intel®

Intel® TCP/IP for System V/386 Administrator's Guide and Reference

Order Number: 464123-001

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

Additional copies of this manual or other Intel literature may be obtained by contacting your Intel sales office.

In locations outside the United States, obtain additional copies of Intel documentation by contacting your local Intel sales office. For your convenience, international sales office addresses are located directly after the reader reply card in the back of this manual.

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update or to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than the circuitry embodied in an Intel product. No other circuit patent licenses are implied.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

Above	iMDDX	iPSC	Plug-A-Bubble
BITBUS	iMMX	iRMX	PROMPT
COMMputer	Insite	iSBC	Promware
CREDIT	intel	iSBX	QUEST
Data Pipeline	Intel376	iSDM	QueX
GENIUS	Intel386	iSSB	Ripplemode
ƒ	intelBOS	iSXM	RMX/80
i	Intelevision	Library Manager	RUPI
I2ICE	inteligent Identifier	MCS	Seamless
ICE	inteligent Programming	Megachassis	SLD
iCEL	Intellec	MICROMAINFRAME	UPI
iCS	Intellink	MULTIBUS	VLSiCEL
iDBP	i860	MULTICHANNEL	376
iDIS	iOSP	MULTIMODULE	386
iLBX	iPDS	ONCE	386SX
im	iPSB	OpenNET	486
			860

XENIX, MS-DOS, Multiplan, and Microsoft are trademarks of Microsoft Corporation. UNIX is a trademark of Bell Laboratories. Ethernet is a trademark of Xerox Corporation. Centronics is a trademark of Centronics Data Computer Corporation. Chassis Trak is a trademark of General Devices Company, Inc. VAX and VMS are trademarks of Digital Equipment Corporation, Smartmodem 1200 and Hayes are trademarks of Hayes Microcomputer Products, Inc. IBM is a registered trademark of International Business Machines. MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.

Credits and Trademarks

System V/386 TCP was developed by
Lachman Associates, Inc,
in joint development with Convergent Technologies, Inc.

REV.	REVISION HISTORY	DATE
-001	Original Issue.	10/89

SYSTEM V/386 TCP

ADMINISTRATOR'S GUIDE AND REFERENCE

CONTENTS

Chapter 1

INTERNETWORKING CONCEPTS

Chapter 1

PAGE

Chapter 1

INTERNETWORKING CONCEPTS

1. INTERNETWORKING CONCEPTS

This chapter introduces some basic network concepts relating to UNIX internetworking, TCP/IP, and network gateways. Throughout the TCP documentation, you will find references to various "RFC" documents. RFC stands for Request for Comment and the RFC documents are published and available from the Network Information Center of SRI International at 333 Ravenswood Avenue, Room EJ291 in Menlo Park, California. If any of the terms in this manual are new to you, look for the term in the glossary at the end of the manual.

2. Overview

TCP (Transmission Control Protocol) and IP (Internet Protocol) form an end-to-end transmission and routing protocol that supports UNIX commands and applications. TCP/IP was originally developed for the Department of Defense (DoD) for use in ARPANET (Advanced Research Projects Agency Network), the first major network to use packet-switching technology. *Packet switching* is an alternative to the circuit switching technology used in telephone and telex systems in which a dedicated communication path is allocated to communications between two users for the duration of a communication. Packets are also called datagrams. A single message can be broken up into several packets, or datagrams, before being sent to the destination.

The Defense Data Network (DDN) is based on ARPANET standard TCP/IP. The DDN is the implementation stage in the evolution of work done by ARPA on packet switching network technology. (SRI, under contract with the DoD, publishes a standard for TCP/IP.)

Currently, TCP/IP is also widely used in UNIX environments and is offered by virtually every UNIX operating system vendor and every ethernet board manufacturer. UNIX TCP/IP is similar to and usually compatible with other systems running TCP/IP over Ethernet including DOS and VMS systems.

2.1 The OSI Model

Although UNIX Internetworking is not strictly an implementation of the well known seven-layer OSI network model, it is helpful to use that model in explaining UNIX Internetworking concepts.

The seven layers of the OSI model are: physical, link, network, transport, session, presentation and application.

Layer four in UNIX Internetworking can be implemented in different protocols. In this chapter, TCP is used as an example of layer four and is explained in some detail. The UNIX Internet Protocol (IP) which implements layer three is also explained. TCP/IP makes possible the end-to-end nature of UNIX Internetworking.

The data link layer is implemented in UNIX Internetworking as Media protocols, such as Ethernet. They are often implemented in hardware.

There are also several higher level specialized protocols for specific applications such as terminal traffic (*telnet*) and file transfer (*ftp*) and protocols for other network functions such as gateway status monitoring and control and error reporting. In this manual, these programs are usually referred to as programs or services. The same applies to the link level protocols, such as Ethernet, which are referred to as media protocols or simply as media.

2.2 HOW PROTOCOLS COMMUNICATE

Protocols communicate logically only with their counterpart protocols and physically only with the layers directly above and below themselves. For example, TCP, at layer 4, the Transport layer of the local host, communicates logically only with its counterpart TCP in the remote host, but, in doing so, the TCP communicates physically with the layer directly beneath it (the IP at the network layer 3). IP passes the message from TCP down to its neighboring layer (the data link layer), and so on. The physical layer (layer 1) transmits the message to the remote host.

The message passes up through the layers of the remote host to arrive at the TCP which is able to decode the message and disposition it appropriately. The intervening layers are unaware of the contents of the message and only decode its destination address to the extent they must to pass it to the

next layer.

Each layer therefore must understand the language of the adjoining upper and lower layers in addition to performing its own proper functions.

2.3 GATEWAYS

An internetwork is a supernetwork made up of two or more networks able to communicate with each other through gateways. A gateway is a software service installed at a switching node that connects two or more networks, even if they use different protocols (If a gateway runs on a dedicated processor, the processor is also considered a component of the gateway. An internetwork using TCP and IP can be composed of different "UNIX-compatible" networks running different local protocols but tied together by a higher level end-to-end protocol such as TCP/IP. To unify a potentially diverse networks into a single internetwork, network gateways are used.

2.3.1 GATEWAY ROUTING PROTOCOLS

Two networks connected by a gateway can employ the same protocols or different protocols. If the same, the gateway must contain modules of the protocol being used. For example, if the networks both use TCP/IP, the gateway must contain TCP and IP modules. If the two networks utilized different protocols, the gateway must contain services capable of converting from one network protocol to another.

When a message arrives at a gateway, the gateway determines the destination network and encapsulates the internet protocol with the protocol header appropriate to the new network.

2.3.2 IP GATEWAY PROTOCOL

The network level protocol, IP, is designed to act as a gateway. All UNIX Internetworking nodes, including gateways, contain an internet module. This IP module contains the appropriate software services for switching and retransmitting packets to their next gateway, to the next network, or to the destination host. Gateways are placed wherever necessary to implement the desired topology and configuration of an internet. A gateway is usually one of the network hosts but it can reside in its own dedicated processor.

3. TCP AND IP PROTOCOLS

TCP and IP are actually separate protocols that work together implementing the major lower level functions in UNIX internetworking.

3.1 TRANSMISSION CONTROL PROTOCOL (TCP)

Transmission Control Protocol is a transport level, connection-oriented protocol that provides highly reliable end-to-end message transmission between hosts in packet-switched networks and in interconnected systems, or internets. TCP interfaces on one side with user or applications processes and on the other side to a lower level protocol such as Internet Protocol (IP). TCP communicates asynchronously with the application process and the IP or other network level protocols. TCP is totally byte-stream oriented.

TCP operates at a sufficiently high level (OSI level 4 and 5) to be media independent. Lower layers can support hard-wired (direct), circuit-switched, and packet-switched communications links. Thus an internetwork using TCP to provide an end-to-end transport service can be made up of multiple subnets using a variety of media protocols.

A TCP module resides at each node in the internet that communicates with other TCP nodes.

To provide its service under the expected circumstances of the datagram model, TCP implements mechanisms for the following functions:

- support for interprocess communication (IPC) and connection functions
- basic data transfer and transmission
- end-to-end reliability
- multiplexing, flow control, and message sequencing
- precedence and security
- out of band data

3.1.1 Interface with Application Process

TCP provides the basis for UNIX interprocess communications over the internet. The interface between TCP and application processes consists of a set of calls much like the calls an operating system provides to a process or manipulating files. For example, there are calls to open and close connections and to send and receive data on established connections.

3.1.2 TCP Messages and IP Packets

TCP Messages are constructed from one or more IP packets or datagrams. If there is more than one datagram in a message, we say the message has been "fragmented", and the individual IP datagrams are called fragments. Datagrams are routed individually and dynamically over the best available routing path. They are then reassembled at the receiving end to recreate the original message.

3.1.3 TCP Ports and the Socket Interface

TCP provides a set of numbered ports to identify and be used by the calling processes. A TCP port is not a hardware communications port such as an RS-232-C port. A TCP port is the portion of a socket

that specifies which logical input or output channel of a process is associated with the data.

A socket is an address which specifically includes a port identifier, that is, the concatenation of an internet address with a TCP port. Port connections are displayed in the Active Connections Display of *netstat*(1M).

For more information on sockets and how TCP uses them, see the Socket Programmer's Primer chapter in the *TCP Programmer's Guide and Reference Manual.*

3.1.4 TCP and Reliable Transmission

The primary purpose of TCP is to provide a reliable, secure, virtual circuit connection service between pairs of communicating processes. Security provisions such as limiting user access to certain nodes can be implemented at the TCP layer.

TCP is concerned only with total end-to-end reliability. It makes few assumptions about the possibility of obtaining reliable datagram service from the lower protocols. If a datagram is sent across an internet to a remote node, the intervening networks do not guarantee delivery. Likewise, the sender of the datagram has no way of knowing the routing path used to send the datagram. Source to destination reliability is provided by TCP.

Reliability is achieved through checksums (error detection codes), positive acknowledgment of data received, and retransmission of unacknowledged data.

3.1.5 Flow Control

Flow control is accomplished by allowing the receiver to regulate the data rate. TCP provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission.

4. INTERNET PROTOCOL

Internet Protocol is the network level protocol designed for packet-switched networks. The IP limits itself to only the delivery of datagrams through an internet. (Reliability is the responsibility of TCP) As with TCP, there must be an Internet Protocol module at each node and each gateway that communicates using Internet Protocol. This module is called the internet module.

The IP addresses, routes, and forwards datagrams to the next gateway or destination host via the local network interface. Internet gateway functions are performed at the IP layer.

IP functions are:

- addressing
- security classification, and compartmentation of TCP segments
- internet datagram routing
- communications with gateways and host protocol modules
- fragmentation and reassembly

4.1 Addressing

Each Internet host has a unique 32-bit Internet address. An Internet address consists of a class identifier, a network identifier and a host identifier. There are three different classes of Internet addresses, A, B and C. Different classes assign different numbers of bits to the network and host identifiers. This is to allow for differing network sizes. The breakdown is as follows:

Internet Addresses			
Class	Class Identifier (in binary)	Network	Host
A	1 bit(always 0)	7 bits	24 bits
B	2 bits(always 10)	14 bits	16 bits
C	3 bits(always 110)	22 bits	8 bits

As you can see from the above table, the three parts of the Internet address in each class have a total of 32 bits. With the Class A adressing scheme, there are 2^7 (or 128) possible networks with 2^{24} (or 16 million) possible hosts on each of those networks, and each host would have a unique address. As you can see, this system provides a unique address for the entire statistical distribution that might be expected in the total population of networks using this address system. There would be a smaller number of large networks, having many nodes (Class A), and a larger number of small networks, consisting of a lesser number of nodes (Class C), and a medium number of networks made up of a medium number of nodes (Class B).

Because each network can have a particular address format and length (class A, B, or C), the IP maps between the internet local addresses and the actual address format used in the particular network.

The internet module maps internet addresses to local net addresses. Local nets and gateways map from local net addresses to routes. A host can have several physical interfaces to different networks in the internetwork with each interface having its own logical internet address.

TCP and User Datagram Protocol (UDP) additionally use a 16-bit number called the *port* to address a connection.

4.2 Hosts Having Multiple Addresses

In the case of a host that is on multiple networks, the host serves as a gateway to each of the networks and to itself. It has a primary address which it uses to address itself.

If the packet is for itself (a local process), the host sends the packet to the loopback driver of the routing code, which chains the output back to the input.

4.2.1 Routing

An internet module can be located in a gateway or a network host. It decides the routing path for the datagram, packages it with the next address, and forwards the resulting internet datagram to the next gateway or to the destination host. IP treats each datagram as an independent entity during all phases of routing. TCP sees the datagram only at the endpoints. Internet modules reassemble datagrams into the original messages only at the destination host.

An internet module packages received datagrams with the appropriate internet header, containing the address of the final destination. Such a datagram is called an internet datagram.

At the destination host, the local network interface strips the internet datagram of its local net header and hands it to the internet module.

The internet module determines whether the datagram is for an application system in the local host. The internet module passes the message (after reassembling it if necessary) to the application system in response to a system call. If the datagram is not for the local host, the IP passes it on.

4.2.2 Fragmentation

In a packet-switched network, packets can be sent by different routes and are retransmitted if necessary. If datagrams arrive at the destination host out of sequence, the IP reassembles them into the original message for the destination host. Some networks have different message sizes. If necessary at the destination host, the IP fragments incoming datagrams to the size required by the destination network.

Chapter 2

SETTING UP THE NETWORK

Chapter 2

PAGE

Chapter 2

NETWORK ADMINISTRATION

1. Introduction

UNIX provides support for the DARPA standard Internet protocols IP, ICMP, TCP, and UDP. These protocols may be used on top of a variety of hardware devices including local area network controllers for the Ethernet. Network services are split between the kernel (communication protocols) and user programs (such as *TELNET* and *FTP*). This section describes how to configure your system to use the Internet networking support.

2. System configuration

2.1 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
rip	/dev/inet/rip	Raw IP
slip	/dev/slip	Serial Line IP interface
llcloop	/dev/llcloop	Loopback interface
socket	/dev/socksys	Socket compatibility package
vty	/dev/ptyp*nn*	Virtual TTY driver†
ttyp	/dev/ttyp*nn*	

These drivers must be included in the **/etc/conf/cf.d/mdevice** file.

In addition to the above-mentioned drivers, you must also include drivers for your network interface hardware.

2.2 STREAMS Configuration

STREAMS configuration (linking the various STREAMS drivers and modules together) is handled by *slink*(1M), which is normally executed at boot time by *tcp*(1M). *Slink* reads the file */etc/strcf*, which contains a list of STREAMS operations to perform. Most of */etc/strcf* will be the same on every system. However, it will be necessary to edit the section of */etc/strcf* which configures the network interfaces. Examples for various types of network drivers are provided. In some cases, it will be necessary to write new driver setup procedures. See *slink*(1M) and *strcf*(4) for further information.

