a host that is not a central mail router. **Relay.cf** should be used on a major mail relay machine in your installation.

Once these variables are changed, the file is now ready for installation as **/usr/lib/sendmail.cf.**

The configuration file you need should be copied to a file with the same name as your system, as in the following example:

```
cp relay.cf laidbak.cf
```

There are some variables that need to be changed in both files, and the **Tune** script in the **cf** directory can be used to take care of this. These variables are as follows:

Tunable Parameters	
Parameter	Value
ZZHOST	Host Name
ZZRELAY	Mail Relay Host Name
ZZDOMAIN	Your subdomain, *e.g.* Lachman.
ZZDOM	Your domain, *e.g.* COM.

You can use the **Tune** script to change these variables. **Tune** takes the qualified hostname and the relay as arguments. It splits the hostname up to generate ZZHOST, ZZDOMAIN, and ZZDOM.

This file is now ready for installation as */usr/lib/sendmail.cf*.

Installation

For details about how to install the sendmail software, see the *TCP/IP Runtime Release and Installation Notes.*

Quick Configuration Startup

A fast version of the configuration file can be set up by using the **-bz** flag, as in the following example:

```
/usr/lib/sendmail -bz
```

This creates the file */usr/lib/sendmail.fc* (frozen configuration). This file is an image of **sendmail**'s data space after reading in the configuration file. If this file exists, it is used instead of */usr/lib/sendmail.cf*. The *sendmail.fc* file must be rebuilt manually every time *sendmail.cf* is changed.

The frozen configuration file will be ignored if a **-C** flag is specified or if **sendmail** detects that it is out of date. However, the heuristics are not strong, and so this should not be trusted.

The System Log

The system log is entered in the file */usr/adm/syslog*.

Format

Each line in the system log consists of a timestamp, the name of the machine that generated it (for logging from several machines over the ethernet), the word "sendmail," and a message.

Levels

A large amount of information can be logged. The log is arranged as a succession of levels. At the lowest level, only extremely strange situations are logged. At the highest level, even the most mundane and ininteresting events are recorded for posterity. As a convention, log levels under ten are consiedered "useful;" log levels above ten are usually for debugging purposes.

The Mail Queue

The mail queue should be processed transparently. However, you may find that manual intervention is sometimes necessary. For example, if a major host is down for a period of time the queue may become clogged. Although **sendmail** ought to recover gracefully when the host comes up, you may find performance unacceptably bad in the meantime.

Printing the Queue

The contents of the queue can be printed using the **mailq** command (or by specifying the **-bp** flag to **sendmail**):

mailq

This produces a listing of the queue identifiers, the size of the message, the date the message entered the queue, and the sender and recipients.

Format of Queue Files

All queue files have the form *x*f*AA99999*, where *AA99999* is the ID for this file and *x* is a type. The types are:

d — The data file. The message body (excluding the header) is kept in this file.

l — The lock file. If this file exists, the job is currently being processed, and a queue run will not process the file. For that reason, an extraneous lf file can cause a job to seem to disappear. (It will not even time out!)

n — This file is created when an ID is being created. It is a separate file to ensure that no mail can ever be destroyed due to a race condition. It should exist for no more than a few milliseconds at any given time.

q — The queue control file. This file contains the information necessary to process the job.

t — A temporary file. This is an image of the qf file when it is being rebuilt. It should be renamed to a qf file very quickly.

x A transcript file, existing during the life of a session, showing everything that happens during that session.

The qf file is structured as a series of lines, each beginning with a code letter. The lines are as follows:

D The name of the data file. There can only be one of these lines.

H A header definition. There can be any number of these lines. The order is important: it represents the order in the final message. This uses the same syntax as header definitions in the configuration file.

R A recipient address. This will normally be completely aliased, but is actually realiased when the job is processed. There will be one line for each recipient.

S The sender address. There can only be one of these lines.

E An error address. If any such lines exist, they represent the addresses that should receive error messages.

T The job creation time. This is used to compute how long a job remains in the queue undelivered, before being returned to the sender.

P The current message priority. This is used to order the queue. Higher numbers mean lower priorities. The priority changes as the message sits in the queue. The initial priority depends on the message class and the size of the message.

M A message. This line is printed by the **mailq** command, and is generally used to store status information. It can contain any text.

As an example, the following is a queue file sent to "mckee@calder" and "wnj:"

```
DdfA13557
Seric
T404261372
P132
Rmckee@calder
Rwnj
H?D?date: 23-Oct-82 15:49:32-PDT (Sat)
H?F?from: eric (Eric Allman)
H?x?full-name: Eric Allman
Hsubject: this is an example message
Hmessage-id: <8209232249.13557@UCBARPA.BERKELEY.ARPA>
Hreceived: by UCBARPA.BERKELEY.ARPA (3.227 [10/22/82])
       id A13557; 23-Oct-82 15:49:32-PDT (Sat)
HTo: mckee@calder, wnj
```

This shows the name of the data file, the person who sent the message, the submission time (in seconds since January 1, 1970), the message priority, the message class, the recipients, and the headers for the message.

Forcing the Queue

The **sendmail** program should run the queue automatically at intervals. The algorithm is to read and sort the queue, and then to attempt to process all jobs in order. When it attempts to run the job, **sendmail** first checks to see if the job is locked. If so, it ignores the job.

There is no attempt to ensure that only one queue processor exists at any time, since there is no guarantee that a job cannot take forever to process. Due to the locking algorithm, it is impossible for one job to freeze the queue. However, an uncooperative recipient host or a program recipient that never returns can accumulate many processes in your system. Unfortunately, there is no way to resolve this without violating the protocol.

In some cases, you may find that if a major host goes down for a couple of days, this can create a prohibitively large queue. This situation will cause **sendmail** to spend an inordinate amount of time sorting the queue. This situation can be fixed by moving the queue to a temporary place and creating a new queue. The old queue can be run later, when the offending host returns to service.

To do this, it is acceptable to move the entire queue directory:

```
cd /usr/spool
mv mqueue omqueue; mkdir mqueue; chmod 777 mqueue
```

You should then kill the existing daemon (since it will still be processing in the old queue directory) and create a new daemon.

To run the old mail queue, run the following command:

```
/usr/lib/sendmail -oQ/usr/spool/omqueue -q
```

The **-oQ** flag specifies an alternate queue directory, and the **-q** flag says just to run every job in the queue. If you have a tendency toward voyeurism, you can use the -v flag to watch what is going on.

When the queue is finally emptied, you can remove the directory:

```
rmdir /usr/spool/omqueue
```

The Alias Database

The alias database exists in two forms. One is a text form, maintained in the file */usr/lib/aliases*. The aliases are of the form

name: *name1, name2, ...*

Only local names can be aliased. For example:

```
eric@mit-xx: eric@berkeley.EDU
```

will not have the desired effect. Aliases can be continued by starting any continuation line with a space or a tab. Blank lines and lines beginning with a pound sign (#) are comments.

The second form is processed by the **dbm**(S) library. This form is in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag*. This is the form that **sendmail** actually uses to resolve aliases. This technique is used to improve performance.

Rebuilding the Alias Database

The DBM version of the database can be rebuilt explicitly by executing the command:

newaliases

This is equivalent to giving **sendmail** the **-bi** flag:

/usr/lib/sendmail -bi

If the "D" option is specified in the configuration, **sendmail** will rebuild the alias database automatically, if possible, when it is out of date. It will do this under either or both of the following conditions:

- The DBM version of the database is mode 666.
- **sendmail** is running setuid to root.

Auto-rebuild can be dangerous on heavily loaded machines with large alias files; if it might take more than five minutes to rebuild the database, there is a chance that several processes will start the rebuild process simultaneously.

Potential Problems

There are a number of problems that can occur with the alias database. They all result when a **sendmail** process accesses the DBM version while it is only partially built. This can happen under two circumstances: either one process accesses the database while another process is rebuilding it, or the process rebuilding the database dies (due to being killed or a system crash) before completing the rebuild.

The **sendmail** program includes two techniques to try to relieve these problems. First, it ignores interrupts while rebuilding the database; this avoids the problem of someone aborting the process and leaving a partially rebuilt database. Second, at the end of the rebuild it adds an alias of the form:

```
@: @
```

(Note that this is not normally legal.) Before **sendmail** will access the database, it checks to ensure that this entry exists. The **sendmail** program will wait for this entry to appear, at which point it will force a rebuild itself.

List Owners

If an error occurs on sending to a certain address, say "*x*," **sendmail** will look for an alias of the form "owner-*x*" to receive the errors. This is typically useful for a mailing list where the submitter of the list has no control over the maintenance of the list itself; in this case the list maintainer would be the owner of the list. For example:

```
unix-wizards: eric@ucbarpa, wnj@monet, nosuchuser,
      sam@matisse
owner-unix-wizards: eric@ucbarpa
```

This would cause "eric@ucbarpa" to get the error that will occur when someone sends to unix-wizards, due to the inclusion of "nosuchuser" on the list.

Per-User Forwarding (.forward Files)

As an alternative to the alias database, any user can put a file with the name *.forward* in his or her home directory. If this file exists, **sendmail** redirects mail for that user to the list of addresses listed in the *.forward* file. For example, if the home directory for user mckee has a *.forward* file with contents:

```
mckee@ernie
kirk@calder
```

then any mail arriving for "mckee" will be redirected to the specified accounts.

Special Header Lines

Several header lines have special interpretations defined by the configuration file. Others have interpretations built into **sendmail** that cannot be changed without changing the code. These built-ins are described here.

Return-Receipt-to:

If this header is sent, a message will be sent to any specified addresses when the final delivery is completed, that is, when successfully delivered to a mailer with the **l** flag (local delivery) set in the mailer descriptor.