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

2.2.1 Non-cloning Drivers

Drivers of this type, such as the *pc586* driver, have a separate device node for each minor device, with some fixed number of minor devices allocated to each network interface. The *slink* functions **senet** and **senetc** are used for this driver type (**senetc** allows the specification of a convergence module). The following command line configures such an interface:

```
senet ip /dev/pc586_0 /dev/pc586_1 en0
```

If a convergence module is required, use **senetc** in place of **senet** and include "eli".

---

† The Virtual TTY driver is used by *rlogin*(1) and *telnet*(1). 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*, *ttyp* minor 0 is referenced by device node */dev/ttyp00*, etc.

The last argument ("en0" in this example) gives the name that the newly-created interface will be known by for the purpose of performing interface configuration operations via *ifconfig*(1M); refer to the section entitled "Interface Configuration" for further information.

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

```
senet ip /dev/pc586_4 /dev/pc586_5 en1
```

2.2.2 Cloning Drivers with One Major Number Per Interface

Drivers of this type, such as the Interlan *ni* 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** is used to configure an interface of this type. The command line to configure such an interface has the form:

```
cenet ip arp /dev/ni en 0
```

To add a second interface, add the line

```
cenet ip arp /dev/ni en 1
```

Note that the device node actually used is formed by concatenating the given device node name prefix ("/dev/ni") and the given unit number ("0" or "1"). The interface name is formed in a similar manner using the supplied interface name prefix ("en") and the unit number. Thus, the first example configures an interface named "en0" which accesses the device referred to by */dev/ni0*.

2.2.3 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 desired interface is selected using either the *unit select* or the *DL_ATTACH* primitive (normally, a given driver will only recognize one of these primitives). The *slink* functions **uenet** and **denet** are used to 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 arp /dev/abc en 0
```

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

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

Denet and **uenet** form the interface name in the same manner as does **cenet** (see above), but the device node name is unchanged (i.e., **/dev/abc** will be opened in both of these examples).

2.3 Interface Configuration

All network interface drivers, including the loopback interface, require that their host addresses be defined at boot time. This is done with *ifconfig*(1M) commands included in the */etc/tcp* shell script. *Ifconfig*(1M) 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 may be set in advance or "published" by a UNIX host by use of the *arp*(1M) command.

2.4 Local subnetworks

In UNIX the DARPA Internet support includes the notion of "subnetworks". This is a mechanism by which multiple local networks may appear as a single Internet network to off-site hosts. Subnetworks are useful because they allow a site to hide their local topology, requiring only a single route in external gateways; it also means that local network numbers may be locally administered.

To set up local subnetworks one must first decide how the available address space (the Internet "host part" of the 32-bit address) is to be partitioned. 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, while 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; i.e. it is not visible to hosts off-site. 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. (The use of subnets 0 and all-1's, 255 in this example, is discouraged to avoid confusion about broadcast addresses.) These 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 at the time a network interface is configured with the *netmask* option to the *ifconfig* program. A "network mask" is specified to define 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 has been assigned to subnets. If no mask is specified when the address is set, it will be set according to the class of the network. For example, at Berkeley (class B network 128.32) 8 bits of the local part have been reserved for defining subnetworks; consequently the */etc/tcp* file contains lines of the form

```
/etc/ifconfig en0 netmask 0xffffff00 128.32.1.7
```

This specifies that for interface "en0", the upper 24 bits of the Internet address should be used in calculating network numbers (netmask 0xffffff00), and the interface's Internet address is "128.32.1.7" (host 7 on network 128.32.1). Hosts *m* on sub-network *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.

2.5 Internet broadcast addresses

The address defined as the broadcast address for Internet networks according to RFC-919 is 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. UNIX 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 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). UNIX 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.

---

* If you are unfamiliar with the Internet addressing structure, consult "Internet Numbers", Internet RFC-1062, J. Postel; available from the Internet Network Information Center at SRI.

2.6 Routing

If your environment allows access to networks not directly attached to your host you will need to set up routing information to allow packets to be properly routed. Two schemes are supported by the system. The first scheme employs the routing table management daemon *routed*(1M) 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 be used to initialize static routes to distant networks (see the next section for further discussion). When the routing daemon is started up (usually from */etc/tcp*) it reads **/etc/gateways** if it exists and installs those routes defined there, 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**; consult *routed*(1M) 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 dynamically create a routing data base. This is done by adding an entry of the form

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

to */etc/tcp*; see *route*(1M) for more information. The default route will be used by the system as a "last resort" in routing packets to their destination. Assuming the gateway to which packets are directed is able to generate the proper routing redirect messages, the system will then add routing table entries based on the information supplied. This approach has certain advantages over the routing daemon, but is unsuitable in an environment where there are only bridges (i.e. pseudo gateways that, for instance, 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 maintaining service.

The system always listens, and processes, routing redirect information, 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*(1M) program may be used to display routing table contents as well as various routing oriented statistics. For example,

```
netstat -r
```

will display the contents of the routing tables, while

```
netstat -r -s
```

will show the number of routing table entries dynamically created as a result of routing redirect messages, etc.

2.7 Use of UNIX Machines as Gateways

Any UNIX machine that has more than one non-loopback network interface will function as a gateway, in that packets received on one network that are destined for a host on another network will be automatically forwarded. If a packet cannot be forwarded to the desired destination, an ICMP error message is sent to the originator of the packet. If a packet is forwarded back through the same 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 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 may be handled by *routed*(1M). 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.

2.8 Network Servers

In UNIX most of the server programs are started up by a "super server", the Internet daemon. The Internet daemon, */etc/inetd*(1M), acts as a master server for programs specified in its configuration file, */etc/inetd.conf*(4), listening for service requests for these servers, and starting up the appropriate program whenever a request is received. The configuration file contains lines containing a service name (as found in */etc/services*(4)), the type of socket the server expects (e.g. stream or dgram), the protocol to be used with the socket (as found in */etc/protocols*(4)), whether to wait for each server to complete before starting up another, the user name as 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*, 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*(1M) for more detail on the format of the configuration file and the operation of the Internet daemon.

2.9 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.

File	Manual reference	Use
/etc/hosts	*hosts* (5)	host names
/etc/networks	*networks* (5)	network names
/etc/services	*services* (5)	list of known services
/etc/protocols	*protocols* (5)	protocol names
/etc/hosts.equiv	*rshd* (1M)	list of "trusted" hosts
/etc/ftpusers	*ftpd* (1M)	list of "unwelcome" ftp users
/etc/inetd.conf	*inetd* (1M)	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 (see also the next section). Network numbers will have to 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.

2.9.1 /etc/hosts.equiv

The remote login and shell servers use an authentication scheme based on trusted hosts. The *hosts.equiv*(4) file contains a list of hosts that are considered trusted and, under a single administrative control. When a user contacts a remote login or shell server requesting service, the client process passes the user's name and the official name of the host on which the client is located. In the simple case, if the host's name is located in *hosts.equiv* and the user has an account on the server's machine, then service is rendered (i.e. the user is allowed to log in, or the command is executed). Users may expand this "equivalence" of machines by installing a *.rhosts*(4) file in their login directory. The root login is handled specially, bypassing the *hosts.equiv* file, and using only the */.rhosts* file.

Thus, to create a class of equivalent machines, the *hosts.equiv* file should contain the *official* names for those machines. If you are running the name server, you may omit the domain part of the host name for machines in your local domain. For example, several machines on our local network are considered trusted, so the *hosts.equiv* file is of the form:

```
ucbarpa
calder
dali
ernie
kim
matisse
monet
ucbvax
miro
degas
```

2.9.2 /etc/ftpusers

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 consider providing such a service.

An anonymous account is enabled by creating a user *ftp*. When a client uses the anonymous account a *chroot* (2) system call is performed by the server to restrict the client from moving outside that part of the file system where the user 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, one 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 dev lib
# 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 wish to place files in the anonymous area, they must be placed in a subdirectory. In the setup here, the directory *~ftp/pub* is used.

Another issue to consider is the copy of */etc/passwd* placed here. It may be copied by users who use the anonymous account. They may 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 may provide 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. This file normally has the following names on our systems.

```
uucp
root
```

Accounts with nonstandard shells should be listed in this file. Accounts without passwords need not be listed in this file; for security reasons, the ftp server will not service these users.

3. Netstat and Network Troubleshooting

If you have anything more than a trivial network configuration, from time to time you are bound to run into problems. Before blaming the software, first check your network connections. On networks such as the Ethernet a loose cable tap or misplaced power cable can result in severely deteriorated service. The *netstat* (1M) program may be of aid in tracking down hardware malfunctions. In particular, look at the **-i** and **-s** options in the manual page.

Should you believe a communication protocol problem exists, consult the protocol specifications and attempt to isolate the problem in a packet trace. The SO_DEBUG option may be supplied before establishing a connection on a socket, in which case the system will trace all traffic and internal actions (such as timers expiring) in a circular trace buffer. This buffer may then be printed out with the *trpt* (1M) 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.

3.1 Active Connections

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

3.2 netstat -a

```
laioff$ netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address      Foreign Address  (state)
ip       0    0 *.*                *.*
tcp      0    0 laioff.telnet      laiter.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 laioff.listen      *.*          LISTEN
tcp      0    0 laioff.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 laioff.domain      *.*
udp      0    0 localhost.domain   *.*
laioff$
```

3.2.1 Descriptions of the Display Headings

Proto	The protocol used in the connection
Recv-Q	Receive queue. The number of received characters (bytes) of data waiting to be processed.
Send-Q	Send queue. The number of characters (bytes) of data waiting to be transmitted.
Local	The port number of the local connection, displayed symbolically. The port numbers are taken from the **/etc/services** file.
Foreign	The port number of the remote connection, displayed symbolically. The port numbers are taken from the **/etc/services** file.
State	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.

3.3 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.

3.4 netstat -i

```
laioff$ netstat -i
Name Mtu  Network    Address     Ipkts  Ierrs Opkts  Oerrs Collis
en0  1500 lai-eng-ne laioff      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
laioff$
```

3.4.1 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 as given 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 have occurred on the network

3.5 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.

3.6 netstat -r

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

3.6.1 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 the route is 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.

3.7 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)

3.8 netstat -s

```
$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
        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
        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 3

SENDMAIL

INSTALLATION AND OPERATION GUIDE

Chapter 3

PAGE

LIST OF FIGURES

Chapter 3

SENDMAIL
INSTALLATION AND OPERATION GUIDE †

Sendmail 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. In the 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. These features are described.

Section one describes how to do a basic *sendmail* installation. Section two explains the day-to-day information you should know to maintain your mail system. If you have a relatively normal site, these two sections should contain sufficient information for you to install *sendmail* and keep it happy. Section three describes some parameters that may be safely tweaked. Section four has information regarding the command line arguments. Section five contains the nitty-gritty information about the configuration file. This section is for masochists and people who must write their own configuration file. Sections six through nine give a brief but detailed explanation of a number of features not described in the rest of the chapter.

† This chapter is based on the document of the same name, written by Eric Allman of Britton-Lee, Inc.

1. BASIC INSTALLATION

There are two basic steps to installing sendmail. The hard part is to build the configuration table. This is a file that sendmail reads when it starts up that describes the mailers it 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 part is actually doing the installation, i.e., creating the necessary files, etc.

The remainder of this section will describe the installation of sendmail assuming you can use one of the existing configurations and that the standard installation parameters are acceptable.

All *sendmail*-related files are found in the directory **/usr/lib**.

1.1 Off-The-Shelf Configurations

Two sample configuration files are included with this release. They are: **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.

The sample configuration file you need should be copied to **sendmail.cf**, e.g.,

```
cp relay.cf sendmail.cf
```

There are some variables that need to be changed in **sendmail.cf**. 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.

Use a text editor such as *vi(1)* to replace the variables specified above with the appropriate replacement values, e.g.,

```
:%s/ZZDOMAIN/Lachman/g
```

Note that some variables appear in the file more than once and that ZZRELAY must be substituted with a fully qualified Internet address of the major mail relay machine, e.g.,

```
laidbak.Lackman.COM
```

To start the *sendmail* daemon, a line is usually added to one of the system startup scripts, e.g,

```
/usr/lib/sendmail -bd -q1h
```

Note that the TCP startup file **/etc/tcp** has had this line added and will start *sendmail* if the configuration file **/usr/lib/sendmail.cf** is present.

2. NORMAL OPERATIONS

2.1 Quick Configuration Startup

A fast version of the configuration file may be set up by using the **-bz** flag:

```
/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* *sendmail.fc* 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 so this should not be trusted.

2.2 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.

2.2.1 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 will produce a listing of the queue id's, the size of the message, the date the message entered the queue, and the sender and recipients.

2.2.2 Format of queue files

All queue files have the form *x*`f`*AA99999* where *AA99999* is the *id* for this file and the *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 apparently 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. These are 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 may only be one of these lines.

H A header definition. There may be any number of these lines. The order is important: they represent the order in the final message. These use 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 may 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 when to time out the job.

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 "mckusick@calder" and "wnj :"

```
DdfA13557
Seric
T404261372
P132
Rmckusick@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: mckusick@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.

2.2.3 Forcing the queue

Sendmail 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 a major host going down for a couple of days may create a prohibitively large queue. This will result in *sendmail* spending 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 to just 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
```

2.3 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 may be aliased; e.g.,

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

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

The second form is processed by the *dbm* (3) library.[1] 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.

2.3.1 Rebuilding the alias database

The DBM version of the database may 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. The conditions under which it will do this are:

1. The DBM version of the database is mode 666. -or-
2. *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

1. As shipped, *sendmail* is not configured to use the *dbm* library. If you have *dbm*, you can configure *sendmail* to use it by making a few minor changes. See the release notes for further information.

process simultaneously.

2.3.2 Potential problems

There are a number of problems that can occur with the alias database. They all result from a *sendmail* process accessing the DBM version while it is only partially built. This can happen under two circumstances: 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.

Sendmail has 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 leaving a partially rebuilt database. Second, at the end of the rebuild it adds an alias of the form

@: @

(which is not normally legal). Before sendmail will access the database, it checks to ensure that this entry exists.[2] *Sendmail* will wait for this entry to appear, at which point it will force a rebuild itself.[3]

2.3.3 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
```

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.

2.4 Per-User Forwarding (.forward Files)

As an alternative to the alias database, any user may 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 "mckusick" has a **.forward** file with contents:

```
mckusick@ernie
kirk@calder
```

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

2.5 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 builtins

---

2. The "a" option is required in the configuration for this action to occur. This should normally be specified unless you are running *delivermail* in parallel with *sendmail.*
3. Note: the "D" option must be specified in the configuration file for this operation to occur. If the "D" option is not specified, a warning message is generated and *sendmail* continues.

are described here.

2.5.1 Return-Receipt-To:

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

2.5.2 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.

2.5.3 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.

At least one recipient line is required under RFC 822.