Errors-To:

If errors occur anywhere during processing, this header will cause error messages to go to the listed addresses rather than to the sender. This is intended for mailing lists.

Apparently-To:

If a message comes in with no recipients listed in the message (in a To:, Cc:, or Bcc: line), then **sendmail** will add an "Apparently-To:" header line for any recipients it is aware of. This is not put in as a standard recipient line to warn any recipients that the list is not complete.

Arguments

Some important arguments of the **sendmail** program are described here.

Queue Interval

The amount of time between forking a process to run through the queue is defined by the **-q** flag. If you run in mode **f** or **a,** this can be relatively large, since it will only be relevant when a host that was down comes back up. If you run in **q** mode, it should be relatively short, since it defines the maximum amount of time that a message may sit in the queue.

Daemon Mode

If you allow incoming mail over an IPC connection, you should have a daemon running. This should be set by your */etc/rc* file using the **-bd** flag. The **-bd** flag and the **-q** flag may be combined in one call:

```
/usr/lib/sendmail -bd -q30m
```

Forcing the Queue

In some cases, you may find that the queue has gotten clogged for some reason. You can force a queue run using the **-q** flag (with no value). It is entertaining to use the **-v** flag (verbose) when this is done, to watch what happens:

```
/usr/lib/sendmail -q -v
```

Debugging

There is a fairly large number of debug flags built into **sendmail**. Each debug flag has a number and a level, where higher levels cause more information to be printed out. The convention is that levels greater than nine are not required. They print out so much information that you would not normally want to see them, except for debugging that particular piece of code. Debug flags are set using the **-d** option. The syntax is:

debug-flag:	**-d** debug-list
debug-list:	debug-option [, debug-option]
debug-option:	debug-range [. debug-level]
debug-range:	integer | integer - integer
debug-level:	integer

where spaces are for reading ease only. For example,

-d12	Set flag 12 to level 1
-d12.3	Set flag 12 to level 3
-d3-17	Set flags 3 through 17 to level 1
-d3-17.4	Set flags 3 through 17 to level 4

For a complete list of the available debug flags you will have to look at the code. (They are too dynamic to keep this documentation up to date.)

Trying a Different Configuration File

An alternative configuration file can be specified using the **-C** flag. The following example uses the configuration file *test.cf* instead of the default */usr/lib/sendmail.cf*.

```
/usr/lib/sendmail -Ctest.cf
```

If the **-C** flag has no value, it defaults to *sendmail.cf* in the current directory.

Changing the Values of Options

Options can be overridden using the **-o** flag. For example:

```
/usr/lib/sendmail -oT2m
```

sets the **T** (timeout) option to two minutes for this run only.

Tuning

There are a number of configuration parameters you may want to change, depending on the requirements of your site. Most of these are set using an option in the configuration file. For example, the line "OT3d" sets option "T" to the value "3d" (three days).

Most of these options default appropriately for most sites. However, sites having very high mail loads may find they need to tune them as appropriate for their mail load. In particular, sites experiencing a large number of small messages, many of which are delivered to many recipients, may find that they need to adjust the parameters dealing with queue priorities.

Timeouts

All time intervals are set using a scaled syntax. For example, "10m" represents ten minutes, whereas "2h30m" represents two-and-a-half hours. The full set of scales is:

s	seconds
m	minutes
h	hours
d	days
w	weeks

The argument to the **-q** flag specifies how often a subdaemon will run the queue. This is typically set to between fifteen minutes and one hour.

Read Timeouts

It is possible to time out when reading the standard input or when reading from a remote SMTP server. Technically, this is not acceptable within the published protocols. However, it might be appropriate to set it to something large (such as an hour) in certain environments. This will reduce the chance of large numbers of idle daemons piling up on your system. This timeout is set using the **r** option in the configuration file.

Message Timeouts

After sitting in the queue for a few days, a message will time out. This means that if a message cannot be delivered for some reason, it is returned to the sender. This ensures that at least the sender is aware that the message was not sent. The timeout is typically set to three days. This timeout is set using the **T** option in the configuration file.

The time of submission is set in the queue, rather than the amount of time left until timeout. As a result, you can flush messages that have been hanging for a short period by running the queue with a short message timeout. For example:

```
/usr/lib/sendmail -oT1d -q
```

will run the queue and flush anything that is one day old.

Forking During Queue Runs

When the **Y** option is set, **sendmail** will fork before each individual message while running the queue. This prevents **sendmail** from consuming large amounts of memory, and so it may be useful in memory-poor environments. However, if the **Y** option is not set, **sendmail** will keep track of hosts that are down during a queue run, which can improve performance dramatically.

Queue Priorities

Every message is assigned a priority when it is first instantiated, consisting of the message size (in bytes) offset by the message class multiplied by the "work class factor" and the number of recipients multiplied by the "work recipient factor." The priority plus the creation time of the message (in seconds since January 1, 1970) are used to order the queue. Higher numbers for the priority mean that the message will be processed later, when running the queue.

The message size is included so that large messages are penalized relative to small messages. The message class allows users to send high-priority messages by including a "Precedence:" field in their message; the value of this field is looked up in the **P** lines of the configuration file. Since the number of recipients affects the size of load a message presents to the system, this is also included into the priority.

The recipient and class factors can be set in the configuration file by using the **y** and **z** options respectively. They default to 1000 (for the recipient factor) and 1800 (for the class factor). The initial priority is:

```
pri = size - (class * z) + (nrcpt * y)
```

(Remember, higher values for this parameter actually mean that the job will be treated with lower priority.)

The priority of a job can also be adjusted each time it is processed (that is, each time an attempt is made to deliver it) using the "work time factor," set by the **Z** option. This is added to the priority, so it normally decreases the precedence of the job, on the grounds that jobs that have failed many times will tend to fail again in the future.

Delivery Mode

There are a number of delivery modes for **sendmail**, and they are set by the **d** configuration option. These modes specify how quickly mail will be delivered. Legal modes are:

- i deliver interactively (synchronously)
- b deliver in background (asynchronously)
- q queue only (do not deliver)

There are tradeoffs. Mode "i" passes the maximum amount of information to the sender, but is hardly ever necessary. Mode "q" puts the minimum load on your machine, but means that delivery may be delayed for up to the queue interval. Mode "b" is probably a good compromise. However, this mode can cause large numbers of processes if you have a mailer that takes a long time to deliver a message.

File Modes

There are several files involved with **sendmail** that can have a number of modes. The modes depend on the functionality you want and the level of security you require.

To suid or not to suid?

The **sendmail** program can safely be made setuid to root. At the point where it is about to **exec**(S) a mailer, it checks to see if the userid is zero. If so, it resets the userid and groupid to a default (set by the **u** and **g** options). (This can be overridden by setting the S flag to the mailer for mailers that are trusted and must be called as root.) However, this will cause mail processing to be accounted to root rather than to the user sending the mail.

Temporary File Modes

The mode of all temporary files that **sendmail** creates is determined by the **F** option. Reasonable values for this option are 0600 and 0644. If the more permissive mode is selected, it will not be necessary to run **sendmail** as root at all (even when running the queue).

Should My Alias Database Be Writable?

The database that **sendmail** actually uses is represented by two files. These files are:

- *aliases.dir*
- *aliases.pag*

Both files are located in */usr/lib*. The mode on these files should match the mode on */usr/lib/aliases*. If *aliases* is writable and the DBM files (*aliases.dir* and *aliases.pag*) are not, users will be unable to reflect their desired changes through to the actual database. However, if *aliases* is read-only and the DBM files are writable, a slightly sophisticated user can arrange to steal mail anyway.

If your DBM files are not writable by the world or you do not have autorebuild enabled (with the **D** option), then you must be careful to reconstruct the alias database each time you change the text version:

```
newaliases
```

If this step is ignored or forgotten, any intended changes will also be ignored or forgotten.

The Configuration File

This section describes the configuration file in detail, including hints on how to write one of your own if you have to.

There is one point that should be made clear immediately: the syntax of the configuration file is designed to be reasonably easy to parse, since this is done every time **sendmail** starts up. As a result, the configuration file is not particularly easy for a human to read or write.

An overview of the configuration file is given first, followed by details of the semantics.

The Syntax

The configuration file is organized as a series of lines, each of which begins with a single character defining the semantics for the rest of the line. Lines beginning with a space or a tab are continuation lines (although the semantics are not well defined in many places). Blank lines and lines beginning with a pound symbol (#) are comments.

R and S - Rewriting Rules

The core of address parsing is the rewriting rules. These are an ordered production system. The **sendmail** command scans through the set of rewriting rules looking for a match on the left hand side (LHS) of the rule. When a rule matches, the address is replaced by the right hand side (RHS) of the rule.

There are several sets of rewriting rules. Some of the rewriting sets are used internally and must have specific semantics. Other rewriting sets do not have specifically assigned semantics, and may be referenced by the mailer definitions or by other rewriting sets.

The syntax of these two commands are:

S*n*

Sets the current ruleset being collected to *n*. If you begin a ruleset more than once, it deletes the old definition.

R*lhs rhs comments*

The fields must be separated by at least one tab character; there may be embedded spaces in the fields. The *lhs* is a pattern that is applied to the input. If it matches, the input is rewritten to the *rhs*. The *comments* are ignored.

D - Define Macro

Macros are named with a single character. These may be selected from the entire ASCII set, but user-defined macros should be selected from the set of uppercase letters only. Lowercase letters and special symbols are used internally.

The syntax for macro definitions is:

D*x val*

where *x* is the name of the macro and *val* is the value it should have. Macros can be interpolated in most places using the escape sequence $*x*.