3. ARGUMENTS

The complete list of arguments to *sendmail* is described in detail in section 6. Some important arguments are described here.

3.1 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.

3.2 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
```

3.3 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
```

3.4 Debugging

There are 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 "absurd," i.e., they print out so much information that you wouldn't 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).

3.5 Trying a Different Configuration File

An alternative configuration file can be specified using the **-C** flag; for example,

/usr/lib/sendmail -Ctest.cf

uses the configuration file *test.cf* instead of the default */usr/lib/sendmail.cf*. If the -C flag has no value it defaults to *sendmail.cf* in the current directory.

3.6 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.

4. 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.

4.1 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

4.1.1 Queue interval

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.

4.1.2 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 in certain environments (such as an hour). 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.

4.1.3 Message timeouts

After sitting in the queue for a few days, a message will time out. This is to ensure that at least the sender is aware of the inability to send a message. 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.

4.2 Forking During Queue Runs

By setting the **Y** option, *sendmail* will fork before each individual message while running the queue. This will prevent *sendmail* from consuming large amounts of memory, 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.

4.3 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 times the "work class factor" and the number of recipients times 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 amount 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 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.

4.4 Delivery Mode

There are a number of delivery modes that *sendmail* can operate in, 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 (don't 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.

4.5 File Modes

There are a number of files that may have a number of modes. The modes depend on what functionality you want and the level of security you require.

4.5.1 To suid or not to suid?

Sendmail can safely be made setuid to root. At the point where it is about to *exec* (2) 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.

4.5.2 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).

4.5.3 Should my alias database be writable?

At Berkeley we have the alias database (/usr/lib/aliases*) mode 666. There are some dangers inherent in this approach: any user can add him-/her-self to any list, or can "steal" any other user's mail. However, we have found users to be basically trustworthy, and the cost of having a read-only database greater than the expense of finding and eradicating the rare nasty person.

The database that *sendmail* actually used is represented by the two files *aliases.dir* and *aliases.pag* (both 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 auto-rebuild 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.

5. THE WHOLE SCOOP ON 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, rather than easy for a human to read or write.

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

5.1 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 sharp symbol ('#') are comments.

5.1.1 R and S - rewriting rules

The core of address parsing are the rewriting rules. These are an ordered production system. *Sendmail* 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.

5.1.2 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 upper case letters only. Lower case 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*.

5.1.3 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 upper case letters. Lower case 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*.

5.1.4 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 to this mailer

Only the first character of the field name is checked.

5.1.5 H - define header

The format of the header lines that sendmail inserts into the message are 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 that will be described below.

5.1.6 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.

5.1.7 T - define trusted users

Trusted users are those users who are permitted to override the sender address using the **-f** flag. These typically are "root," "uucp," and "network," but on some users 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.

5.1.8 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 less than 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
```

5.2 The Semantics

This section describes the semantics of the configuration file.

5.2.1 Special macros, conditionals

Macros are interpolated using the construct **$***x*, where *x* is the name of the macro to be interpolated. In particular, lower case letters are reserved to have special semantics, used to pass information in or out of sendmail, and some special characters are reserved to provide conditionals, etc.

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 which will be considered tokens and which will separate tokens when doing 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@Berkeley (Eric Allman)
Eric Allman <eric@Berkeley>
```

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 **$f** macro is the id of the sender as originally determined; when mailing to a specific host the **$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 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 **$z** macros get set to the host, user, and home directory (if local) of the recipient. The first two are set from the **$@** and **$:** part of the rewriting rules, respectively.

The **$p** and **$t** macros are used to create unique strings (e.g., 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 been proven 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," i.e., 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.

5.2.2 Special classes

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

5.2.3 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
```

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

- $1 UCBARPA
- $2 eric

5.2.4 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 may 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 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 may be omitted. The *mailer* and *host* must be a single word, but the *user* may be multi-part.

A RHS may 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 may precede a **$>** spec; for example:

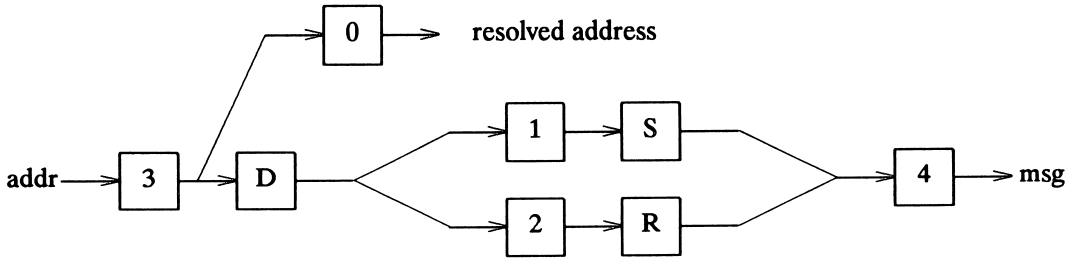
```
R$+     $:$>7$1
```

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.

5.2.5 Semantics of rewriting rule sets

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



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

Figure 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 *may* 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 going to actually 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.

5.2.6 Mailer flags etc.

There are a number of flags that may be associated with each mailer, each identified by a letter of the alphabet. Many of them are assigned semantics internally. These are detailed in section 8. Any other flags may be used freely to conditionally assign headers to messages destined for particular mailers.

5.2.7 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.

5.3 Building a Configuration File From Scratch

Building a configuration table 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 it is that you are trying to do and come up with a philosophy for the configuration table. This section is intended to explain what the real purpose of a configuration table is and to give you some ideas for what your philosophy might be.

5.3.1 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, etc. Since these are described in other places, 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. *Sendmail* does this in three subphases. Rulesets one and two are applied to all sender and

recipient addresses respectively. After this, you may 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 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. The mailer descriptor describes the interface requirements of the mailer.

5.3.2 Philosophy

The particular philosophy you choose will depend heavily on the size and structure of your organization. I will present a few possible philosophies here.

One general point applies to all of these philosophies: it is almost always a mistake to try to do full name resolution. For example, if you are trying to get names of the form "user@host" to the Arpanet, it does not pay to route them to "xyzvax!decvax!ucbvax!c70:user@host" since you then depend on several links not under your control. The best approach to this problem is to simply forward to "xyzvax!user@host" and let xyzvax worry about it from there. In summary, just get the message closer to the destination, rather than determining the full path.

5.3.2.1 Large site, many hosts - minimum information

Berkeley is an example of a large site, i.e., more than two or three hosts and multiple mail connections. We have decided that the only reasonable philosophy in our environment is to designate one host as the guru for our site. It must be able to resolve any piece of mail it receives. The other sites should have the minimum amount of information they can get away with. In addition, any information they do have should be hints rather than solid information.

For example, a typical site on our local ether network is "monet." When monet receives mail for delivery, it checks whether it knows that the destination host is directly reachable; if so, mail is sent to that host. If it receives mail for any unknown host, it just passes it directly to "ucbvax," our master host. Ucbvax may determine that the host name is illegal and reject the message, or may be able to do delivery. However, it is important to note that when a new mail connection is added, the only host that *must* have its tables updated is ucbvax; the others *may* be updated if convenient, but this is not critical.

This picture is slightly muddied due to network connections that are not actually located on ucbvax. For example, some UUCP connections are currently on "ucbarpa." However, monet *does not* know about this; the information is hidden totally between ucbvax and ucbarpa. Mail going from monet to a UUCP host is transferred via the ethernet from monet to ucbvax, then via the ethernet from ucbvax to ucbarpa, and then is submitted to UUCP. Although this involves some extra hops, we feel this is an acceptable tradeoff.

An interesting point is that it would be possible to update monet to send appropriate UUCP mail directly to ucbarpa if the load got too high; if monet failed to note a host as connected to ucbarpa it would go via ucbvax as before, and if monet incorrectly sent a message to ucbarpa it would still be sent by ucbarpa to ucbvax as before. The only problem that can occur is loops, for example, if ucbarpa thought that ucbvax had the UUCP connection and vice versa. For this reason, updates should *always* happen to the master host first.

This philosophy results as much from the need to have a single source for the configuration files (typically built using *m4* (1) or some similar tool) as any logical need. Maintaining more than three separate tables by hand is essentially an impossible job.

5.3.2.2 Small site - complete information

A small site (two or three hosts and few external connections) may find it more reasonable to have complete information at each host. This would require that each host know exactly where each network connection is, possibly including the names of each host on that network. As long as the site remains small and the the configuration remains relatively static, the update problem will probably not be too great.

5.3.2.3 Single host

This is in some sense the trivial case. The only major issue is trying to ensure that you don't have to know too much about your environment. For example, if you have a UUCP connection you might find it useful to know about the names of hosts connected directly to you, but this is really not necessary since this may be determined from the syntax.

5.3.3 Relevant issues

The canonical form you use should almost certainly be as specified in the Arpanet protocols RFC819 and RFC822.

RFC822 describes the format of the mail message itself. *Sendmail* 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 which 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 Berkeley one legal host might be "a.CC.Berkeley.EDU ;" reading from right to left, "EDU" is a top level domain comprising educational institutions, "Berkeley" 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.

Beware when reading RFC819 that there are a number of errors in it.

5.3.4 How to proceed

Once you have decided on a philosophy, it is worth examining the available configuration tables to decide if any of them are close enough to steal major parts of. Even under the worst of conditions, there is a fair amount of boiler plate 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, since anything you do will reflect through to the message. In particular, stripping of local domains is best deferred, since this can leave you with addresses with no domain spec at all. Since *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 Berkeley 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.

5.3.5 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 apply the set to. Test mode shows you the steps it takes as it proceeds, finally showing you the address it ends up with. 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 is probably going to print out several pages worth of information.

5.3.6 Building mailer descriptions

To add an outgoing mailer to your mail system, you will have to 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 $g" in the argv template.[4] If the mailer must be called as **root** the "S" flag should be given; this will not reset the userid before calling the mailer.[5] If this mailer is local (i.e., will perform final delivery rather than another network hop) the "l" flag should be given. Quote characters (backslashes and " marks) can be stripped from addresses if the "s" flag is specified; if this is not given they are passed through. If the mailer is capable of sending to more than one user on the same host in a single transaction the "m" flag should be stated. If this flag is on, then the argv template containing **$u** will be repeated for each unique user on a given host. The "e" flag will mark the mailer as being

4. The System V *mail*(1) command does not have such an option. A modified version of *mail* is included with this release which contains a "-x" option to allow the specification of the name of the sender.

5. *Sendmail* must be running setuid to root for this to work.

"expensive," which will cause *sendmail* to defer connection until a queue run.[6]

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 (i.e., 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, mckusick
```

will be modified to:

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

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

Other flags are described in section 8.

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
```

specifies a mailer to do local delivery and a mailer for ethernet delivery. The first is called "local," is located in the file "/bin/mail," takes a picky **-r** flag, 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

6. The "c" configuration option must be given for this to be effective.

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.

6. 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 if *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 Don't do aliasing or forwarding.

-t Read the header for "To:," "Cc:," and "Bcc:" lines, and send to everyone listed 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*. 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, don't 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, a 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. *Sendmail* 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 may be specified as primitive flags (provided for compatibility with *delivermail*). These are the e, i, m, and v options. Also, the f option may be specified as the -s flag.

7. CONFIGURATION OPTIONS

The following options may 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, don't 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 in to *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" by default. The 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, i.e., 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 to just 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 10000.

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

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

s Be super-safe when running things, i.e., always instantiate the queue file, even if you are going to attempt immediate delivery. *Sendmail* always instantiates the queue file before returning control the the client under any circumstances.

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*. Mailers 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 (i.e., don't 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, i.e., 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.

8. MAILER FLAGS

The following flags may be set in the mailer description.

f The mailer wants a **-f***from* flag, but only if this is a network forward operation (i.e., 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 Don't 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 (e.g, 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 (i.e., final delivery will be performed).

s Strip quote characters off of 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 for this mailer.

h 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 (i.e., 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 as:

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

automatically.

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

9. 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* (3) 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

Chapter 4

PAGE

Chapter 4

NAME SERVER OPERATIONS GUIDE FOR BIND †

1. Introduction

The Berkeley Internet Name Domain (BIND) Server implements the DARPA Internet name server for the UNIX operating system. A name server is a network service that enables clients to name resources or objects and share this information with other objects in the network. This in effect is a distributed data base system for objects in a computer network. BIND is fully integrated into network programs for use in storing and retrieving host names and address. The system administrator can configure the system to use BIND as a replacement to the original host table lookup of information in the network hosts file */etc/hosts*. The default configuration for System V/386 Streams TCP uses BIND.

1.1 The Name Service

The basic function of the name server is to provide information about network objects by answering queries. The specifications for this name server are defined in RFC974, RFC1034, and RFC1035. These documents are found in the *DDN Protocol Handbook, Volume II*, which is available from the *Network Information Center* at:

SRI International
333 Ravenswood Avenue, Room EJ291
Menlo Park, CA 94025

They can also be *ftp*'d from sri-nic.arpa (10.0.0.51). It is also recommended that you read the related manual pages, *named* (1M), *resolver* (3), and *resolver* (4).

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 fine for small networks where there are only a few machines and the different organizations responsible for them cooperate. But this does not work well for large networks where machines cross organizational boundaries.

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

† This chapter is based on the document of the same name, written by Kevin J. Dunlap of Digital Equipment Corporation, and Michael J. Karels of the University of California at Berkeley.

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

2. Types of Servers

There are several types of servers, Master, Caching, Remote and Slave.

2.1 Master Servers

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

2.1.1 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.

2.1.2 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.

2.2 Caching Only Server

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

2.3 Remote Server

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

2.4 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 nameservers 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 give 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 nameservers to resolv 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 Slave mode and forwarding is discussed further under the description of the named bootfile commands.

3. 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:

3.1 DARPA Internet

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

bind-request@ucbarpa.Berkeley.EDU.

3.2 CSNET

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

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

3.3 BITNET

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

4. Files

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

4.1 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 specifying the location on the command line when *named* is started up.

4.1.1 Domain

A default domain may be specified for the nameserver using a line such as for the server looks as follows:

domain Berkeley.Edu

The name server uses this information when it receives a query for a name without a "." that is not known. 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.

4.1.2 Directory

The directory line specifies the directory in which the nameserver should run, allowing the other file names in the boot file to use relative path names.

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 path names with $Include and to allow *named* to run in a location that is reasonable to dump core if it feels the urge.

4.1.3 Primary Master

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

primary Berkeley.Edu 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.

4.1.4 Secondary Master

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

secondary Berkeley.Edu 128.32.0.10 128.32.0.4 ucbhosts.bak

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 will be 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.

4.1.5 Caching Only Server

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

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

cache . root.cache

The period specifies the current domain. All cache files listed will be read in at named boot time and any values still valid will be reinstated in the cache and the root nameserver information in the cache files will always be used. For information on cache file see section on *Cache Initialization*.

4.1.6 Forwarders

Any server can make use of *forwarders*. A *forwarder* is another server capable of processing recursive queries that is willing to try resolving 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 prevent from sending any IP packets into the rest of the network and therefore must rely on a forwarder which does have access to the full net. The second reason is that the forwarder sees a union of all queries as they pass through his server and therefore he builds up a very rich cache of data compared to the cache in a typical workstation nameserver. 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.

4.1.7 Slave Mode

Slave mode is used if the use of forwarders is the only possible way to resolve queries due to lack of full net access or if you wish to prevent the nameserver 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 will forward each query to each of the the forwarders until an answer is found or the list of forwarders is exhausted.

4.1.8 Remote Server

To set up a host that will use a remote server instead of a local server to answer queries, the file */etc/resolv.conf* needs to be created. 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* () or *gethostbyaddr* () is called.

4.2 Cache Initialization

4.2.1 root.cache

The name server needs to know the servers that are the authoritative name servers for the root domain of the network. To do this we have to prime the name server's cache with the address of these higher authorities. The location of this file is specified in the boot file. This file uses the Standard Resource Record Format (aka. Masterfile Format) covered further on in this paper.

4.3 Domain Data Files

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

4.3.1 named.local

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

4.3.2 hosts

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

4.3.3 hosts.rev

This file specifies the IN-ADDR.ARPA domain. This is a special domain for allowing address to name mapping. As internet host addresses do not fall within domain boundaries, this special domain was formed to allow inverse mapping. The IN-ADDR.ARPA domain has four labels preceding it. These labels correspond to the 4 octets of an Internet address. All four octets must be specified even if an octets 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.

4.4 Standard Resource Record Format

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

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

Resource records have a standard format shown above. The first field is always the name of the domain record and it must always start in column 1. For some RR's the name may be left blank; in that case it takes on the name of the previous RR. The second field is an optional time to live field. This specifies how long this data will be stored in the data base. By leaving this field blank the default time to live is specified in the *Start Of Authority* resource record (see below). 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 RR. Case is preserved in names and data fields when loaded into the name server. All

comparisons and lookups in the name server data base are case insensitive.

The following characters have special meanings:

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

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

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

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

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

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

; Semicolon starts a comment; the remainder of the line is ignored.

* An asterisk signifies wildcarding.

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

4.4.1 $INCLUDE

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

$INCLUDE /usr/named/data/mailboxes

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

4.4.2 $ORIGIN

The origin is a way of changing the origin in a data file. The line starts in column 1, and is followed by a domain origin. This is useful for putting more than one domain in a data file. For example /etc/named.hosts might contain lines of the form

$ORIGIN CC.Berkeley.EDU
Domain data
$ORIGIN EE.Berkeley.EDU
Domain data

4.4.3 SOA - Start Of Authority

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

4.4.4 NS - Name Server

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

The *Name Server* record, *NS*, lists a name server responsible for a given domain. The first name field lists the domain that is serviced by the listed name server. There should be one *NS* record for each Primary Master server for the domain.

4.4.5 A - Address

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

The *Address* record, *A*, lists the address for a given machine. The name field is the machine name and the address is the network address. There should be one *A* record for each address of the machine.

4.4.6 HINFO - Host Information

{*name*}	{*ttl*}	*addr-class*	*HINFO*	*Hardware*	*OS*
		IN	HINFO	VAX-11/780	UNIX

Host Information resource record, *HINFO*, is for host specific data. This lists the hardware and operating system that are running at the listed host. It should be noted that only a single space separates the hardware info and the operating system info. If you want to include a space in the machine name you must quote the name.

4.4.7 WKS - Well Known Services

{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 nameserver)

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

4.4.8 CNAME - Canonical Name

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

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 (e.g. NS or MX) should list the canonical name, not the alias.

4.4.9 PTR - Domain Name Pointer

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

A *Domain Name Pointer* record, *PTR*, allows special names to point to some other location in the domain. The above example of a *PTR* record is used in setting up reverse pointers for the special *IN-ADDR.ARPA* domain. This line is from the example *hosts.rev* file. 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.

4.4.10 MB - Mailbox

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

MB is the *Mailbox* record. This lists the machine where a user wants to receive mail. The name field is the users login; the machine field denotes the machine to which mail is to be delivered. Mail Box names should be unique to the zone.

4.4.11 MR - Mail Rename Name

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

Mail Rename, 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.

4.4.12 MINFO - Mailbox Information

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

Mail Information record, *MINFO*, creates a mail group for a mailing list. This resource record is usually associated with a mail group *Mail Group*, but may 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.

4.4.13 MG - Mail Group Member

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

Mail Group, MG lists members of a mail group.

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

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

4.4.14 MX - Mail Exchanger

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 specify 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 can not deliver mail directly to Munnari. These two machines may have a private connection or use a different transport medium. The preference value is the order that a mailer should follow when there is more then one way to deliver mail to a single machine. See RFC974 for more detailed information.

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

4.5 Sample Files

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

4.5.0.1 Caching Only Server

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

type        domain                  source file or host

directory   /usr/local/lib/named
cache       .                       root.cache
primary     0.0.127.in-addr.arpa    /etc/named.local
```

4.5.1 Boot File

4.5.1.1 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
```

4.5.1.2 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
```

4.5.2 Remote Server

4.5.2.1 /etc/resolv.conf

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

4.5.3 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
A.ISI.EDU.              99999999    IN    A     26.3.0.103
BRL-AOS.ARPA.           99999999    IN    A     128.20.1.2
BRL-AOS.ARPA.           99999999    IN    A     192.5.25.82
BRL-AOS.ARPA.           99999999    IN    A     192.5.22.82
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
```

4.5.4 named.local

```
@    IN    SOA    ucbvax.Berkeley.Edu. kjd.ucbvax.Berkeley.Edu. (
                  1       ; Serial
                  3600    ; Refresh
                  300     ; Retry
                  3600000 ; Expire
                  3600 )  ; Minimum
     IN    NS     ucbvax.Berkeley.Edu.
1    IN    PTR    localhost.
```

4.5.5 Hosts

```
;
;  @(#)ucb-hosts  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.
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
                              nameserver )
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.
```

4.5.6 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.
```

5. Domain Management

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

5.1 Starting the Name Server

The hostname should be set to the full domain style name, (*i.e.* monet.Berkeley.EDU.) using *hostname* (*1*). The Name Server will be started automatically if the configuration file */etc/named.boot* is present. **Do Not** attempt to run *named* from *inetd*. This will continuously restart the name server and defeat the purpose of having a cache.

5.2 /etc/named.pid

When *named* is successfully started up it writes its process id into the file */etc/named.pid*. This is useful to programs that want to send signals to *named*.

5.3 /etc/hosts

The *gethostbyname* () library call can detect if *named* is running. If it is determined that *named* is not running it will look in */etc/hosts*(4) to resolve an address. This option was added to allow *ifconfig* (*1M*) to configure the machines local interfaces and to enable a system manager to access the network while the system is in single user mode. It is advisable to put the local machines interface addresses and a couple of machine names and address in */etc/hosts* so the system manager can rcp files from another machine when the system is in single user mode. The format of */etc/hosts* has not changed. See *hosts* (*4*) for more information. Since the process of reading */etc/hosts* is slow, it is not advised to use this option when the system is in multi user mode.

5.4 Signals

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

5.4.1 Reload

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

5.4.2 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 data base and cache to **/usr/tmp/named_dump.db** This should give you an indication to whether the data base was loaded correctly. The name of the dump file may be changed by defining *DUMPFILE* to the new name when compiling **named**.

Note: the following two signals only work when **named** is built with *DEBUG* defined.

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

SIGUSR2 - Turns off debugging completely.

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

REFERENCES

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

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

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

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

[RFC1032] Stahl, M., "Domain Administrators Guide" *Internet request for Comment 1032* Network Information Center, SRI International, Menlo Park, California. November 1987.

[RFC1033] Lottor, M., "Domain Administrators Operations Guide" *Internet request for Comment 1033* Network Information Center, SRI International, Menlo Park, California. November 1987.

[RFC1034] Mockapetris, P., "Domain Names - Concept and Facilities." *Internet Request For Comment 1034* Network Information Center, SRI International, Menlo Park, California. November 1983.

[RFC1035] Mockapetris, P., "Domain Names - Implementation and Specification." *Internet Request For Comment 1035* Network Information Center, SRI International, Menlo Park, California. November 1983.

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

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

Chapter 5

TIMED

INSTALLATION AND OPERATION GUIDE

Chapter 5

PAGE

Chapter 5

TIMED INSTALLATION AND OPERATION GUIDE†

1. Introduction

The clock synchronization service for *System V STREAMS TCP/IP* is composed of a collection of time daemons (*timed*) running on the machines in a local area network. The algorithms implemented by the service is 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 and described in detail in the TCP Programmer's Guide.

A time daemon has a twofold 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 next paragraphs are a brief overview of how the time daemon works. This document is mainly concerned with the administrative and technical issues of running *timed* at a particular site.

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. The master computes the *network time* as the average of the times provided by nonfaulty clocks.[1] It then sends to each *slave time daemon* the correction that should be performed on the clock of its machine. This process is repeated periodically. Since the correction is expressed as a time difference rather than an absolute time, transmission delays do not interfere with the accuracy of the synchronization. When a machine comes up and joins the network, it starts a slave time daemon which will ask the master for the correct time and will reset the machine's clock before any user activity can begin. The time daemons are 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 run-time error), or the network be partitioned. Under our 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, since the failure of the master results only in a gradual divergence of clock values, 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

---

† This chapter is based on the document of the same name, written by Riccardo Gusella, Stefano Zatti, and James M. Bloom of the University of California at Berkeley, along with Kirk Smith of Purdue University.

1. 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.

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 which 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.

2. Guidelines

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

If we exclude submasters, there will normally be, at most, one master time daemon in a local area internetwork. During an election, only one of the slave time daemons will become the new master. However, because of the characteristics of its machine, a slave can be prevented from becoming the master. Therefore, a subset of machines must be designated as potential master time daemons. A master time daemon will require CPU resources proportional to the number of slaves, in general, more than a slave time daemon, 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 *-n network* and *-i network* options respectively at start-up time. Typically, the time daemon would be instructed to ignore all but the networks belonging to the local administrative control.

In addition, there is a *pathological situation* to be avoided at all costs, that might occur when time daemons run on multiply-connected local area networks. In this case, as we have seen, time daemons running on gateway machines will be submasters and they will 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 is a slave on network Y and the master on network X. This *loop* of master time daemons will not function properly or guarantee a unique time on both networks, and will cause 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.

3. Options

The *flags* are:

-n network	to consider the named network.
-i network	to ignore the named network.
-t	to place tracing information in */usr/adm/timed.log*.
-M	to allow this time daemon to become a master. A time daemon run without this option will be forced in the state of slave during an election.

4. Daily Operation

Timedc(1M) is used to control the operation of the time daemon. It may be used to:

- 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 page on *timed* (1M) and *timedc* (1M)
for more detailed information.

The *rdate*(1M) command can be used to set the network date.

Chapter 6

GLOSSARY

alias	An alias is an alternate host name, which can be created as a convenience in addressing a host on a local network whose unique primary name is long and/or complicated.
ARP	Address Resolution Protocol is used by Ethernet for address mapping.
ARPA	Now called DARPA, stands for Defense Advanced Research Projects Agency. ARPANET is the network based on the work sponsored by this agency. See also **DDN**
block	A block (noun) is usually 1024 bytes.
broadcast network	A broadcast network is one that "broadcasts" all transmissions instead of from point to point. Each node then "grabs" the transmissions intended for them. For example, Ethernet broadcasts down its bus.
BSD	Berkeley Software Distribution
bus	A set of parallel signals implemented in hardware in a standard manner so that multiple devices can access it and communicate over it.
connection	A connection is a logical communication path identified by a pair of sockets.
DARPA	Department of Defense Advanced Research Project Agency, formerly called ARPA. This agency sponsored the network architecture research project upon which ARPANET is based. ARPANET is a large governmental internetwork, called the Internet, part of which is the Defense Data Network (DDN). See also **DDN** and **Internet.**
data link level	Data link level is the communications protocol for the physical media-link used to transport the data.
datagram	A datagram is a message sent in a packet switched computer communications network. The message made up of source and destination addresses and the data itself. The datagram model implies that no connection, such as a virtual circuit is needed, to send them and that they are not required to be delivered reliably or in sequence. See also **packet.**
DDN	Defense Data Network. The Defense Data Network (DDN) is a set of communications capabilities which links together computer systems within the Department of Defense (DoD). The DDN allows users of these computer systems to send mail and files between systems and to access other computers on the network in interactive terminal sessions. The DDN is part of the DARPA Internet. See **Internet**
daemon	A daemon is a UNIX system service. It is a program that is active in the background but not connected to a terminal.
destination	The destination address, an internet header field.
destination address	The destination address, usually the network and host identifiers.
flags	Flags is an internet header field carrying various control flags.
flow control	Flow control is the function and process of regulating the traffic and amount of data between flowing nodes so that neither node is sent more data than it can handle at a given time
fragment	A fragment is an IP packet that is one part of a UDP or TCP message. Messages may be split up, or fragmented, into fragments by IP if the message length exceeds the capability of the data link.