C and F - Define Classes

Classes of words may be defined to match on the left-hand side of rewriting rules. For example, a class of all local names for this site might be created so that attempts to send to oneself can be eliminated. These can either be defined directly in the configuration file or read in from another file. Classes may be given names from the set of uppercase letters. Lowercase letters and special characters are reserved for system use.

The syntax is:

C*c word1 word2...*
F*c file*

The first form defines the class *c* to match any of the named words. It is permissible to split them among multiple lines; for example, the two forms:

```
CHmonet ucbmonet
```

and:

```
CHmonet
CHucbmonet
```

are equivalent. The second form reads the elements of the class *c* from the named *file*.

M - Define Mailer

Programs and interfaces to mailers are defined in this line. The format is:

M*name*, {*field=value* }*

where *name* is the name of the mailer (used internally only) and the "field=name" pairs define attributes of the mailer. Fields are:

Path	The pathname of the mailer
Flags	Special flags for this mailer
Sender	A rewriting set for sender addresses
Recipient	A rewriting set for recipient addresses
Argv	An argument vector to pass to this mailer
Eol	The end-of-line string for this mailer
Maxsize	The maximum message length for this mailer

Only the first character of the field name is checked.

H - Define Header

The format of the header lines that **sendmail** inserts into the message is defined by the **H** line. The syntax of this line is:

H[?*mflags*?]*hname*: *htemplate*

Continuation lines in this spec are reflected directly into the outgoing message. The *htemplate* is macro-expanded before insertion into the message. If the *mflags* (surrounded by question marks) are specified, at least one of the specified flags must be stated in the mailer definition for this header to be automatically output. If one of these headers is in the input, it is reflected to the output, regardless of these flags.

Some headers have special semantics, described below.

O - Set Option

There are a number of "random" options that can be set from a configuration file. Options are represented by single characters. The syntax of this line is:

O*o value*

This sets option *o* to be *value*. Depending on the option, *value* may be a string, an integer, a boolean (with legal values "t", "T", "f", or "F"; the default is TRUE), or a time interval.

T - Define Trusted Users

Trusted users are those who are permitted to override the sender address using the **-f** flag. These typically are "root," "uucp," and "network," but in some cases it may be convenient to extend this list to include other users, perhaps to support a separate UUCP login for each host. The syntax of this line is:

T*user1 user2...*

There may be more than one of these lines.

P - Precedence Definitions

Values for the "Precedence:'' field may be defined using the **P** control line. The syntax of this field is:

P*name=num*

When the *name* is found in a "Precedence:'' field, the message class is set to *num*. Higher numbers mean higher precedence. Numbers below zero have the special property that error messages will not be returned. The default precedence is zero. For example, our list of precedences is:

```
Pfirst-class=0
Pspecial-delivery=100
Pjunk=-100
```

The Semantics

This section describes the semantics of the configuration file.

Special Macros, Conditionals

Macros are interpolated using the construct $*x*, where *x* is the name of the macro to be interpolated. In particular: lowercase letters are reserved to

have special semantics, used to pass information in or out of **sendmail**; some special characters are reserved to provide conditionals; and so on.

Conditionals can be specified using the syntax:

$?x *text1* **$|** *text2* **$.**

This interpolates *text1* if the macro **$x** is set, and *text2* otherwise. The "else" ($|) clause may be omitted.

The following macros must be defined to transmit information into **sendmail**:

e	The SMTP entry message
j	The official domain name for this site
l	The format of the UNIX from line
n	The name of the daemon (for error messages)
o	The set of "operators" in addresses
q	default format of sender address

The **$e** macro is printed out when SMTP starts up. The first word must be the **$j** macro. The **$j** macro should be in RFC821 format. The **$l** and **$n** macros can be considered constants, except under terribly unusual circumstances. The **$o** macro consists of a list of characters that will be considered tokens and that will separate tokens when parsing. For example, if "@" were in the **$o** macro, then the input "a@b" would be scanned as three tokens: "a", "@", and "b". Finally, the **$q** macro specifies how an address should appear in a message when it is defaulted. For example, on our system, these definitions are:

```
De$j Sendmail $v ready at $b
DnMAILER-DAEMON
DlFrom $g  $d
Do.:%@!^=/
Dq$g$?x ($x)$.
Dj$H.$D
```

An acceptable alternative for the **$q** macro is "$?x$x $.<$g>". These correspond to the following two formats:

```
eric@Lachman (Eric Allman)
Eric Allman <eric@Lachman>
```

Some macros are defined by **sendmail** for interpolation into argv's for mailers or for other contexts. These macros are:

a	The origination date in Arpanet format
b	The current date in Arpanet format
c	The hop count
d	The date in UNIX (ctime) format
f	The sender (from) address
g	The sender address relative to the recipient
h	The recipient host
i	The queue id
p	**sendmail**'s pid
r	Protocol used
s	Sender's host name
t	A numeric representation of the current time
u	The recipient user
v	The version number of **sendmail**
w	The hostname of this site
x	The full name of the sender
z	The home directory of the recipient

There are three types of dates that can be used. The **$a** and **$b** macros are in Arpanet format; **$a** is the time as extracted from the "Date:" line of the message (if there was one), and **$b** is the current date and time (used for postmarks). If no "Date:" line is found in the incoming message, **$a** is set to the current time also. The **$d** macro is equivalent to the **$a** macro in UNIX (ctime) format.

The **$f** macro is the id of the sender as originally determined; when you are mailing to a specific host, the **$g** macro is set to the address of the sender relative to the recipient. For example, if I send to "bollard@matisse" from the machine "ucbarpa," the **$f** macro will be "eric" and the **$g** macro will be "eric@ucbarpa."

The **$x** macro is set to the full name of the sender. This can be determined in several ways. It can be passed as a flag to **sendmail.** The second choice is the value of the "Full-name:" line in the header if it exists, and the third choice is the comment field of a "From:" line. If all of these fail, and if the message is being originated locally, the full name is looked up in the */etc/passwd* file.

When sending, the **$h**, **$u**, and **$z** macros are set to the host, user, and home directories (if local) of the recipient. The first two are set from the **$@** and **$:** parts of the rewriting rules, respectively.

The **$p** and **$t** macros are used to create unique strings (for example, for the "Message-Id:" field). The **$i** macro is set to the queue id on this host;

if put into the timestamp line, it can be extremely useful for tracking messages. The **$v** macro is set to be the version number of **sendmail**; this is normally put in timestamps and has proved extremely useful for debugging. The **$w** macro is set to the name of this host, if it can be determined. The **$c** field is set to the "hop count," that is, the number of times this message has been processed. This can be determined by the **-h** flag on the command line or by counting the timestamps in the message.

The **$r** and **$s** fields are set to the protocol used to communicate with **sendmail** and the sending *hostname*; these are not supported in the current version.

Special classes

The class **$=w** is set to be the set of all names by which this host is known. This can be used to delete local hostnames.

The Left-Hand Side

The left-hand side of rewriting rules contains a pattern. Normal words are simply matched directly. Metasyntax is introduced using a dollar sign. The metasymbols are:

$*	Match zero or more tokens
$+	Match one or more tokens
$-	Match exactly one token
$=*x*	Match any token in class *x*
$~*x*	Match any token not in class *x*

If any of these match, they are assigned to the symbol **$***n* for replacement on the right-hand side, where *n* is the index in the LHS. For example, if the LHS:

```
$-:$+
```

is applied to the input:

```
UCBARPA:eric
```

then the rule will match, and the values passed to the RHS will be:

$1	UCBARPA
$2	eric

The Right-Hand Side

When the left-hand side of a rewriting rule matches, the input is deleted and replaced by the right-hand side. Tokens are copied directly from the RHS, unless they begin with a dollar sign. Metasymbols are:

$*n*	Substitute indefinite token *n* from LHS
$[*name*$]	Canonicalize *name*
$>*n*	Call ruleset *n*
$#*mailer*	Resolve to *mailer*
$@*host*	Specify *host*
$:*user*	Specify *user*

The $*n* syntax substitutes the corresponding value from a $+, $-, $*, $=, or $~ match on the LHS. It can be used anywhere.

A host name enclosed between $[and $] is looked up using the **gethostent**(3) routines and replaced by the canonical name. For example, "$[csam$]" would become "lbl-csam.arpa", and "$[[128.32.130.2]$]" would become "vangogh.berkeley.edu."

The $>*n* syntax causes the remainder of the line to be substituted as usual and then passed as the argument to ruleset *n*. The final value of ruleset *n* then becomes the substitution for this rule.

The $# syntax should only be used in ruleset zero. It causes evaluation of the ruleset to terminate immediately, and it signals to **sendmail** that the address has completely resolved. The complete syntax is:

$#*mailer*$@*host*$:*user*

This specifies the {mailer, host, user} 3-tuple necessary to direct the mailer. If the mailer is local, the host part can be omitted. The *mailer* and *host* must be a single word, but the *user* can be multi-part.

A RHS can also be preceded by a $@ or a $: to control evaluation. A $@ prefix causes the ruleset to return with the remainder of the RHS as the value. A $: prefix causes the rule to terminate immediately, but the ruleset to continue. This can be used to avoid continued application of a rule. The prefix is stripped before continuing.

The $@ and $: prefixes can precede a $> spec. For example,

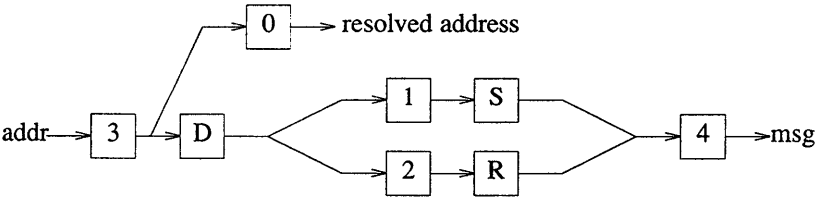
R$+ $:$>7$1

matches anything, passes that to ruleset seven, and continues; the $: is necessary to avoid an infinite loop.

Substitution occurs in the order described; that is, parameters from the LHS are substituted, hostnames are canonicalized, "subroutines" are called and, finally, **$#**, **$@**, and **$:** are processed.

Semantics of Rewriting Rule Sets

There are five rewriting sets that have specific semantics. These are related as depicted by Figure 4-1.



D - sender domain addition
S - mailer-specific sender rewriting
R - mailer-specific recipient rewriting

Figure 3-1 Rewriting Set Semantics

Ruleset three should turn the address into "canonical form." This form should have the basic syntax:

local-part@host-domain-spec

If no "@" sign is specified, then the host-domain-spec can be appended from the sender address (if the **C** flag is set in the mailer definition corresponding to the sending mailer). Ruleset three is applied by **sendmail** before doing anything with any address.

Ruleset zero is applied after ruleset three to addresses that are actually going to specify recipients. It must resolve to a {mailer, host, user} triple. The mailer must be defined in the mailer definitions from the configuration file. The host is defined into the **$h** macro for use in the argv expansion of the specified mailer.

Rulesets one and two are applied to all sender and recipient addresses, respectively. They are applied before any specification in the mailer definition. They must never resolve.

Ruleset four is applied to all addresses in the message. It is typically used to translate internal to external form.

Mailer Flags

There are several flags that can be associated with each mailer, each identified by a letter of the alphabet. Many of them are assigned semantics internally. Any other flags may be used freely to assign headers conditionally to messages destined for particular mailers.

The "error" Mailer

The mailer with the special name "error" can be used to generate a user error. The (optional) host field is a numeric exit status to be returned, and the user field is a message to be printed. For example, the entry:

```
$#error$:Host unknown in this domain
```

on the RHS of a rule will cause the specified error to be generated if the LHS matches. This mailer is only functional in ruleset zero.

Building a Configuration File from Scratch

Building a configuration file from scratch is an extremely difficult job. Fortunately, it is almost never necessary to do so; nearly every situation that may come up may be resolved by changing an existing table. In any case, it is critical that you understand what you are trying to do and come up with a philosophy for the configuration table. This section is intended to explain the real purpose of a configuration table and to give you some ideas as to what your philosophy might be.

What You are Trying to Do

The configuration table has three major purposes. The first and simplest is to set up the environment for **sendmail**. This involves setting the options, defining a few critical macros, and so on. Since these are described in other sections, we will not go into more detail here.

The second purpose is to rewrite addresses in the message. This should typically be done in two phases. The first phase maps addresses in any format into a canonical form. This should be done in ruleset three. The second phase maps this canonical form into the syntax appropriate for the receiving mailer.

The **sendmail** program performs this second phase in the following three subphases: Rulesets one and two are applied to all sender and recipient addresses, respectively. After this, you can specify per-mailer rulesets for

both sender and recipient addresses. This allows mailer-specific customization. Finally, ruleset four is applied to do any default conversion to external form.

The third purpose of the configuration table is to map addresses into the actual set of instructions necessary to get the message delivered. Ruleset zero must resolve to the internal form, which is in turn used as a pointer to a mailer descriptor. This describes the interface requirements of the mailer.

Relevant Issues

The canonical form you use should almost certainly be as specified in the Arpanet standards documents RFC819 and RFC822.
RFC822 describes the format of the mail message itself. The **sendmail** program follows this RFC closely, to the extent that many of the standards described in this document can not be changed without changing the code. In particular, the following characters have special interpretations:

```
<>()"\
```

Any attempt to use these characters for other than their RFC822 purpose in addresses is probably doomed to disaster.

RFC819 describes the specifics of the domain-based addressing. This is touched on in RFC822 as well. Essentially, each host is given a name that is a right-to-left dot-qualified pseudo-path from a distinguished root. The elements of the path need not be physical hosts; the domain is logical rather than physical. For example, at Lachman, one legal host might be "a.CC.Lachman.EDU"; reading from right to left, "EDU" is a top level domain comprising educational institutions, "Lachman" is a logical domain name, "CC" represents the Computer Center, (in this case a strictly logical entity), and "a" is a host in the Computer Center.

When reading RFC819, be aware that there are a number of errors in it.

How to Proceed

Once you have decided on a philosophy, it is worth examining the available configuration tables to decide whether any of them are close enough for you to steal their major parts. Even under the worst of conditions, there is a fair amount of boilerplate that can be collected safely.

The next step is to build ruleset three. This will be the hardest part of the job. Beware of doing too much to the address in this ruleset, because

anything you do will reflect through to the message. In particular, stripping of local domains is best deferred, as this can leave you with addresses with no domain specs at all. Because **sendmail** likes to append the sending domain to addresses with no domain, this can change the semantics of addresses. Also, try to avoid fully qualifying domains in this ruleset. Although technically legal, this can lead to unpleasantly and unnecessarily long addresses reflected into messages. The Lachman configuration files define ruleset nine to qualify domain names and strip local domains. This is called from ruleset zero to get all addresses into a cleaner form.

Once you have ruleset three finished, the other rulesets should be relatively trivial. If you need hints, examine the supplied configuration tables.

Testing the Rewriting Rules: the -bt Flag

When you build a configuration table, you can do a certain amount of testing using the "test mode" of **sendmail**. For example, you could invoke **sendmail** as:

```
sendmail -bt -Ctest.cf
```

which would read the configuration file "test.cf" and enter test mode. In this mode, you enter lines of the form:

```
rwset address
```

where *rwset* is the rewriting set you want to use and *address* is an address to which to apply the set. Test mode shows you the steps it takes as it proceeds, finally showing you the address with which it ends up. You may use a comma-separated list of rwsets for sequential application of rules to an input; ruleset three is always applied first. For example:

```
1,21,4 monet:bollard
```

first applies ruleset three to the input "monet:bollard." Ruleset one is then applied to the output of ruleset three, followed similarly by rulesets twenty-one and four.

If you need more detail, you can also use the "-d21" flag to turn on more debugging. For example:

```
sendmail -bt -d21.99
```

turns on an incredible amount of information. A single-word address will probably print out several pages of information.

Building Mailer Descriptions

To add an outgoing mailer to your mail system, you must define the characteristics of the mailer.

Each mailer must have an internal name. This can be arbitrary, except that the names "local" and "prog" must be defined.

The pathname of the mailer must be given in the P field. If this mailer should be accessed via an IPC connection, use the string "[IPC]" instead.

The F field defines the mailer flags. You should specify an "f" or "r" flag to pass the name of the sender as a **-f** or **-r** flag, respectively. These flags are only passed if they were passed to **sendmail**, so that mailers that give errors under some circumstances can be placated. If the mailer is not picky, you can just specify "-f $g" in the argv template. If the mailer must be called as **root**, the S flag should be given. This will not reset the userid before calling the mailer. If this mailer is local (that is, it will perform final delivery rather than another network hop), the l flag should be given. Quote characters (backslashes and " marks) can be stripped from addresses if the s flag is specified. If this is not given, they are passed through. If the mailer is capable of sending to more than one user on the same host in a single transaction, the m flag should be stated. If this flag is on, then the argv template containing **$u** will be repeated for each unique user on a given host. The e flag will mark the mailer as being expensive, which will cause **sendmail** to defer connection until a queue run.

An unusual case is the C flag. This flag applies to the mailer that the message is received from, rather than the mailer being sent to; if set, the domain spec of the sender (that is, the @host.domain part) is saved and is appended to any addresses in the message that do not already contain a domain spec. For example, a message of the form:

```
From: eric@ucbarpa
To: wnj@monet, mckee
```

will be modified to:

```
From: eric@ucbarpa
To: wnj@monet, mckee@ucbarpa
```

if and only if the C flag is defined in the mailer corresponding to eric@ucbarpa.

The S and R fields in the mailer description are per-mailer rewriting sets to be applied to sender and recipient addresses, respectively. These are applied after the sending domain is appended and the general rewriting

sets (numbers one and two) are applied, but before the output rewrite (ruleset four) is applied. A typical use is to append the current domain to addresses that do not already have a domain. For example, a header of the form:

```
From: eric
```

might be changed to be:

```
From: eric@ucbarpa
```

or:

```
From: ucbvax!eric
```

depending on the domain it is being shipped into. These sets can also be used to do special-purpose output rewriting in cooperation with ruleset four.

The E field defines the string to use as an end-of-line indication. A string containing only newline is the default. The usual backslash escapes (\r, \n, \f, \b) may be used.

Finally, an argv template is given as the E field. It may have embedded spaces. If there is no argv with a **$u** macro in it, **sendmail** will speak SMTP to the mailer. If the pathname for this mailer is [IPC], the argv should be:

IPC $h [*port*]

where *port* is the optional port number to connect to.

For example, the specifications:

```
Mlocal, P=/usr/bin/mail, F=lsDFMmnPS S=10, R=20, A=lmail $u
Mether, P=[IPC], F=meC, S=11, R=21, A=IPC $h, M=100000
```

specify a mailer to do local delivery and a mailer for Ethernet delivery. The first is called local; it is located in the file /bin/mail, takes a picky **-r** flag, and does local delivery; quotes should be stripped from addresses, and multiple users can be delivered at once; ruleset ten should be applied to sender addresses in the message, and ruleset twenty should be applied to recipient addresses. The argv to send to a message will be the word mail, the word -d, and words containing the name of the receiving user. If a **-r** flag is inserted, it will be between the words mail and -d. The second mailer is called ether; it should be connected to via an IPC connection; it can handle multiple users at once; connections should be deferred; and any domain from the sender address should be appended to any receiver

name without a domain. Sender addresses should be processed by ruleset eleven and recipient addresses by ruleset twenty-one. There is a 100,000-byte limit on messages passed through this mailer.