gateway	A software service installed at a switching node that connects two or more networks, especially if they use different protocols. A gateway provides UNIX internetworking with an extended logical network by transparently attaching one or more physical networks.
header	A header is the control information at the beginning of a message, segment, datagram, fragment, packet or block of data.
host	A host is a computer. In particular a source or destination of messages from the point of view of the communication network.
ICMP	Internet Control Message Protocol. ICMP is used by a gateway or destination host to communicate with a source host, for example, to report an error in datagram processing. ICMP, uses the basic support of IP as if ICMP were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.
Identification	Identification is an Internet Protocol field. This identifying value assigned by the sender aids in assembling the fragments of a datagram.
install	Install means to move the executable files from the distribution media to the system disk. In context, install can also mean to perform all the steps necessary to make a server or protocol operative
Internet	The Internet (spelled with initial capitalization) is the DARPA Internet System. See **DARPA**
Internet Address	Address is a source or destination address specific to the host level. It consists of a four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.
Internet datagram	An internet datagram is the unit of data exchanged between a pair of internet modules (includes the internet header).
internet module	An internet module is an instance or individual implementation of the Internet Protocol, residing at a local host or gateway.
Internet Protocol	Internet Protocol (IP) is the network level protocol used by UNIX internetworking
internetwork	An internetwork is a supernetwork made up of two or more networks able to communicate with each other through gateways. See **gateway**
IP	See **Internet Protocol**
layer	A layer is a network function or set of related network functions that forms an autonomous functional block in the superset of network architectural functions. This method of partitioning the necessary network functions allows each layer to interface transparently to adjoining layers and thereby provides a method of making network components more manageable.
Local Address	The Local Address the address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.
local packet	A local packet is the unit of transmission within a local network.
machine	A machine is a host computer. The use of this term is similar to "host," and "node," but "machine" connotes the machine-specific or hardware aspects of the host computer, whereas "node" connotes the logical aspects of a network host. Host connotes the relationship of the local node machine to application systems and remote hosts.
module	A module is an implementation, usually in software, of a protocol or other procedure

network	A network is a collection of computer nodes able to communicate with each other
network interface	A network interface is the hardware and driver software that connects a host to a physical network.
octet	An octet is an eight bit byte.
OSI	(Open Systems Interconnection). OSI is a standard of the ISO. This standard attempts to provide for consistent hardware and software interfaces among network products. OSI and other standard setters such as IBM and the National Bureau of Standards generally divide network architecture into seven layers: physical, link, network, transport, session, presentation, and application.
packet	A packet is a package of data with a header which may or may not be logically complete. More often a physical packaging than a logical packaging of data.
point-to-point	Point-to-point is a network configuration in which two points are connected to each other by a dedicated line, which can be a direct cable connection, a leased line, or a dialup to a service providing dedicated lines
port	A port is the portion of a socket that specifies which logical input or output channel of a process is associated with the data.
process	A process is a program in execution. A source or destination of data from the point of view of the TCP or other host-to-host protocol.
protocol	A protocol, in general, is a set of rules that enable a network entity to understand a communicating entity; however, the entity that employs these rules, such as the transport level protocol, TCP, is commonly referred to as a protocol. Therefore, a protocol is a software entity that implements a specific layer or function in a network architecture. In using the programmatic interface, a protocol is the next higher level protocol identifier, an internet header field.
RFC	Request For Comment. These refer to the "official" specification of Internet protocols and addressing mechanisms which are published by the Network Information Center of SRI in Palo Alto, CA and take the form of requests for comment.
root	Root is the login name of the super user. The super user is the user who has the widest form of machine privileges.
routing	Dynamic, or adaptive, routing is the ability to transfer data automatically to the destination node via alternative paths consisting of one or intermediate nodes. Routing includes the ability to ascertain available paths and to decide the best path, taking into account topology changes or node failures as they occur
server	A server is a system service, called a demon. It is usually a user program that runs in background, in user space, to provide a defined set of functions to the user who uses it through the command interface. Each time a user invokes it, the server provides a separate process for that user
socket	A socket is a file descriptor made up of system of data structures and pointers used by the kernel to identify and keep track of a process. It is an address which specifically includes a port identifier, that is, the concatenation of an Internet Address with a TCP port. Sockets are transparent to the user. Programs must open sockets to access network functions. Any one process cannot have more than 20 open files at a given time.

superuser	See **root**
TCP	Transmission Control Protocol is a transport level, connection-oriented protocol that provides reliable end-to-end message transmission over an internetwork
tuple	A tuple is a mathematical term for set of numbers composed of two or more factors. For example: [(XY)(AB)].
UDP	User datagram protocol is an unreliable user level transport protocol for transaction-oriented applications. It handles datagram sockets. It uses the IP for network services.
user	The user of the internet protocol. This may be a higher level protocol module, an application program, or a gateway program.
X.25	X.25 is a circuit-switched network protocol used commonly in Europe and less so in the United States. X.25 is based on a three-layer, peer-communications protocol standard defined by the International Telegraph and Telephone Consultative Committee (CCITT).

Chapter 7

ADMINISTRATOR'S REFERENCE

NAME

intro - introduction to network maintenance and operation commands

DESCRIPTION

This section contains information related to network operation and maintenance. It describes commands used to bring up the transport, **slink**; configure network interfaces, **ifconfig, slattach**; test status of remote hosts, **ping**; display packet tracing information, **trpt**; invoke network services, (those commands ending in '**d**') ; and other network administration functions.

NAME

arp - address resolution display and control

SYNOPSIS

arp hostname
arp -a [namelist] [corefile]
arp -d hostname
arp -s hostname ether_addr [**temp**] [**pub**] [**trail**]
arp -f filename

DESCRIPTION

The *arp* program displays and modifies the Internet-to-Ethernet address translation table, which is normally maintained by the address resolution protocol (*arp*(7)).

When *hostname* is the only argument, *arp* displays the current ARP entry for *hostname*. The host may be specified by name or by number, using Internet dot notation [see *hosts*(4) and *inet*(7)].

Options are interpreted as follows:

-a [*namelist*] [*corefile*]
Display all of the current ARP entries by reading the table from the file *corefile* (default **/dev/kmem**) based on the kernel file *namelist* (default **/unix**).

-d Delete an entry for the host whose name is *hostname*. (This can be performed only by the superuser.)

-s *hostname ether_addr* [**temp**] [**pub**] [**trail**]
Create an ARP entry for the host whose name is *hostname* with the Ethernet address *ether_addr*. The Ethernet address is given as six colon-separated, two-digit hexadecimal numbers. The entry will be permanent unless the argument **temp** is specified on the command line. If **pub** is specified, the entry will be "published": that is, this system will act as an ARP server, responding to requests for *hostname* even though the host address is not an address of the local host. If **trail** is specified, trailer encapsulations are to be used with this host. *N.B.* Trailers are a link-dependent issue. Currently, no known LLI-compliant ethernet driver supports trailers, and it is unwise to advertise them, unless it is certain that the link layer can handle them.

-f *filename*
Read the file *filename* and set multiple entries in the ARP tables. Entries in the file should be of the form

hostname ether_addr [**temp**] [**pub**] [**trail**]

with argument meanings as given above.

SEE ALSO

inet(3), arp(7), ifconfig(1M).

NAME

fingerd - remote user information server

SYNOPSIS

/etc/fingerd

DESCRIPTION

Fingerd is a server that provides a network interface to the *finger*(1) program (or, on some other systems, the *name* program). This interface allows *finger* to display information about remote users.

Fingerd listens for TCP connections on the *finger* port (see *services*(4)). For each connection, *fingerd* reads a single input line (terminated by a <CRLF>), passes the line to *finger*, and copies the output of *finger* to the user on the client machine.

Fingerd is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

SEE ALSO

finger(1), inetd(1M), inetd.conf(4), services(4).

WARNING

Connecting to *fingerd* using TELNET (see *telnet*(1)) can have unpredictable consequences and is not recommended.

NAME

ftpd - DARPA Internet File Transfer Protocol server

SYNOPSIS

/etc/ftpd [-d] [-l] [-ttimeout]

DESCRIPTION

Ftpd is the DARPA Internet File Transfer Protocol server process. The server uses the TCP protocol and listens at the port specified in the "ftp" service specification; see *services*(4).

Ftpd is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

If the **-d** option is specified, debugging information is written to the syslog.

If the **-l** option is specified, each FTP session is logged in the syslog.

The FTP server will timeout an inactive session after 15 minutes. If the **-t** option is specified, the inactivity timeout period will be set to *timeout*.

The FTP server currently supports the following FTP requests; case is not distinguished.

Request	Description
ABOR	abort previous command
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CDUP	change to parent of current working directory
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory ("ls -l")
MKD	make a directory
MODE	specify data transfer *mode*
NLST	give name list of files in directory ("ls")
NOOP	do nothing
PASS	specify password
PASV	prepare for server-to-server transfer
PORT	specify data connection port
PWD	print the current working directory
QUIT	terminate session
RETR	retrieve a file
RMD	remove a directory
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
STOU	store a file with a unique name
STRU	specify data transfer *structure*
SYST	display operating system information
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

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

The FTP server will abort an active file transfer only when the ABOR command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in Internet RFC 959.

Ftpd interprets file names according to the "globbing" conventions used by *sh*(1). This allows users to utilize the metacharacters "*?[]{}~".

Ftpd authenticates users according to three rules.

1) The user name must be in the password data base, */etc/passwd*, and not have a null password. In this case a password must be provided by the client before any file operations may be performed.

2) The user name must not appear in the file */etc/ftpusers*.

3) If the user name is "anonymous" or "ftp", an anonymous ftp account must be present in the password file (user "ftp"). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, *ftpd* takes special measures to restrict the client's access privileges. The server performs a *chroot*(2) command to the home directory of the "ftp" user. In order that system security is not breached, it is recommended that the "ftp" subtree be constructed with care; the following rules are recommended. (*N.B.* ~ftp means "the home directory of user ftp")

~ftp) Make the home directory owned by "ftp" and unwritable by anyone.

~ftp/bin)
Make this directory owned by the super-user and unwritable by anyone. The program *ls*(1) must be present to support the list commands. This program should have mode 111.

~ftp/etc)
Make this directory owned by the super-user and unwritable by anyone. The files *passwd*(4) and *group*(4) must be present for the *ls* command to work properly. These files should be mode 444.

~ftp/pub)
Make this directory mode 777 and owned by "ftp." Users should then place files which are to be accessible via the anonymous account in this directory.

SEE ALSO

ftp(1), syslog(3)

BUGS

The anonymous account is inherently dangerous and should avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user id of the logged in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

FILES

/etc/ftpusers- restricted user list
/etc/passwd - the user database
/etc/group - the group database
/usr/adm/syslog- the system log file

The following files are needed for anonymous ftp:

~ftp/etc/passwd- used by ~ftp/bin/ls
~ftp/etc/group- used by ~ftp/bin/ls
~ftp/bin/ls - to support the LIST and NLST commands

In addition, if your /bin/ls is linked with shared libraries, you will need to copy /shlib/libc_s to ~ftp/shlib/libc_s. If your implementation is using the SIOCSOCKSYS ioctl, you will need to *mknod*(1M) ~ftp/dev/socksys.

NAME

ifconfig - configure network interface parameters

SYNOPNSIS

/etc/ifconfig interface address_family [address [dest_address]] [parameters]

/etc/ifconfig interface [protocol_family]

DESCRIPTION

Ifconfig is used to assign an address to a network interface and/or configure network interface parameters; it defines the network address of each interface present on a machine. *Ifconfig* is run at system start-up time via *tcp*(1M). *Ifconfig* may be run at other times to redefine an interface's address or other operating parameters (for example, *slattach*(1M) also runs *ifconfig*).

The *interface* parameter is a string of the form "name unit", e.g. "en0".

Since an interface may receive transmissions in differing protocols, each of which may require separate naming schemes, it is necessary to specify the *address_family*, which may change the interpretation of the remaining parameters. Currently only the Internet address family is supported: thus, the only valid value for *address_family* is **inet**.

For the DARPA-Internet family, the address is either a host name or a DARPA Internet address expressed in the Internet standard "dot notation". (Host name translation is performed either by the name server or by an entry in **/etc/hosts** [see *named*(1M) and *hosts*(4)]. Internet "dot notation" is described in *hosts*(4) and *inet*(7). Other address families may use different notations.)

The following parameters may be set with *ifconfig*:

up Mark an interface "up". This may be used to enable an interface after an "ifconfig down." It happens automatically when setting the first address on an interface. If the interface was reset when previously marked down, the hardware will be re-initialized.

down Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well. This action does not automatically disable routes using the interface.

detach Remove an interface from the system. This command is applicable to transient interfaces only, such as serial line interfaces.

trailers Request the use of a "trailer" link level encapsulation when sending (default). If a network interface supports *trailers*, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver. On networks that support the Address Resolution Protocol (see *arp*(7); currently, only 10 Mb/s Ethernet), this flag indicates that the system should request that other systems use trailers when sending to this host. Similarly, trailer encapsulations will be sent to other hosts that have made such requests. Currently used by Internet protocols only.

-trailers Disable the use of a "trailer" link level encapsulation.

arp Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses. This option is not applicable in the STREAMS environment. Use of arp for an interface is specified in **/etc/strcf**. The arp driver will be opened when the STREAMS stack is built.

-arp Disable the use of the Address Resolution Protocol.

metric *n* Set the routing metric of the interface to *n*, default 0. The routing metric is used by the routing protocol. Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.

debug Enable driver dependent debugging code; usually, this turns on extra console error logging.

-debug Disable driver dependent debugging code.

netmask *mask* (Inet only) Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation Internet address, or with a pseudo-network name listed in the network table *networks*(4). The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion.

dstaddr Specify the address of the correspondent on the other end of a point to point link.

broadcast (Inet only) Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 1's.

onepacket Enable the *one-packet* mode of operation (used for interfaces that can't handle back-to-back packets). The keyword **onepacket** must be followed by two numeric parameters, giving the small packet size and threshold, respectively. If small packet detection is not desired, these values should be zero. See *tcp*(7) for an explanation of one-packet mode.

-onepacket Disable one-packet mode.

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

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

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, or the user is not privileged and tried to alter an interface's configuration.

FILES

/etc/slattach
calls ifconfig to start serial lines

SEE ALSO

arp(1M), tcp(1M), netstat(1M), hosts(4), networks(4), strcf(4), arp(7), tcp(7)..

NAME

inetd - internet "super-server"

SYNOPSIS

/etc/inetd [**-d**] [configuration file]

DESCRIPTION

Inetd listens on multiple ports for incoming connection requests. When it receives a request, it spawns the appropriate server. The use of a "super-server" allows other servers to be spawned only when needed and to terminate when they have satisfied a particular request.

The mechanism is as follows: *inetd* is started by the super-user (usually during init 2, if **/etc/tcp** is linked to **/etc/rc2.d/Snntcp**.) To obtain information about the servers it needs to spawn, *inetd* reads its configuration file (by default, this is **/etc/inetd.conf**) and issues a call to *getservbyname* [see *getservent*(3)]. (Note that **/etc/services** and **/etc/protocols** must be properly configured.) *inetd* then creates a socket for each server and binds each socket to the port for that server. It does a *listen*(2) on all connection-based sockets (that is, stream rather than datagram), and waits, using *select*(2), for a connection or datagram.

- When a connection request is received on a listening (stream) socket, *inetd* does an *accept*(2), thereby creating a new socket. (*inetd* continues to listen on the original socket for new requests). *Inetd* forks, dups, and execs the appropriate server, passing it any server program arguments specified in *inetd*'s configuration file. The invoked server has I/O to *stdin*, *stdout*, and *stderr* done to the new socket; this connects the server to the client process. (Some "built-in," internal services are performed via function calls rather than child processes.)
- When there is data waiting on a datagram socket, *inetd* forks, dups, and execs the appropriate server, passing it any server program arguments; unlike a connection-based server, a datagram server has I/O to *stdin*, *stdout*, and *stderr* done to the original socket. If the datagram socket is marked as "wait" (this corresponds to an entry in *inetd*'s configuration file), the invoked server must process the message before *inetd* considers the socket available for new connections. If the datagram socket is marked as "nowait," *inetd* continues to process incoming messages on that port. *tftpd* is an exceptional case: although its entry in *inetd*'s configuration file must be "wait" (this is to avoid contention for the port), *inetd* is able to continue processing new messages on the port.

The following servers may be started by *inetd*: *fingerd*, *ftpd*, *rexecd*, *rlogind*, *rshd*, *talkd*, *telnetd*, and *tftpd*. *inet* must also start several internal services: these are described in *inetd.conf*(4). Do **not** arrange for *inetd* to start *named*, *routed*, *rwhod*, *sendmail, listen* (RFS listening server), or any NFS server.

Inetd rereads its configuration file when it receives a hangup signal, SIGHUP. Services may be added, deleted or modified when the configuration file is reread.

The **-d** option turns on socket-level debugging and prints debugging information to *stdout*.

FILES

/etc/inetd.conf
/etc/protocols
/etc/services

SEE ALSO

fingerd(1M), ftpd(1M), rexecd(1M), rlogind(1M), rshd(1M), talkd(1M), telnetd(1M), tftpd(1M), inetd.conf(4), protocols(4), services(4).

NAME

ldsocket - load socket configuration

SYNOPSIS

ldsocket [-v] [-c file]

DESCRIPTION

Ldsocket initializes the System V/386 Streams TCP Berkeley networking compatability interface, which is an alternate stream head supporting the *socket*(2) system call family. *Ldsocket* loads the kernel with associations between the protocol family, type and number triplets passed to the *socket* system call, and the STREAMS devices supporting those protocols. *Ldsocket* reads the file **/etc/sockcf** to obtain configuration information, and must be run before the Berkeley networking interface can be used.

The following options may be specified on the *ldsocket* command line:

-c *file* Use *file* instead of **/etc/sockcf**.

-v Verbose mode (a message is written to *stderr* for each protocol loaded)

FILES

/etc/sockcf

SEE ALSO

sockcf(4), intro(7), socket(2) in the TCP Programmer's Guide and Reference.

NAME

lmail - handle local mail delivery from sendmail

SYNOPSIS

lmail user ...

DESCRIPTION

Lmail interprets incoming mail received from *sendmail*(1M), and delivers it to the specified user on the local machine. It locks the user's mailbox using the *mail*(1) locking mechanism.

SEE ALSO

mail(1), sendmail(1M)

NAME

logger - make entries in the system log

SYNOPSIS

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

ARGUMENTS

-t *tag*	Mark every line in the log with the specified *tag*.
-p *pri*	Enter the message with the specified priority. The priority may be specified numerically or as a "facility.level" pair. For example, "-p local3.info" logs the message(s) as *info*rmational level in the *local3* facility. The default is "user.notice."
-i	Log the process id of the logger process with each line.
-f *file*	Log the specified file.
message	The message to log; if not specified, the **-f** file or standard input is logged.

DESCRIPTION

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

A message can be given on the command line, which is logged immediately, or a file is read and each line is logged.

EXAMPLES

logger System rebooted

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

SEE ALSO

syslog(3), syslogd(1M).

NAME

mkhosts - make node name commands

SYNOPSIS

/etc/mkhosts

DESCRIPTION

mkhosts makes the simplified forms of the *rcmd*(1) and *rlogin*(1) commands. For each node listed in **/etc/hosts**, *mkhosts* creates a link to **/usr/bin/rcmd** in **/usr/hosts**. Each link's name is the same as the node's official name in **/etc/hosts**.

SEE ALSO

rcmd (1), rlogin(1). Bell Technologies System V/386 TCPeroff -man intro.1m eroff -man arp.1m eroff -man fingerd.1m eroff -man ftpd.1m eroff -man ifconfig.1m eroff -man inetd.1m eroff -man ldsocket.1m eroff -man lmail.1m eroff -man logger.1m eroff -man mkhosts.1m eroff -man named.1m eroff -man netstat.1m eroff -man ping.1m eroff -man rdate.1m eroff -man rexecd.1m eroff -man rlogind.1m eroff -man rmail.1m eroff -man route.1m eroff -man routed.1m eroff -man rshd.1m eroff -man rwhod.1m eroff -man sendmail.1m eroff -man slattach.1m eroff -man slink.1m eroff -man talkd.1m eroff -man tcp.1m eroff -man telnetd.1m eroff -man tftpd.1m eroff -man timed.1m eroff -man timedc.1m eroff -man trace.1m eroff -man trpt.1m

NAME

named - Internet domain name server

SYNOPSIS

named [**-d** *debuglevel*] [**-p** *port#*] [{-b} *bootfile*]

DESCRIPTION

Named is the Internet domain name server. See RFC1035 for more information on the Internet name-domain system. Without any arguments, *named* will read the default boot file **/etc/named.boot**, read any initial data and listen for queries.

Options are:

-d Print debugging information. A number after the "d" determines the level of messages printed.

-p Use a different port number. The default is the standard port number as listed in **/etc/services.**

-b Use an alternate boot file. This is optional and allows you to specify a file with a leading dash.

Any additional argument is taken as the name of the boot file. The boot file contains information about where the name server is to get its initial data. If multiple boot files are specified, only the last is used. Lines in the boot file cannot be continued on subsequent lines. The following is a small example:

```
;
;       boot file for name server
;
directory       /usr/local/lib/named

; type      domain                  source host/file        backup file

cache       .                                               root.cache
primary     Berkeley.EDU            berkeley.edu.zone
primary     32.128.IN-ADDR.ARPA     ucbhosts.rev
secondary   CC.Berkeley.EDU         128.32.137.8 128.32.137.3 cc.zone.bak
secondary   6.32.128.IN-ADDR.ARPA   128.32.137.8 128.32.137.3 cc.rev.bak
primary     0.0.127.IN-ADDR.ARPA                            localhost.rev
forwarders  10.0.0.78 10.2.0.78
; slave
```

The "directory" line causes the server to change its working directory to the directory specified. This can be important for the correct processing of $INCLUDE files in primary zone files.

The "cache" line specifies that data in "root.cache" is to be placed in the backup cache. Its main use is to specify data such as locations of root domain servers. This cache is not used during normal operation, but is used as "hints" to find the current root servers. The file "root.cache" is in the same format as "berkeley.edu.zone". There can be more than one "cache" file specified. The cache files are processed in such a way as to preserve the time-to-live's of data dumped out. Data for the root nameservers is kept artificially valid if necessary.

The first "primary" line states that the file "berkeley.edu.zone" contains authoritative data for the "Berkeley.EDU" zone. The file "berkeley.edu.zone" contains data in the master file format described in RFC1035. All domain names are relative to the origin, in this case, "Berkeley.EDU" (see below for a more detailed description). The second "primary" line states that the file "ucbhosts.rev" contains authoritative data for the domain "32.128.IN-ADDR.ARPA," which is used to translate addresses in network 128.32 to hostnames. Each master file should begin with an SOA record for the zone (see below).

The first "secondary" line specifies that all authoritative data under "CC.Berkeley.EDU" is to be transferred from the name server at 128.32.137.8. If the transfer fails it will try 128.32.137.3 and continue trying the addresses, up to 10, listed on this line. The secondary copy is also authoritative for the specified domain. The first non-dotted-quad address on this line will be taken as a filename in which to backup the transferred zone. The name server will load the zone from this backup file if it exists when it boots, providing a complete copy even if the master servers are unreachable. Whenever a new copy of the domain is received by automatic zone transfer from one of the master servers, this file will be updated. The second "secondary" line states that the address-to-hostname mapping for the subnet 128.32.136 should be obtained from the same list of master servers as the previous zone.

The "forwarders" line specifies the addresses of sitewide servers that will accept recursive queries from other servers. If the boot file specifies one or more forwarders, then the server will send all queries for data not in the cache to the forwarders first. Each forwarder will be asked in turn until an answer is returned or the list is exhausted. If no answer is forthcoming from a forwarder, the server will continue as it would have without the forwarders line unless it is in "slave" mode. The forwarding facility is useful to cause a large sitewide cache to be generated on a master, and to reduce traffic over links to outside servers. It can also be used to allow servers to run that do not have access directly to the Internet, but wish to act as though they do.

The "slave" line (shown commented out) is used to put the server in slave mode. In this mode, the server will only make queries to forwarders. This option is normally used on machine that wish to run a server but for physical or administrative reasons cannot be given access to the Internet, but have access to a host that does have access.

The "sortlist" line can be used to indicate networks that are to be preferred over other, unlisted networks. Queries for host addresses from hosts on the same network as the server will receive responses with local network addresses listed first, then addresses on the sort list, then other addresses. This line is only acted on at initial startup. When reloading the nameserver with a SIGHUP, this line will be ignored.

The master file consists of control information and a list of resource records for objects in the zone of the forms:

```
$INCLUDE <filename> <opt_domain>
$ORIGIN <domain>
<domain> <opt_ttl> <opt_class> <type> <resource_record_data>
```

where *domain* is "." for root, "@" for the current origin, or a standard domain name. If *domain* is a standard domain name that does not end with ".", the current origin is appended to the domain. Domain names ending with "." are unmodified. The *opt_domain* field is used to define an origin for the data in an included file. It is equivalent to placing a $ORIGIN statement before the first line of the included file. The field is optional. Neither the *opt_domain* field nor $ORIGIN statements in the included file modify the current origin for this file. The *opt_ttl* field is an optional integer number for the time-to-live field. It defaults to zero, meaning the minimum value specified in the SOA record for the zone. The *opt_class* field is the object address type; currently only one type is supported, IN, for objects connected to the DARPA Internet. The *type* field contains one of the following tokens; the data expected in the *resource_record_data* field is in parentheses.

A	a host address (dotted quad)
NS	an authoritative name server (domain)
MX	a mail exchanger (domain)
CNAME	the canonical name for an alias (domain)
SOA	marks the start of a zone of authority (domain of originating host, domain address of maintainer, a serial number and the following parameters in seconds: refresh, retry, expire and minimum TTL (see RFC1035))

MB a mailbox domain name (domain)

MG a mail group member (domain)

MR a mail rename domain name (domain)

NULL a null resource record (no format or data)

WKS a well known service description (not implemented yet)

PTR a domain name pointer (domain)

HINFO host information (cpu_type OS_type)

MINFO mailbox or mail list information (request_domain error_domain)

Resource records normally end at the end of a line, but may be continued across lines between opening and closing parentheses. Comments are introduced by semicolons and continue to the end of the line.

Each master zone file should begin with an SOA record for the zone. An example SOA record is as follows:

```
@       IN      SOA     ucbvax.Berkeley.EDU.  rwh.ucbvax.Berkeley.EDU. (
                                        2.89    ; serial
                                        10800   ; refresh
                                        3600    ; retry
                                        3600000 ; expire
                                        86400 ) ; minimum
```

The SOA lists a serial number, which should be changed each time the master file is changed. Secondary servers check the serial number at intervals specified by the refresh time in seconds; if the serial number changes, a zone transfer will be done to load the new data. If a master server cannot be contacted when a refresh is due, the retry time specifies the interval at which refreshes should be attempted until successful. If a master server cannot be contacted within the interval given by the expire time, all data from the zone is discarded by secondary servers. The minimum value is the time-to-live used by records in the file with no explicit time-to-live value.

NOTES

The boot file directives "domain" and "suffixes" have been obsoleted by a more useful resolver based implementation of suffixing for partially qualified domain names. The prior mechanisms could fail under a number of situations, especially when then local nameserver did not have complete information.

The following signals have the specified effect when sent to the server process using the *kill*(1) command.

SIGHUP

Causes server to read named.boot and reload database.

SIGINT

Dumps current data base and cache to /usr/tmp/named_dump.db

SIGIOT

Dumps statistics data into /usr/tmp/named.stats if the server is compiled -DSTATS. Statistics data is appended to the file.

SIGSYS

Dumps the profiling data in /usr/tmp if the server is compiled with profiling (server forks, chdirs and exits).

SIGTERM

Dumps the primary and secondary database files. Used to save modified data on shutdown if the server is compiled with dynamic updating enabled.

SIGUSR1

Turns on debugging; each SIGUSR1 increments debug level.

SIGUSR2

Turns off debugging completely.

FILES

/etc/named.boot	name server configuration boot file
/etc/named.pid	the process id
/usr/tmp/named.run	debug output
/usr/tmp/named_dump.db	dump of the name server database
/usr/tmp/named.stats	nameserver statistics data

SEE ALSO

gethostent(3), signal(2), sigset(2), resolver(3), resolver(4) in the TCP Programmer's Reference, kill(1) in the UNIX User's Reference, hostname(5), and the chapter titled *Name Server Operations Guide for BIND* in this manual.

NAME

netstat - show network status

SYNOPSIS

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

DESCRIPTION

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

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

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

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

-m show network memory usage

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

-s show per-protocol statistics

-r show the routing tables

-S show serial line configuration

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

-I show interface state for *interface* only.

-p limit statistics and control block displays to *protocol-name, e.g.* **tcp.**

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

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

There are a number of display formats, depending on the information presented. The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and, optionally, the internal state of the protocol.

Address formats are of the form "host.port" or "network.port" if a socket's address specifies a network but no specific host address. When known, the host and network addresses are displayed symbolically according to the data bases **/etc/hosts** and **/etc/networks**, respectively. If a symbolic name for an address is unknown, or if the **-n** option is specified, the address is printed in the Internet "dot format"; refer to *rhosts*(4) for more information regarding this format. Unspecified, or "wildcard," addresses and ports appear as "*".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network address (currently Internet specific) of the interface and the maximum transmission unit ("mtu") are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The "flags" field shows the state of the route ("U" if "up"), and whether the route is to a gateway ("G"). Direct routes are created for each interface attached to the local host. The "refcnt" field gives the current number of active uses of the route. Connection-oriented protocols normally hold on to a single route for the duration of a connection, while connectionless protocols obtain a route then discard it. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column summarizing information for all interfaces and a column for the interface with the most traffic since the system was last rebooted. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

The serial line display shows the mapping of serial line units to serial devices. The baud rate and protocols in use are also shown.

SEE ALSO

slattach(1M), hosts(4), networks(4), protocols(4), services(4).

BUGS

Interface statistics are dependent on the link driver. If it does not attach itself to the *ifstats* structure in the kernel, the message "No Statistics Available" will be printed for that interface. sc arp.1m < sc.input sc fingerd.1m < sc.input sc ftpd.1m < sc.input sc ifconfig.1m < sc.input sc inetd.1m < sc.input sc intro.1m < sc.input sc ldsocket.1m < sc.input sc lmail.1m < sc.input sc mkhosts.1m < sc.input sc named.1m < sc.input sc ping.1m < sc.input sc portmap.1m < sc.input sc rdate.1m < sc.input sc rexecd.1m < sc.input sc rlogind.1m < sc.input sc rmail.1m < sc.input sc route.1m < sc.input sc routed.1m < sc.input sc rshd.1m < sc.input sc rwhod.1m < sc.input sc sendmail.1m < sc.input sc slattach.1m < sc.input sc slink.1m < sc.input sc slipd.1m < sc.input sc talkd.1m < sc.input sc tcp.1m < sc.input sc telnetd.1m < sc.input sc tftpd.1m < sc.input sc timed.1m < sc.input sc timedc.1m < sc.input sc trace.1m < sc.input sc trpt.1m < sc.input

NAME

ping - send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

/etc/ping [**-r**] [**-v**] host [packetsize] [count]

DESCRIPTION

ping is a troubleshooting tool for tracking a single-point hardware or software failure in the Internet. It uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams ("pings") have an IP and ICMP header, followed by a **struct timeval**, and then an arbitrary number of "pad" bytes used to fill out the packet. Default datagram length is 64 bytes, but this may be changed using the command-line option. Other options are:

-r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it.