Command Line Flags

Arguments must be presented with flags before addresses. The flags are:

-f *addr* The sender's machine address is *addr*. This flag is ignored unless the real user is listed as a "trusted user" or *addr* contains an exclamation point (because of certain restrictions in UUCP).

-r *addr* An obsolete form of **-f**.

-h *cnt* Sets the "hop count" to *cnt*. This represents the number of times this message has been processed by **sendmail** (to the extent that it is supported by the underlying networks). *cnt* is incremented during processing, and if it reaches MAXHOP (currently 30) **sendmail** throws away the message with an error.

-F*name* Sets the full name of this user to *name*.

-n Do not alias or forward.

-t Read the header for To:, Cc:, and Bcc: lines, and send to everyone in those lists. The Bcc: line will be deleted before sending. Any addresses in the argument vector will be deleted from the send list.

-b*x* Set operation mode to *x*. The operation modes are:

m	Deliver mail (default)
a	Run in ARPANET mode (see below)
s	Speak SMTP on input side
d	Run as a daemon
t	Run in test mode
v	Just verify addresses, do not collect or deliver
i	Initialize the alias database
p	Print the mail queue
z	Freeze the configuration file

The special processing for the ARPANET includes reading the From: line from the header to find the sender, printing ARPANET-style messages (preceded by three-digit reply codes for compatibility with the FTP protocol), and ending lines of error messages with <CRLF>.

-q*time* Try to process the queued up mail. If the time is given, **sendmail** will run through the queue at the specified interval to deliver queued mail; otherwise, it only runs once.

-C*file* Use a different configuration file. The **sendmail** program runs as the invoking user (rather than root) when this flag is specified.

-d*level* Set debugging level.

-o*x value* Set option *x* to the specified *value*. These options are described in the next section.

There are a number of options that can be specified as primitive flags (provided for compatibility with **delivermail**). These are the e, i, m, and v options. Also, the f option can be specified as the **-s** flag.

Configuration Options

The following options can be set using the **-o** flag on the command line or the **O** line in the configuration file. Many of them cannot be specified unless the invoking user is trusted.

A*file* Use the named *file* as the alias file. If no file is specified, use *aliases* in the current directory.

a*N* If set, wait up to *N* minutes for an @:@ entry to exist in the alias database before starting up. If it does not appear in *N* minutes, rebuild the database (if the **D** option is also set) or issue a warning.

B*c* Set the blank substitution character to *c*. Unquoted spaces in addresses are replaced by this character.

c If an outgoing mailer is marked as being expensive, do not connect immediately. This requires that queueing be compiled in, since it will depend on a queue run process to actually send the mail.

d*x* Deliver in mode *x*. Legal modes are:

- i Deliver interactively (synchronously)
- b Deliver in background (asynchronously)
- q Just queue the message (deliver during queue run)

D If set, rebuild the alias database if necessary and possible. If this option is not set, **sendmail** will never rebuild the alias database unless explicitly requested using **-bi**.

e*x* Dispose of errors using mode *x*. The values for *x* are:

- p Print error messages (default)
- q No messages, just give exit status
- m Mail back errors
- w Write back errors (mail if user not logged in)
- e Mail back errors and give zero exit stat always

F*n* The temporary file mode, in octal. 644 and 600 are good choices.

f — Save UNIX-style "From" lines at the front of headers. Normally, they are assumed redundant and discarded.

g*n* — Set the default group id for mailers to run into *n*.

H*file* — Specify the help file for SMTP.

i — Ignore dots in incoming messages.

L*n* — Set the default log level to *n*.

M*x value* — Set the macro *x* to *value*. This is intended only for use from the command line.

m — Send to me, too, even if I am in an alias expansion.

N*netname* — The name of the home network (ARPA is the default). The argument of an SMTP HELO command is checked against hostname.netname, where *hostname* is requested from the kernel for the current connection. If they do not match, Received: lines are augmented by the name that is determined in this manner, so that messages can be traced accurately.

o — Assume that the headers may be in old format; that is, spaces delimit names. This actually turns on an adaptive algorithm: if any recipient address contains a comma, parenthesis, or angle bracket, it will be assumed that commas already exist. If this flag is not on, only commas delimit names. Headers are always output with commas between the names.

Q*dir* — Use the named *dir* as the queue directory.

q*factor* — Use *factor* as the multiplier in the map function to decide when just to queue up jobs rather than run them. This value is divided by the difference between the current load average and the load average limit (**x** flag) to determine the maximum message priority that will be sent. Defaults to 10,000.

r*time* — Timeout reads after *time* interval.

S*file* — Log statistics in the named *file*.

s — Be extra safe when running things, that is, always instantiate the queue file, even if you are going to attempt immediate delivery. The **sendmail** program

always instantiates the queue file before returning control the client.

T*time* — Set the queue timeout to *time*. After this interval, messages that have not been successfully sent will be returned to the sender.

t*S,D* — Set the local time zone name to *S* for standard time and *D* for daylight time. This is only used under version six.

u*n* — Set the default userid for mailers to *n*. Any mailer without the *S* flag in the mailer definition will run as this user.

v — Run in verbose mode.

x*LA* — When the system-load average exceeds *LA*, just queue messages (that is, do not try to send them).

X*LA* — When the system load average exceeds *LA*, refuse incoming SMTP connections.

y*fact* — The indicated *fact*or is added to the priority (thus *lowering* the priority of the job) for each recipient, that is, this value penalizes jobs with large numbers of recipients.

Y — If set, deliver each job that is run from the queue in a separate process. Use this option if you are short of memory, since the default tends to consume considerable amounts of memory while the queue is being processed.

z*fact* — The indicated *fact*or is multiplied by the message class (determined by the Precedence: field in the user header and the **P** lines in the configuration file) and subtracted from the priority. Thus, messages with a higher Priority will be favored.

Z*fact* — The *fact*or is added to the priority every time a job is processed. Thus, each time a job is processed, its priority will be decreased by the indicated value. In most environments, this should be positive, since hosts that are down are all too often down for a long time.

Mailer Flags

The following flags can be set in the mailer description.

f The mailer wants a **-f***from* flag, but only if this is a network forward operation (that is, the mailer will give an error if the executing user does not have special permissions).

r Same as **f**, but sends a **-r** flag.

S Do not reset the userid before calling the mailer. This would be used in a secure environment where **sendmail** ran as root. This could be used to avoid forged addresses. This flag is suppressed if given from an unsafe environment (for example, a user's mail.cf file).

n Do not insert a UNIX-style From: line on the front of the message.

l This mailer is local (that is, final delivery will be performed).

s Strip quote characters off the address before calling the mailer.

m This mailer can send to multiple users on the same host in one transaction. When a **$u** macro occurs in the *argv* part of the mailer definition, that field will be repeated as necessary for all qualifying users.

F This mailer wants a From: header line.

D This mailer wants a Date: header line.

M This mailer wants a Message-Id: header line.

x This mailer wants a Full-Name: header line.

P This mailer wants a Return-Path: line.

u Uppercase should be preserved in user names for this mailer.

h Uppercase should be preserved in host names for this mailer.

A This is an Arpanet-compatible mailer, and all appropriate modes should be set.

U This mailer wants UNIX-style From: lines with the ugly UUCP-style "remote from <host>" on the end.

e This mailer is expensive to connect to, so try to avoid connecting normally. Any necessary connection will occur during a queue run.

X This mailer want to use the hidden dot algorithm as specified in RFC821. Basically, any line beginning with a dot will have an extra dot prepended (to be stripped at the other end). This ensures that lines in the message containing a dot will not terminate the message prematurely.

L Limit the line lengths as specified in RFC821.

P Use the return-path in the SMTP "MAIL FROM:" command, rather than just the return address. Although this is required in RFC821, many hosts do not process return paths properly.

I This mailer will be speaking SMTP to another **sendmail**. As such, it can use special protocol features. This option is not required (that is, if this option is omitted, the transmission will still operate successfully, although perhaps not as efficiently as possible).

C If mail is received from a mailer with this flag set, any addresses in the header that do not have an at sign (@) after being rewritten by ruleset three will have the @domain clause from the sender tacked on. This allows mail with headers of the form:

```
From: usera@hosta
To: userb@hostb, userc
```

to be rewritten automatically as:

```
From: usera@hosta
To: userb@hostb, userc@hosta
```

E Escape lines beginning with From in the message with a '>' sign.

Summary of Support Files

This is a summary of the support files that **sendmail** creates or generates.

/usr/lib/sendmail	The binary of **sendmail**.
/usr/bin/newaliases	A link to */usr/lib/sendmail*; causes the alias database to be rebuilt. Running this program is completely equivalent to giving **sendmail** the **-bi** flag.
/usr/bin/mailq	Prints a listing of the mail queue. This program is equivalent to using the **-bp** flag to **sendmail**.
/usr/lib/sendmail.cf	The configuration file, in textual form.
/usr/lib/sendmail.fc	The configuration file represented as a memory image.
/usr/lib/sendmail.hf	The SMTP help file.
/usr/lib/sendmail.st	A statistics file; need not be present.
/usr/lib/aliases	The textual version of the alias file.
/usr/lib/aliases.{pag,dir}	The alias file in **dbm**(S) format.
/usr/spool/mqueue	The directory in which the mail queue and temporary files reside.
*/usr/spool/mqueue/qf**	Control (queue) files for messages.
*/usr/spool/mqueue/df**	Data files.
*/usr/spool/mqueue/lf**	Lock files
*/usr/spool/mqueue/tf**	Temporary versions of the qf files, used during queue-file rebuild.
*/usr/spool/mqueue/nf**	A file used when creating a unique id.
*/usr/spool/mqueue/xf**	A transcript of the current session.

Chapter 4

Name Server Operations Guide for BIND

Introduction

A name server is a network service that enables clients to name resources or objects and share this information with other objects in the network. The Berkeley Internet Name Domain (BIND) Server implements the DARPA Internet name server for the UNIX operating system. In effect, this is a distributed database system for objects in a computer network. BIND is fully integrated into network programs for use in storing and retrieving host names and addresses. The system administrator can configure the system to use BIND as a replacement for the original host table lookup of information in the network hosts file */etc/hosts*. The default configuration does not use BIND. BIND is initially disabled. If you want to use it, you must first set up the necessary configuration files.

The Name Service

The basic function of the name server is to provide information about network objects by answering queries. The advantage of using a name server over the host table lookup for host-name resolution is to avoid the need for a single centralized clearinghouse for all names. The authority for this information can be delegated to the different organizations on the network responsible for it.

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

With the name server, the network can be broken into a hierarchy of domains. The name space is organized as a tree, according to organizational or administrative boundaries. Each node, called a domain, is given a label, and the name of the domain is the concatenation of all the labels of the domains from the root to the current domain, listed from right to left, separated by dots. A label need only be unique within its domain. The whole space is partitioned into several areas called zones, each starting at a domain and extending down to the leaf domains or to domains where other zones start. Zones usually represent administrative boundaries. An example of a host address for a host at the University of California, Berkeley, would look as follows:

```
monet.Berkeley.EDU
```

The top-level domain for educational organizations is EDU; Berkeley is a subdomain of EDU and monet is the name of the host. Additional top-level domains include:

COM	Commercial Organizations
GOV	Government Organizations
MIL	Military Departments
ORG	Miscellaneous Organizations

Types of Servers

There are several types of servers. These are:

- master servers
- caching-only servers
- remote servers
- slave servers

These types of servers are described in more detail in the following four sections.

Master Servers

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

Primary

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

Secondary

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

Caching-Only Servers

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

Remote Servers

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

Slave Server

A slave server is a server that always forwards queries it cannot satisfy locally to a fixed list of forwarding servers instead of interacting with the master name servers for the root and other domains. The queries to the forwarding servers are recursive queries. There may be one or more forwarding servers, and they are tried in turn until the list is exhausted. A slave and forwarder configuration is typically used when you do not wish all the servers at a given site to be interacting with the rest of the Internet servers. A typical scenario would involve a number of workstations and a departmental timesharing machine with Internet access. The workstations might be administratively prohibited from having Internet access. To give the workstations the appearance of access to the Internet domain system, the workstations could be slave servers to the timesharing machine, which would forward the queries and interact with other name servers to resolve the query before returning the answer. An added benefit of using the forwarding feature is that the central machine develops a much more complete cache of information that all the workstations can take advantage of. The use of slave mode and forwarding is discussed further under the description of the named bootfile commands.

Setting Up Your Own Domain

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

The contacts are as follows:

DARPA Internet

Sites that are already on the DARPA Internet and need information on setting up a domain should contact HOSTMASTER@SRI-NIC.ARPA. You may also want to be placed on the BIND mailing list, which is a mail group for people on the DARPA Internet running BIND. This group discusses future design decisions, operational problems, and other related topics. To request placement on this mailing list, send mail to the following address:

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

CSNET

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

An organization that already has a registered domain name should keep the CIC informed about how it would like its mail routed. In general, the CSNET relay prefers to send mail via CSNET if possible (as opposed to BITNET or the Internet). For an organization on multiple networks, this may not always be the preferred behavior. The CIC can be reached via electronic mail at cic @ sh.cs.net, or by phone at (617) 497-2777.

BITNET

If you are on the BITNET and need to set up a domain, contact INFO@BITNIC.

Boot File

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

Domain

The boot file contains a line of code that designates the default domain. The line for the server looks like this:

```
domain      Berkeley.Edu
```

The name server uses this information when it receives a query for a name without a "." that is unknown. When it receives one of these queries, it appends the name in the second field to the query name. This is an obsolete facility, which will be removed from future releases.

Directory

The directory line specifies the directory in which the name server should run, allowing the other filenames in the boot file to use relative pathnames.

```
directory           /usr/local/lib/named
```

If you have more than a couple of named files to be maintained, you may wish to place the named files in a directory such as */usr/local/domain* and adjust the directory command properly. The main purposes of this command are to make sure **named** is in the proper directory when trying to include files by relative pathnames with $INCLUDE and to allow **named** to run in a location that is reasonable to dump core if it feels the urge.

Primary Master

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

```
primary             Berkeley.Edu    ucbhosts
```

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

Secondary Master

The line for a secondary server is similar to that for the primary, except that it lists addresses of other servers. (usually primary servers) from which the zone data is obtained.

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

The first field specifies that the server is a secondary master server for the zone stated in the second field. The two network addresses specify the name servers that are primary for the zone. The secondary server gets its data across the network from the listed servers. Each server is tried in the order listed until it successfully receives the data from a listed server. If a filename is present after the list of primary servers, data for the zone is dumped into that file as a backup. When the server is first started, the data are loaded from the backup file if possible, and a primary server is then consulted to check that the zone is still up-to-date.

Caching-Only Server

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

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

```
cache              .                    root.cache
```

The period (.) specifies the current domain. All cache files listed are read in at named boot time and any values still valid are reinstated in the cache and the root name server information in the cache files are always used. For information on the cache file, see the later section, "Initializing the Cache."

Forwarders

Any server can make use of forwarders. A forwarder is another server capable of processing recursive queries to try to resolve queries on behalf of other systems. The **forwarders** command specifies forwarders by internet address as follows:

```
forwarders          128.32.0.10 128.32.0.4
```

There are two main reasons for wanting to do so. First, the other systems may not have full network access and may be prevented from sending any IP packets into the rest of the network and, therefore, must rely on a forwarder that does have access to the full net. The second reason is that the forwarder sees a union of all queries as they pass through the forwarder's server and, therefore, the forwarder builds up a very rich cache of data compared to the cache in a typical workstation name server. In effect, the forwarder becomes a meta-cache that all hosts can benefit from, thereby reducing the total number of queries from that site to the rest of the net.

Slave Mode

Slave mode is used if the use of forwarders is the only possible way to resolve queries because of lack of full net access or if you wish to prevent the name server from using other than the listed forwarders. Slave mode is activated by placing the simple command

slave

in the bootfile. If **slave** is used, then you must specify forwarders. When in slave mode, the server forwards each query to each of the forwarders until an answer is found or the list of forwarders is exhausted.

Remote Servers

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

Initializing the Cache

The name server needs to know the identities of the authoritative name servers for the root domain of the network. To do this, you have to prime the name server's cache with the address of these higher authorities. This is done in a file called *root.cache*. The location of this file is specified in the boot file */etc/named.boot*.

There are three standard files used to specify the data for a domain. These files are:

named.local
hosts
host.rev.

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

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

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

Standard Resource Records

The records in the name server data files are called resource records. The following is a general description of a resource record:

```
{name}    {ttl}    addr-class    Record Type    Record Specific data
```

Resource records have a standard format, as shown above. The first field is always the name of the domain record and it must always start in column 1. For some resource records, the name can be left blank. In such cases, the name of the previous resource record is used. The second field is an optional time-to-live field. This specifies how long this data is stored in the database. When this field is left blank, the default time-to-live is specified in the Start of Authority resource record discussed later in this chapter. The third field is the address class. There are currently two classes: IN for internet addresses and ANY for all address classes. The fourth field states the type of the resource record. The fields after that are dependent on the type of the resource record. Case is preserved in names and data fields when loaded into the name server. All comparisons and lookups in the name server database are case-insensitive.

The following characters have special meanings:

.	A free-standing dot in the name field refers to the current domain.
@	A free-standing @ in the name field denotes the current origin.
..	Two free-standing dots represent the null domain name of the root when used in the name field.
\X	Where X is any character other than a digit (0-9), \X quotes that character so that its special meaning does not apply. For example, "\." can be used to place a dot character in a label.
\DDD	Where each D is a digit, \DDD is the octet corresponding to the decimal number described by DDD. The resulting octet is assumed to be text and is not checked for special meaning.
( )	Parentheses are used to group data that crosses a line. In effect, line terminations are not recognized within parentheses.

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

* An asterisk signifies a wildcard.

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

Separating Data into Multiple Files

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

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

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

Changing an Origin in a Data File

Use the $ORIGIN command to change the origin in a data file. The line starts in column 1 and is followed by a domain origin. This is useful for putting more than one domain in a data file. For example, */etc/named.hosts* might contain lines of the form:

```
$ORIGIN CC.Berkeley.EDU
[assorted domain data...]
$ORIGIN EE.Berkeley.EDU
[assorted domain data...]
```

The Start of Authority Resource Record (SOA)

The Start of Authority record designates the start of a zone. An SOA record includes the following fields:

- Name
- Origin
- Person in charge
- Serial number
- Refresh
- Retry
- Expire
- Minimum

"Name" is the name of the zone. "Origin" is the name of the host on which this data file resides. "Person in charge" is the mailing address for the person responsible for the name server. "Serial number" is the version number of this data file; this number should be incremented whenever a change is made to the data. (Note that the name server cannot handle numbers over 9999 after the decimal point.) "Refresh" indicates how often, in seconds, a secondary name server is to check with the primary name server to see if an update is needed. "Retry" indicates how long, in seconds, a secondary server is to retry after a failure to check for a refresh. "Expire" is the upper time limit, in seconds, that a secondary name server is to use the data before it expires for lack of getting a refresh. Minimum is the default number of seconds to be used for the time-to-live field on resource records. There should only be one SOA record per zone. Here is an example of an SOA record:

```
name  {ttl}  addr-class  SOA     Origin              Person in charge
@            IN          SOA     ucbvax.Berkeley.Edu.  kjd.ucbvax.Berkeley.Edu. (
                         1.1     ; Serial
                         3600    ; Refresh
                         300     ; Retry
                         3600000 ; Expire
                         3600 )  ; Minimum
```

The Name Server Resource Record (NS)

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

```
{name}    {ttl}    addr-class    NS    Name servers name
                   IN            NS    ucbarpa.Berkeley.Edu.
```

The address class is IN (Internet addresses), and the record type is name server (NS). The record uses the default ttl (time-to-live) value. Here, the record-specific data is the identity of the name server.