-v Verbose output. ICMP packets other than ECHO RESPONSE that are received are listed.

When using *ping* for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be "pinged". *ping* sends one datagram per second, and prints one line of output for every ECHO_RESPONSE returned. No output is produced if there is no response. If an optional *count* is given, only that number of requests is sent. Round-trip times and packet loss statistics are computed. When all responses have been received or the program times out (with a *count* specified), or if the program is terminated with a SIGINT, a brief summary is displayed.

This program is intended for use in network testing, measurement and management. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is unwise to use *ping* during normal operations or from automated scripts.

SEE ALSO

netstat(1M), ifconfig(1M).

BUGS

The round-trip times are always reported to be 0ms.

NAME

rdate - notify time server that date has changed

SYNOPSIS

rdate

DESCRIPTION

Rdate notifies *timed(1M)* that the system date has changed. If the local time server is a master, it will notify all of the slaves that the time has changed. If it is a slave, it will request that the master update the time.

Rdate should be run whenever the super-user sets the date with *date(1)*. A shell script such as the following could be used to do both automatically.

```
:
#
# mv /bin/date /bin/s5date
# install as /bin/date
#
PATH=/bin:/usr/bin
s5date $*
rdate
```

SEE ALSO

date(1), gettimeofday(3), rdate(1M), timed(1M), timedc(1M). adjtime(2), icmp(7) in the TCP Programmer's Reference,
Timed Installation and Operation Guide chapter in this manual.

NAME

rexecd - remote execution server

SYNOPSIS

/etc/rexecd

DESCRIPTION

Rexecd is the server for the *rexec*(3) routine. The server provides remote execution facilities with authentication based on user names and passwords.

Rexecd listens for service requests at the port indicated in the "exec" service specification; see *services*(4). When a service request is received the following protocol is initiated:

1) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine.

3) A null terminated user name of at most 16 characters is retrieved on the initial socket.

4) A null terminated, unencrypted password of at most 16 characters is retrieved on the initial socket.

5) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

6) *Rexecd* then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.

7) A null byte is returned on the initial socket and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rexecd*.

rexecd is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf**.

DIAGNOSTICS

Except for the last one listed below, all diagnostic messages are returned on the initial socket, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

"username too long"
The name is longer than 16 characters.

"password too long"
The password is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"Password incorrect."
The wrong was password supplied.

"No remote directory."
The *chdir* command to the home directory failed.

"Try again."
A *fork* by the server failed.

"<shellname>: ..."
The user's login shell could not be started. This message is returned on the connection associated with the **stderr**, and is not preceded by a flag byte.

SEE ALSO

rexec(3), inetd(1M), inetd.conf(4), services(4)

BUGS

Indicating "Login incorrect" as opposed to "Password incorrect" is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data and password exchanges to be encrypted should be present.

NAME

rlogind - remote login server

SYNOPSIS

/etc/rlogind

DESCRIPTION

Rlogind is a network server which supports remote logins by programs such as *rlogin*(1). It is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

Rlogind enforces an authentication procedure based on equivalence of user names (see *rhosts*(4)). This procedure assumes all host on the network are equally secure.

SEE ALSO

inetd(1M), rlogin(1), inetd.conf(4), rhosts(4), services(4).

NAME

rmail - handle remote mail received via uucp

SYNOPSIS

rmail user ...

DESCRIPTION

Rmail interprets incoming mail received via *uucp*(1C), collapsing "From" lines in the form generated by *mail*(1) into a single line of the form "return-path!sender", and passing the processed mail on to *sendmail*(1M).

Rmail is explicitly designed for use with *uucp* and *sendmail*.

SEE ALSO

mail(1), uucp(1C), sendmail(1M)

NAME

route - manually manipulate the routing tables

SYNOPSIS

/etc/route [**-f**] [**-n**] [command destination gateway [metric]]

DESCRIPTION

route is a program used to manually manipulate the network routing tables. It is normally not needed, since the routing daemon, *routed* manages the system routing table and therefore handles this function.

route accepts two commands: *add*, to add a route; and *delete*, to delete a route.

All commands have the following syntax:

/etc/route *command destination gateway* [*metric*]

where *destination* is a host or network for which the route is "to", *gateway* is the gateway to which packets should be addressed, and *metric* is an optional count indicating the number of hops to the *destination*. If no metric is specified, *route* assumes a value of 0. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. If the *destination* has a "local address part" of INADDR_ANY, the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. *NOTE*: If the route is to a destination connected via a gateway, *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first in the host name database; see *hosts*(4). If this lookup fails, the name is then looked for in the network name database; see *networks*(4).

route uses a raw socket and the SIOCADDRT and SIOCDELRT *ioctl*'s to do its work. As such, only the super-user may modify the routing tables.

If the **-f** option is specified, *route* will "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

The **-n** option prevents attempts to print host and network names symbolically when reporting actions. *route*

DIAGNOSTICS

"add [host | network] *name*: **gateway** *host* **flags** *hex-flags*"
The specified route is being added to the tables. The values printed are from the routing table entry supplied in the *ioctl* call.

"delete *host*: **gateway** *host* **flags** *hex-flags*"
As above, but when deleting an entry.

"*host host* **done**"
When the **-f** flag is specified, each routing table entry deleted is indicated with a message of this form.

"not in table"
A delete operation was attempted for an entry which wasn't present in the tables.

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

SEE ALSO

routed(1M), intro(4), hosts(4), networks(4).

NAME

rshd - remote shell server

SYNOPSIS

/etc/rshd

DESCRIPTION

Rshd is the network server for programs such as *rcmd*(1) and *rcp*(1) which need to execute a noninteractive shell on remote machines. *Rshd* is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

rshd enforces an authentication procedure based on equivalence of user names (see *rhosts*(4)). This procedure assumes all nodes on the network are equally secure.

SEE ALSO

inetd(1M), rcmd(1), rcp(1), inetd.conf(4), rhosts(4).

NAME

routed - network routing daemon

SYNOPSIS

/etc/routed [-d] [-g] [-s] [-t] [logfile **]**

DESCRIPTION

routed manages the Internet routing tables using a variant of the Xerox NS Routing Information Protocol. *routed* is invoked by the super-user (usually during init 2).

In normal operation *routed* listens on the *udp*(7) socket for the *route* service (see *services*(4)) for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When *routed* is started, it uses the SIOCGIFCONF *ioctl* to find those directly connected interfaces configured into the system and marked "up" (the software loopback interface is ignored). If multiple interfaces are present, it is assumed that the host will forward packets between networks. *routed* then transmits a *request* packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, *routed* formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a "hop count" metric (a count of 16, or greater, is considered "infinite"). The metric associated with each route returned provides a metric *relative to the sender*.

Response packets received by *routed* are used to update the routing tables if one of the following conditions is satisfied:

(1) No routing table entry exists for the destination network or host, and the metric indicates the destination is "reachable" (i.e. the hop count is not infinite).

(2) The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.

(3) The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.

(4) The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, *routed* records the change in its internal tables and updates the kernel routing table. The change is reflected in the next *response* packet sent.

In addition to processing incoming packets, *routed* also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to insure the invalidation is propagated throughout the local internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks. The response is sent to the broadcast address on nets capable of that function, to the destination address on point-to-point links, and to the router's own address on other networks. The normal routing tables are bypassed when sending gratuitous responses. The reception of responses on each network is used to determine that the network and interface are functioning correctly. If no response is received on an interface, another route may be chosen to route around the interface, or the route may be dropped if no alternative is available.

routed supports several options:

-d Enable additional debugging information to be logged, such as bad packets received.

-g This flag is used on internetwork routers to offer a route to the "default" destination. This is typically used on a gateway to the Internet, or on a gateway that uses another routing protocol whose routes are not reported to other local routers.

-s Supplying this option forces *routed* to supply routing information whether it is acting as an internetwork router or not. This is the default if multiple network interfaces are present, or if a point-to-point link is in use.

-q This is the opposite of the **-s** option.

-t If the **-t** option is specified, all packets sent or received are printed on the standard output. In addition, *routed* will not divorce itself from the controlling terminal so that interrupts from the keyboard will kill the process.

Any other argument supplied is interpreted as the name of file in which *routed*'s actions should be logged. This log contains information about any changes to the routing tables and, if not tracing all packets, a history of recent messages sent and received which are related to the changed route.

In addition to the facilities described above, *routed* supports the notion of "distant" *passive* and *active* gateways. When *routed* is started up, it reads the file */etc/gateways* to find gateways which may not be located using only information from the SIOCGIFCONF *ioctl*. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (i.e. they should have a *routed* process running on the machine). Passive gateways are maintained in the routing tables forever, and information regarding their existence is included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of time, the associated route is deleted. External gateways are also passive, but are not placed in the kernel routing table nor are they included in routing updates. The function of external entries is to inform *routed* that another routing process will install such a route, and that alternate routes to that destination should not be installed. Such entries are only required when both routers may learn of routes to the same destination.

The */etc/gateways* is comprised of a series of lines, each in the following format:

< net| host > *name1* **gateway** *name2* **metric** *value* **< passive| active| external >**

The **net** or **host** keyword indicates if the route is to a network or specific host.

Name1 is the name of the destination network or host. This may be a symbolic name located in */etc/networks* or */etc/hosts* (or, if started after *named*(1M), known to the name server), or an Internet address specified in "dot" notation; see *hosts*(4) and *inet*(7).

Name2 is the name or address of the gateway to which messages should be forwarded.

Value is a metric indicating the hop count to the destination host or network.

One of the keywords **passive, active** or **external** indicates if the gateway should be treated as *passive* or *active* (as described above), or whether the gateway is external to the scope of the *routed* protocol.

FILES

/etc/gateways for distant gateways

SEE ALSO

udp(7) in the TCP Programmer's Guide and Reference.

BUGS

The kernel's ICMP routing tables may not correspond to those of *routed* when ICMP redirects change or add routes.

NAME

rwhod - system status server

SYNOPSIS

/etc/rwhod

DESCRIPTION

Rwhod is the server which maintains the database used by the *rwho*(1) and *ruptime*(1) programs. Its operation is predicated on the ability to *broadcast* messages on a network.

Rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other *rwhod* servers' status messages, validating them, then recording them in a collection of files located in the directory **/usr/spool/rwho.**