The Address Resource Record (A)

The address record (A) lists the address for a given machine. The name field is the machine name and the address is the network address. There should be one A record for each address of the machine. Here is an example of an address record for a machine named *ucbarpa* with two network addresses:

```
{name}    {ttl}    addr-class    A    address
ucbarpa            IN            A    128.32.0.4
                   IN            A    10.0.0.78
```

The Host Information Resource Record (HINFO)

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

```
{name}    {ttl}    addr-class    HINFO    Hardware      OS
                   ANY           HINFO    VAX-11/780    UNIX
```

Note that the current release ignores any records that appear after an HINFO record. Thus, you can use only one HINFO record within the file, and it should be the last record in the file.

The Well-Known Services Resource Record (WKS)

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

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

The Canonical Name Resource Record (CNAME)

The canonical name resource record (CNAME) specifies an alias for a canonical name. An alias should be the only record associated with the alias name; all other resource records should be associated with the canonical name and not with the alias. Any resource records that include a domain name as their value (for example, NS or MX) should list the canonical name, not the alias. Here is an example of a CNAME record:

```
aliases     {ttl}   addr-class   CNAME   Canonical name
ucbmonet            IN           CNAME   monet
```

The Domain Name Pointer Resource Record (PTR)

A domain name pointer record (PTR) allows special names to point to some other location in the domain. The following example of a PTR record is used in setting up reverse pointers for the special IN-ADDR.ARPA domain. This line is from the example:

```
hosts.rev file.
```

In this record, the name field is the network number of the host in reverse order. You only need to specify enough octets to make the name unique. For example, if all hosts are on network 127.174.14, then only the last octet needs to be specified. If hosts are on networks 128.174.14 and 127.174.23, then the last two octets need to be specified. PTR names should be unique to the zone. Here is an example of a PTR record:

```
name    {ttl}   addr-class    PTR   real name
7.0             IN            PTR   monet.Berkeley.Edu.
```

The Mailbox Resource Record (MB)

The mailbox resource record has a record type of MB. It lists the machine where a user wants to receive mail. The name field is the user's login; the machine field denotes the machine to which mail is to be delivered. Mail box names should be unique to the zone. Here is an example of an MB record:

```
name      {ttl}   addr-class    MB   Machine
miriam            IN            MB   vineyd.DEC.COM.
```

The Mail Rename Resource Record (MR)

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

```
name            {ttl}   addr-class    MR   corresponding MB
Postmistress            IN            MR   miriam
```

The Mailbox Information Resource Record (MINFO)

The mail information record MINFO creates a mail group for a mailing list. This resource record is usually associated with a mail group, but it can be used with a mail box record. The "name" specifies the name of the mailbox. The "requests" field is where mail such as requests to be

added to a mail group should be sent. The "maintainer" is a mailbox that should receive error messages. This is particularly appropriate for mailing lists when errors in members' names should be reported to a person other than the sender. Here is an example of this record:

```
name    {ttl}   addr-class   MINFO   requests       maintainer
BIND            IN           MINFO   BIND-REQUEST   kjd.Berkeley.Edu.
```

The Mail Group Member Resource Record (MG)

The mail group record (MG) lists members of a mail group.

```
{mail group name}    {ttl}   addr-class   MG   member name
                             IN           MG   Bloom
```

An example for setting up a mailing list is as follows:

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

The Mail Exchanger Resource Record (MX)

```
name           {ttl}  addr-class  MX  preference value  mailer exchanger
Munnari.OZ.AU. IN                 MX  0                 Seismo.CSS.GOV.
*.IL.          IN                 MX  0                 RELAY.CS.NET.
```

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

Wildcard names containing the character "*" may be used for mail routing with MX records. There are likely to be servers on the network that simply state that any mail to a domain is to be routed through a relay. In

the second example above, all mail to hosts in the domain IL is routed through RELAY.CS.NET. This is done by creating a wildcard resource record, which states that *.IL has an MX of RELAY.CS.NET.

Some Sample Files

The following sections contain sample files for the name server. This covers example boot files for the different types of server and example domain database files.

Caching-Only Server

```
;
; Boot file for Caching Only Name Server
;

; type    domain                source file or host
;
domain    Berkeley.Edu
cache     .                     /etc/named.ca
primary   0.0.127.in-addr.arpa  /etc/named.local
```

Primary Master Server

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

; type       domain                  source file or host
;
directory    /usr/local/lib/named
primary      Berkeley.Edu            ucbhosts
primary      32.128.in-addr.arpa     ucbhosts.rev
primary      0.0.127.in-addr.arpa    named.local
cache        .                       root.cache
```

Secondary Master Server

```
;
; Boot file for Secondary Name Server
;
; type      domain                  source file or host
;
directory   /usr/local/lib/named
secondary   Berkeley.Edu            128.32.0.4 128.32.0.10 128.32.136.22 ucbhost.bak
secondary   32.128.in-addr.arpa     128.32.0.4 128.32.0.10 128.32.136.22 ucbhosts.rev.bak
primary     0.0.127.in-addr.arpa    named.local
cache       .                       root.cache
```

The /etc/resolv.conf File

```
domain Berkeley.Edu
name server 128.32.0.4
name server 128.32.0.10
```

root.cache

```
;
; Initial cache data for root domain servers.
;
.                   99999999    IN    NS    SRI-NIC.ARPA.
                    99999999    IN    NS    NS.NASA.GOV.
                    99999999    IN    NS    TERP.UMD.EDU.
                    99999999    IN    NS    A.ISI.EDU.
                    99999999    IN    NS    BRL-AOS.ARPA.
                    99999999    IN    NS    GUNTER-ADAM.ARPA.
                    99999999    IN    NS    C.NYSER.NET.
;  Prep the cache (hotwire the addresses).
SRI-NIC.ARPA.       99999999    IN    A     10.0.0.51
SRI-NIC.ARPA.       99999999    IN    A     26.0.0.73
NS.NASA.GOV.        99999999    IN    A     128.102.16.10
BRL-AOS.ARPA.       99999999    IN    A     128.20.1.2
A.ISI.EDU.          99999999    IN    A     26.3.0.103
BRL-AOS.ARPA.       99999999    IN    A     192.5.25.82
GUNTER-ADAM.ARPA.   99999999    IN    A     26.1.0.13
C.NYSER.NET.        99999999    IN    A     128.213.5.17
TERP.UMD.EDU.       99999999    IN    A     10.1.0.17
```

named.local

```
@     IN    SOA    ucbvax.Berkeley.Edu. kjd.ucbvax.Berkeley.Edu. (
                   1        ; Serial
                   10800    ; Refresh
                   1800     ; Retry
                   3600000 ; Expire
                   86400 )   ; Minimum
      IN    NS     ucbvax.Berkeley.Edu.
1     IN    PTR    localhost.
```

hosts

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

@                IN    SOA     ucbvax.Berkeley.Edu. kjd.monet.Berkeley.Edu. (
                                 1.1       ; Serial
                                 3600      ; Refresh
                                 300       ; Retry
                                 3600000   ; Expire
                                 3600 )    ; Minimum
                 IN    NS      ucbarpa.Berkeley.Edu.
                 IN    NS      ucbvax.Berkeley.Edu.
localhost        IN    A       127.1
ucbarpa          IN    A       128.32.4
                 IN    A       10.0.0.78
                 IN    HINFO   VAX-11/780 UNIX
arpa             IN    CNAME   ucbarpa
ernie            IN    A       128.32.6
                 IN    HINFO   VAX-11/780 UNIX
ucbernie         IN    CNAME   ernie
monet            IN    A       128.32.7
                 IN    A       128.32.130.6
                 IN    HINFO   VAX-11/750 UNIX
ucbmonet         IN    CNAME   monet
```

```
ucbvax          IN   A       10.2.0.78
                IN   A       128.32.10
                IN   HINFO   VAX-11/750 UNIX
                IN   WKS     128.32.0.10 UDP syslog route timed domain
                IN   WKS     128.32.0.10 TCP ( echo telnet
                             discard sunrpc sftp
                             uucp-path systat daytime
                             netstat qotd nntp
                             link chargen ftp
                             auth time whois mtp
                             pop rje finger smtp
                             supdup hostnames
                             domain
                             name server )
vax             IN   CNAME   ucbvax
toybox          IN   A       128.32.131.119
                IN   HINFO   Pro350 RT11
toybox          IN   MX      0  monet.Berkeley.Edu
miriam          IN   MB      vineyd.DEC.COM.
postmistress    IN   MR      Miriam
Bind            IN   MINFO   Bind-Request kjd.Berkeley.Edu.
                IN   MG      Ralph.Berkeley.Edu.
                IN   MG      Zhou.Berkeley.Edu.
                IN   MG      Painter.Berkeley.Edu.
                IN   MG      Riggle.Berkeley.Edu.
                IN   MG      Terry.pa.Xerox.Com.
```

hosts.rev

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

@       IN   SOA   ucbvax.Berkeley.Edu. kjd.monet.Berkeley.Edu. (
                   1.1      ; Serial
                   10800    ; Refresh
                   1800     ; Retry
                   3600000  ; Expire
                   86400 )  ; Minimum
        IN   NS    ucbarpa.Berkeley.Edu.
        IN   NS    ucbvax.Berkeley.Edu.
4.0     IN   PTR   ucbarpa.Berkeley.Edu.
6.0     IN   PTR   ernie.Berkeley.Edu.
7.0     IN   PTR   monet.Berkeley.Edu.
10.0    IN   PTR   ucbvax.Berkeley.Edu.
6.130   IN   PTR   monet.Berkeley.Edu.
```

Additional Sample Files

The following sections contain an additional set of sample files for the name server.

named.boot

```
;
; Name Server boot file for Domain sco.COM
;
; Type        Domain        Source file or Host
;
domain        sco.COM
primary       sco.COM       /etc/named.data/sco-hosts
cache         .             /etc/named.data/root.cache
secondary     sco.COM       /etc/named.data/sco-host.s.rev
primary       sco.COM       /etc/named.data/named.local
```

root.cache

```
;
; Initial cached data for root domain servers.
;
;.                99999999 IN NS    USC-ISIB.ARPA.
;                 99999999 IN NS    BRL-AOS.ARPA.
;                 99999999 IN NS    SRI-NIC.ARPA.
;
; Insert your own name servers here
;
;                 99999999 IN NS    scovert.sco.COM
;
; Prep the cache (hotwire the addresses)
;
tandy.sco.COM.          99999999 IN A     192.9.200.2
;viscous.sco.COM. 99999999 IN A     128.0.21.6
;
; Root servers go here
;
tandy.sco.COM.          99999999 IN A     192.9.200.2
;SRI-NIC.ARPA.          99999999 IN A     10.0.0.51
;USC-ISIB.ARPA.         99999999 IN A     10.3.0.52
;BRL-AOS.ARPA.          99999999 IN A     128.20.1.2
;BRL-AOS.ARPA.          99999999 IN A     192.5.22.82
```

named.local

```
;
; Don't forget to increment the serial number in
; named.soa
;

$INCLUDE /etc/named/sco.soa
192.9.200.2 IN PTR      localhost.
```

sco-host.s.rev

```
;
; Don't forget to increment the serial number in
; named.soa
;

$INCLUDE /etc/named/sco.soa

192.9.200.1 IN    PTR   merlin
192.9.200.2 IN    PTR   tandy
192.9.200.3 IN    PTR   tvi
```

sco.soa

```
;
; Don't forget to increment the serial number when you
; change this.  SCCS or RCS might be a good idea here.
;

@           IN SOA  tandy.sco.COM.   root.tandy.sco.COM. (
                1.0     ; Serial
                3600    ; Refresh
                300     ; Retry
                3600000 ; Expire
                3600 )  ; Minimum
            IN NS   tandy.sco.COM.
```

Domain Management

This section contains information for starting, controlling, and debugging **named**(ADMN), the Internet domain name server.

Starting the Name Server

The host name should be set to the full domain style name (that is, monet.Berkeley.EDU.) using **hostname**(TC). The name server is started automatically if the configuration file */etc/named.boot* is present. Do not attempt to run **named** from **inetd**(ADMN). This continuously restarts the name server and defeats the purpose of having a cache.

/etc/named.pid

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

/etc/hosts

The **gethostbyname** library call can detect whether **named** is running. If it is determined that **named** is not running, it looks in */etc/hosts* to resolve an address. This option was added to allow **ifconfig**(ADMN) to configure the machine's local interfaces and to enable a system manager to access the network while the system is in single-user mode. It is advisable to put the local machine's interface addresses and a couple of machine names and addresses in */etc/hosts*, so the system manager can copy files from another machine with **rcp** when the system is in single-user mode. The format of */etc/hosts* has not changed. See **hosts**(SFF) for more information. Because the process of reading */etc/hosts* is slow, it is not advisable to use this option when the system is in multiuser mode.

Reload

There are several signals that can be sent to the **named** process to have it do tasks without restarting the process. The SIGHUP signal causes **named** to read *named.boot* and reload the database. All previously cached data is lost. This is useful when you have made a change to a data file and you want **named** 's internal database to reflect the change.

Debugging

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

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

The following two signals only work when **named** *is built with DEBUG defined.*

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

SIGUSR2 - Turns off debugging completely.

For more detailed debugging, define DEBUG when compiling the resolver routines into */usr/lib/libsocket.a.*

Chapter 5

Synchronizing Network Clocks

Introduction

The clock synchronization service is composed of a collection of time daemons (**timed**(ADMN)) running on the machines in a local-area network. The algorithms implemented by the service are based on a master-slave scheme. The time daemons communicate with each other using the Time Synchronization Protocol (TSP), which is built on the DARPA UDP protocol.

A time daemon has a two-fold function. First, it supports the synchronization of the clocks of the various hosts in a local-area network. Second, it starts (or takes part in) the election that occurs among slave time daemons when, for any reason, the master disappears. The synchronization mechanism and the election procedure employed by the program **timed** are described in the manual page **timed**(ADMN). This chapter is mainly concerned with the administrative and technical issues of running **timed** at a particular site. The next section is a brief overview of how the time daemon works. A master time daemon measures the time differences between the clock of the machine on which it is running and those of all other machines on its network. The master computes the network time as the average of the times provided by nonfaulty clocks. (A clock is considered to be faulty when its value is more than a small specified interval apart from the majority of the clocks of the other machines.) The master time daemon then sends to each slave time daemon the correction that should be performed on the clock of its machine. This process is repeated periodically.

Because the correction is expressed as a time difference rather than an absolute time, transmission delays do not interfere with the accuracy of the synchronization. When a machine comes up and joins the network, it starts a slave time daemon that asks the master for the correct time and resets the machine's clock before any user activity can begin. The time daemons are thus able to maintain a single network time in spite of the drift of clocks away from each other. The present implementation is capable of keeping processor clocks synchronized to within 20 milliseconds, but some hardware is not adjustable at less than 1 second intervals.

To ensure that the service provided is continuous and reliable, it is necessary to implement an election algorithm to elect a new master should the machine running the current master crash, the master terminate (for example, because of a runtime error), or the network be partitioned.

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

The machines that are gateways between distinct local-area networks require particular care. A time daemon on such machines may act as a "submaster." This artifact depends on the current inability of transmission protocols to broadcast a message on a network other than the one to which the broadcasting machine is connected. The submaster appears as a slave on one network and as a master on one or more of the other networks to which it is connected.

A submaster classifies each network as one of three types. A slave network is a network on which the submaster acts as a slave. There can only be one slave network. A master network is a network on which the submaster acts as a master. An ignored network is any other network that already has a valid master. The submaster tries periodically to become master on an ignored network, but gives up immediately if a master already exists.

Guidelines

While the synchronization algorithm is quite general, the election algorithm, which requires a broadcast mechanism, puts constraints on the kind of network on which time daemons can run. The time daemon works only on networks with broadcast capability augmented with point-to-point links. Machines that are only connected to point-to-point, non-broadcast networks cannot use the time daemon.

If submasters are excluded, there is normally only one master time daemon in a local-area internetwork. During an election, only one of the slave time daemons becomes the new master. Not all machines are suitable as masters; some do not have sufficiently accurate timing mechanisms or cannot afford the extra overhead. Therefore, a subset of machines must be designated as potential master time daemons. A master time daemon requires CPU resources proportional to the number of slaves (in general, more than a slave time daemon), and so it may be advisable to limit master time daemons to machines with more powerful processors or lighter loads. Also, machines with inaccurate clocks should not be used as masters. This is a purely administrative decision; an organization may well allow all of its machines to run master time daemons.

At the administrative level, a time daemon on a machine with multiple network interfaces may be told to ignore all but one network or to ignore one network. This is done with the **timed -n** *network* and **-i** *network* options, respectively, at startup time. Typically, the time daemon would be instructed to ignore all but the networks belonging to the local administrative control.

There are some limitations to the current implementation of the time daemon. It is expected that these limitations will be removed in future releases. The constant NHOSTS in */usr/src/etc/timed/globals.h* limits the maximum number of machines that can be directly controlled by one master time daemon. The maximum is (NHOSTS - 1). Currently, the maximum is 99. The constant must be changed and the program recompiled if a site wishes to run **timed** on a larger network.

In addition, there is a pathological situation to be avoided at all costs. This situation can occur when time daemons run on multiply-connected local-area networks. In this case, time daemons running on gateway machines are submasters, and they act on some of those networks as master time daemons. Consider machines A and B that are both gateways between networks X and Y. If time daemons were started on both A and B without constraints, it would be possible for submaster time daemon A to be a slave on network X and the master on network Y, while submaster

time daemon B would be a slave on network Y and the master on network X. This loop of master time daemons does not function properly or guarantee a unique time on both networks, and it causes the submasters to use large amounts of system resources in the form of network bandwidth and CPU time. In fact, this kind of loop can also be generated with more than two master time daemons, when several local-area networks are interconnected.

Options

The options for the **timed** command are:

-n *network*	Considers the named network.
-i *network*	Ignores the named network.
-t	Places tracing information in */usr/adm/timed.log*.
-M	Allows this time daemon to become a master. A time daemon run without this option is forced into the state of slave during an election.

Daily Operation

The **timedc(ADMN)** command is used to control the operation of the time daemon. It can be used to do the following:

- measure the differences between machines' clocks
- find the location where the master **timed** is running
- cause election timers on several machines to expire at the same time
- enable or disable tracing of messages received by **timed**

See the manual pages on **timed**(ADMN) and **timedc**(ADMN) for more detailed information.

The **rdate(ADMN)** command can be used to set the network date.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

P

Q

R

S

T

U

W