The server transmits and receives messages at the port indicated in the "rwho" service specification; see *services*(4). The messages sent and received are of the form:

```
struct    outmp {
          char      out_line[8];      /* tty name */
          char      out_name[8];      /* user id */
          long      out_time;         /* time on */
};

struct    whod {
          char      wd_vers;
          char      wd_type;
          char      wd_fill[2];
          int       wd_sendtime;
          int       wd_recvtime;
          char      wd_hostname[32];
          int       wd_loadav[3];
          int       wd_boottime;
          struct    whoent {
                    struct    outmp we_utmp;
                    int       we_idle;
          } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the *sar*(1M) program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission; they are multiplied by 100 for representation in an integer. The host name included is that returned by the *gethostname*(2) system call, with any trailing domain name omitted. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the *utmp*(4) entry for each non-idle terminal line and a value indicating the time in seconds since a character was last received on the terminal line.

Messages received by the *rwho* server are discarded unless they originated at an *rwho* server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by *rwhod* are placed in files named **whod.hostname** in the directory **/usr/spool/rwho**. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 5 minutes. *Rwhod* performs an *nlist*(3) on **/unix** every 30 minutes to guard against the possibility that this file is not the system image currently operating.

SEE ALSO

rwho(1), ruptime(1).

BUGS

There should be a way to relay status information between networks. Status information should be sent only upon request rather than continuously. People often interpret the server dying or network communication failures as a machine going down.

Some mechanism for cleaning dead machine data out of the spool directory is needed.

NAME

sendmail - send mail over the internet

SYNOPSIS

/usr/lib/sendmail [flags] [address ...]

newaliases

mailq [**-v**]

DESCRIPTION

Sendmail sends a message to one or more *recipients*, routing the message over whatever networks are necessary. *Sendmail* does internetwork forwarding as necessary to deliver the message to the correct place.

Sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; *sendmail* is used only to deliver pre-formatted messages.

With no flags, *sendmail* reads its standard input up to an end-of-file or a line consisting only of a single dot and sends a copy of the message found there to all of the addresses listed. It determines the network(s) to use based on the syntax and contents of the addresses.

Local addresses are looked up in a file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in any alias expansions, e.g., if 'john' sends to 'group', and 'group' includes 'john' in the expansion, then the letter will not be delivered to 'john'.

Flags are:

-ba Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the "From:" and "Sender:" fields are examined for the name of the sender.

-bd Run as a daemon. *Sendmail* will fork and run in background listening on TCP port 25 for incoming SMTP connections. This is normally run from */etc/rc*.

-bi Initialize the alias database. This works only if *sendmail* was built with a DBM library. Otherwise, this option does nothing.

-bm Deliver mail in the usual way (default).

-bp Print a listing of the queue.

-bs Use the SMTP protocol as described in RFC821 on standard input and output. This flag implies all the operations of the **-ba** flag that are compatible with SMTP.

-bt Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.

-bv Verify names only - do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.

-bz Create the configuration freeze file.

-C*file* Use alternate configuration file. *Sendmail* refuses to run as root if an alternate configuration file is specified. The frozen configuration file is bypassed.

-d*X* Set debugging value to *X*.

-F*fullname* Set the full name of the sender.

-f*name* Sets the name of the "from" person (i.e., the sender of the mail). **-f** can only be used by "trusted" users (normally *root, daemon,* and *network*) or if the person you are trying to become is the same as the person you are.

-h*N* Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the

victim of an aliasing loop. If not specified, "Received:" lines in the message are counted.

-n Don't do aliasing.

-o*x value* Set option *x* to the specified *value*. Options are described below.

-q[*time*] Process saved messages in the queue at given intervals. If *time* is omitted, process the queue once. *Time* is given as a tagged number, with 's' being seconds, 'm' being minutes, 'h' being hours, 'd' being days, and 'w' being weeks. For example, "-q1h30m" or "-q90m" would both set the timeout to one hour thirty minutes. If *time* is specified, *sendmail* will run in background. This option can be used safely with **-bd**.

-r*name* An alternate and obsolete form of the **-f** flag.

-t Read message for recipients. To:, Cc:, and Bcc: lines will be scanned for recipient addresses. The Bcc: line will be deleted before transmission. Any addresses in the argument list will be suppressed, that is, they will *not* receive copies even if listed in the message header.

-v Go into verbose mode. Alias expansions will be announced, etc.

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **-o** flag or in the configuration file. These are described in detail in the *Sendmail Installation and Operation Guide.* The options are:

A*file* Use alternate alias file.

c On mailers that are considered "expensive" to connect to, don't initiate immediate connection. This requires queueing.

d*x* Set the delivery mode to *x*. Delivery modes are 'i' for interactive (synchronous) delivery, 'b' for background (asynchronous) delivery, and 'q' for queue only - i.e., actual delivery is done the next time the queue is run.

D Try to automatically rebuild the alias database if necessary.

e*x* Set error processing to mode *x*. Valid modes are 'm' to mail back the error message, 'w' to "write" back the error message (or mail it back if the sender is not logged in), 'p' to print the errors on the terminal (default), 'q' to throw away error messages (only exit status is returned), and 'e' to do special processing for the BerkNet. If the text of the message is not mailed back by modes 'm' or 'w' and if the sender is local to this machine, a copy of the message is appended to the file "dead.letter" in the sender's home directory.

F*mode* The mode to use when creating temporary files.

f Save UNIX-style From lines at the front of messages.

g*N* The default group id to use when calling mailers.

H*file* The SMTP help file.

i Do not take dots on a line by themselves as a message terminator.

m Send to "me" (the sender) also if I am in an alias expansion.

o If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (i.e., commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.

Q*queuedir*	Select the directory in which to queue messages.
r*timeout*	The timeout on reads; if none is set, *sendmail* will wait forever for a mailer. This option violates the word (if not the intent) of the SMTP specification, show the timeout should probably be fairly large.
S*file*	Save statistics in the named file.
s	Always instantiate the queue file, even under circumstances where it is not strictly necessary. This provides safety against system crashes during delivery.
T*time*	Set the timeout on undelivered messages in the queue to the specified time. After delivery has failed (e.g., because of a host being down) for this amount of time, failed messages will be returned to the sender. The default is three days.
t*stz,dtz*	Set the name of the time zone.
u*N*	Set the default user id for mailers.

In aliases, the first character of a name may be a vertical bar to cause interpretation of the rest of the name as a command to pipe the mail to. It may be necessary to quote the name to keep *sendmail* from suppressing the blanks from between arguments. For example, a common alias is:

msgs: "|/usr/ucb/msgs -s"

Aliases may also have the syntax ":include:*filename*" to ask *sendmail* to read the named file for a list of recipients. For example, an alias such as:

poets: ":include:/usr/local/lib/poets.list"

would read */usr/local/lib/poets.list* for the list of addresses making up the group.

Sendmail returns an exit status describing what it did. The codes are defined in <*sysexits.h*>

EX_OK	Successful completion on all addresses.
EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as cannot fork.
EX_NOHOST	Host name not recognized.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

If invoked as *newaliases, sendmail* will rebuild the alias database. This works only if *sendmail* was built with a DBM library. Otherwise, this command does nothing. If invoked as *mailq, sendmail* will print the contents of the mail queue.

FILES

Except for /usr/lib/sendmail.cf, these pathnames are all specified in /usr/lib/sendmail.cf. Thus, these values are only approximations.

/usr/lib/aliases	raw data for alias names
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	help file
/usr/lib/sendmail.st	collected statistics
/usr/spool/mqueue/*	temp files

SEE ALSO

mail(1), aliases(4), mailaddr(5)>
Sendmail - An Internetwork Mail Router (*TCP Programmer's Guide and Reference*)
Sendmail Installation and Operation Guide chapter in this manual.

NAME

slattach, sldetach - attach and detach serial lines as network interfaces

SYNOPSIS

/etc/slattach devname source destination [baudrate]

/etc/sldetach interface-name

MACHINE DEPENDENCY

DESCRIPTION

Slattach is used to assign a serial (tty) line to a network interface using the DARPA Internet Protocol, and to define the source and destination network addresses. The *devname* parameter is the name of the device the serial line is attached to, *e.g.*, **/dev/tty01.** The source and destination are either host names present in the host name data base (see *hosts*(4)), or DARPA Internet addresses expressed in the Internet standard "dot notation". The optional *baudrate* parameter is used to set the speed of the connection; if not specified, the default of 9600 is used.

Only the superuser may attach or detach a network interface.

There should not be a *getty*(1M) or *uugetty*(1M) on the line.

Sldetach is used to remove the serial line that is being used for IP from the network tables and allow it to be used as a normal terminal again. *Interface-name* is the name that is shown by *netstat*(1M).

EXAMPLES

```
/etc/slattach tty01 tom-src genstar
/etc/slattach /dev/tty01 hugo dahl 4800
/etc/sldetach sl01
```

FILES

/dev/spx /etc/hosts
/dev/*
/usr/spool/locks/slippid.*

DIAGNOSTICS

Various messages indicating:
- the specified interface does not exist
- the requested address is unknown
- the user is not the superuser

SEE ALSO

hosts(4), netstat(1M), ifconfig(1M).

CAVEAT

SLIP depends on the Streams Pipe facility, which on some systems is installed as part of Remote File Sharing. If SP is not installed, SLIP will not work.

NAME

slink - streams linker

SYNOPSIS

slink [-v] [-f] [-c file] [func [arg1 arg2 ...]]

DESCRIPTION

Slink is a STREAMS configuration utility which is used to link together the various STREAMS modules and drivers required for System V/386 Streams TCP. Input to *slink* is in the form of a script specifying the STREAMS operations to be performed. Input is normally taken from the file **/etc/strcf**.

The following options may be specified on the *slink* command line:

-c *file* Use *file* instead of **/etc/strcf**.

-v Verbose mode (each operation is logged to *stderr*).

-f Don't fork (i.e. slink will remain in foreground).

The configuration file contains a list of *functions*, each of which is composed of a list of *commands*. Each command is a call to one of the functions defined in the configuration file or to one of a set of bulit-in functions. Among the built-in functions are the basic STREAMS operations *open*, *link*, and *push*, along with several TCP/IP-specific functions.

Slink processing consists of parsing the input file, then calling the user-defined function **boot**, which is normally used to set up the standard configuration at boot time. If a function is specified on the *slink* command line, that function will be called instead of **boot**. Following the execution of the specified function, *slink* goes into the background and remains idle, holding open whatever file descriptors have been opened by the configuration commands.

A function definition has the following form:

```
function-name {
        command1
        command2
        ...
}
```

The syntax for commands is:

```
function arg1 arg2 arg3 ...
```

or

```
var = function arg1 arg2 arg3 ...
```

The placement of newlines is important: a newline must follow the left and right braces and every command. Extra newlines are allowed, i.e. where one newline is required, more than one may be used. A backslash ('\') followed immediately by a newline is considered equivalent to a space, i.e. may be used to continue a command on a new line. The use of other white space characters (spaces and tabs) is at the discretion of the user, except that there must be white space separating the function name and the arguments of a command.

Comments are delimited by '#' and newline, and are considered equivalent to a newline.

Function and variable names may be any string of characters taken from A-Z, a-z, 0-9, and '_', except that the first character cannot be a digit. Function names and variable names occupy separate name spaces. All functions are global and may be forward referenced. All variables are local to the functions in which they occur.

Variables are defined when they appear to the left of an equals ('=') on a command line, e.g.

```
tcp = open /dev/inet/tcp
```

The variable acquires the value returned by the command. In the above example, the value of the

variable *tcp* will be the file descriptor returned by the *open* call.

Arguments to a command may be either variables, parameters, or strings.

A variable that appears as an argument must have been assigned a value on a previous command line in that function.

Parameters take the form of a dollar sign ('$') followed by one or two decimal digits, and are replaced with the corresponding argument from the function call. If a given parameter was not specified in the function call, an error results (e.g. if a command references $3 and only two arguments were passed to the function, an execution error will occur).

Strings are sequences of characters optionally enclosed in double quotes (""). Quotes may be used to prevent a string from being interpreted as a variable name or a parameter, and to allow the inclusion of spaces, tabs, and the special characters '{', '}', '=', and '#'. The backslash ('\') may also be used to quote the characters '{', '}', '=', '#', "", and '\' individually.

The following built-in functions are provided by *slink*:

open *path* — Open the device specified by pathname *path*. Returns a file descriptor referencing the open stream.

link *fd1 fd2* — Link the stream referenced by *fd2* beneath the stream referenced by *fd1*. Returns the link identifier associated with the link. Note: *fd2* cannot be used after this operation.

push *fd module* — Push the module *module* onto the stream referenced by *fd*.

sifname *fd link name* — Send a SIOCSIFNAME (set interface name) ioctl down the stream referenced by *fd* for the link associated with link identifier *link* specifying the name *name*.

unitsel *fd unit* — Send a IF_UNITSEL (unit select) ioctl down the stream referenced by *fd* specifying unit *unit*.

dlattach *fd unit* — Send a DL_ATTACH_REQ message down the stream referenced by *fd* specifying unit *unit*.

initqp *path qname lowat hiwat ...*

Send an INITQPARMS (initialize queue parameters) ioctl to the driver corresponding to pathname *path*. *qname* specifies the queue for which the low and high water marks will be set, and must be one of:

hd	stream head
rq	read queue
wq	write queue
muxrq	multiplexor read queue
muxwq	multiplexor write queue

lowat and *hiwat* specify the new low and high water marks for the queue. Both *lowat* and *hiwat* must be present. To change only one of these parameters, the other may be replaced with a dash ('-'). Up to five *qname lowat hiwat* triplets may be present.

strcat *str1 str2* — Concatenate strings *str1* and *str2* and return the resulting string.

return *val* — Set the return value for the current function to *val*. Note: executing a **return** command does not terminate execution of the current function.

FILES

/etc/strcf

SEE ALSO

strcf(4), intro(7) in the TCP Programmer's Guide and Reference.

NAME

talkd - remote user communication server

SYNOPSIS

/etc/talkd

DESCRIPTION

Talkd is the server that notifies a user that somebody else wants to initiate a conversation. It acts as a repository of invitations, responding to requests by clients wishing to rendezvous to hold a conversation. In normal operation, a *talk* client initiates a rendezvous by sending a CTL_MSG to the server of type LOOK_UP (see *<protocols/talkd.h>*). This causes the server to search its invitation tables to check if an invitation currently exists for the client. If the lookup fails, the caller then sends an ANNOUNCE message causing the server to broadcast an announcement on the callee's login ports requesting contact. When the callee responds, the local server uses the recorded invitation to respond with the appropriate rendezvous address and the caller and callee client programs establish a stream connection through which the conversation takes place.

SEE ALSO

talk(1), write(1)

NAME

/etc/tcp - TCP start/stop script

SYNOPSIS

/etc/tcp start
/etc/tcp stop

DESCRIPTION

/etc/tcp is used to start or stop the TCP software. TCP will start automatically at system startup time if **/etc/tcp** is linked to **/etc/rc2.d/S***name*. Similarly, TCP will stop automatically at system shutdown time if **/etc/tcp** is linked to **/etc/rc0.d/K***name*. See *rc0*(1M) and *rc2*(1M) for further information.

/etc/tcp must be customized for a particular installation before it can be used. The following items must be edited:

Domain name	The environment variable "DOMAIN" must be set to the name of your domain.
Interface configuration	*Ifconfig* commands must be used to set the internet address (and any other desired options) for each of your interfaces. The *ifconfig* line for the loop-back interface should not require modification. See *ifconfig*(1M) for further information.

The following items may need to be edited:

PATH	The supplied path may require modification if commands run by */etc/tcp* are in other directories.
PROCS	The "PROCS" variable contains a space-separated list of names of processes to kill when executing the **stop** function. If additional daemons are used, their names can be added to this list.
Network initialization	Certain network hardware may require the execution of an initialization command. Any such commands should be included in this section.
Daemons	The standard internetworking daemons are started at this point. Any additional daemons or other commands may be included in this section. Any of the standard daemons that are not desired may be removed or commented out.

SEE ALSO

slink(1M), ldsocket(1M), hostname(1), ifconfig(1M), named(1M), inetd(1M), routed(1M), timed(1M), rwhod(1M), sendmail(1M),
rc0(1M), rc2(1M), sh(1), strerr(1M) in the UNIX User's/Administrator's Reference.

NAME

telnetd - DARPA TELNET protocol server

SYNOPSIS

/etc/telnetd

DESCRIPTION

Telnetd is a server which supports the DARPA standard **TELNET** virtual terminal protocol. *Telnetd* is invoked by the internet server (see *inetd*(1M)), normally for requests to connect to the **TELNET** port as indicated by the */etc/services* file (see *services*(4)).

Telnetd operates by allocating a pseudo-terminal device (see *vty*(7)) for a client, then creating a login process which has the slave side of the pseudo-terminal as **stdin, stdout,** and **stderr**. *Telnetd* manipulates the master side of the pseudo-terminal, implementing the **TELNET** protocol and passing characters between the remote client and the login process.

When a **TELNET** session is started up, *telnetd* sends **TELNET** options to the client side indicating a willingness to do *remote echo* of characters, to *suppress go ahead*, and to receive *terminal type information* from the remote client. If the remote client is willing, the remote terminal type is propagated in the environment of the created login process. The pseudo-terminal allocated to the client is configured to operate in ICANON mode, and with TAB3 and ICRNL enabled (see *termio*(7)).

Telnetd is willing to *do*: *echo*, *binary*, *suppress go ahead*, and *timing mark*. *Telnetd* is willing to have the remote client *do*: *binary*, *terminal type*, and *suppress go ahead*.

SEE ALSO

telnet(1)

BUGS

Some **TELNET** commands are only partially implemented.

The **TELNET** protocol allows for the exchange of the number of lines and columns on the user's terminal, but *telnetd* doesn't make use of them.

Because of bugs in the original 4.2 BSD *telnet*(1C), *telnetd* performs some dubious protocol exchanges to try to discover if the remote client is, in fact, a 4.2 BSD *telnet*(1C).

Binary mode has no common interpretation except between similar operating systems (Unix in this case).

The terminal type name received from the remote client is converted to lower case.

The *packet* interface to the pseudo-terminal (see *vty*(7)) should be implemented for intelligent flushing of input and output queues.

Telnetd never sends **TELNET** *go ahead* commands.

NAME

tftpd - DARPA Trivial File Transfer Protocol server

SYNOPSIS

/etc/tftpd

DESCRIPTION

Tftpd is a server that supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the "tftp" service description; see *services*(4). This port number may be overridden (for debugging purposes) by specifying a port number on the command line.

The use of *tftp* does not require an account or password on the remote system. Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling tftp service.

Tftpd is spawned by the "super-server" *inetd*, and therefore must have an entry in *inetd*'sP configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)]. Note that the *tftpd* entry in this file must be "wait": this is to avoid subsequent *selects* from being successful before the first *tftpd* process does its *receive*. *Tftpd* takes care to prevent multiple *tftpd* processes from being spawned to service the same request. (*inetd* is able to continue processing new messages on the port.)

SEE ALSO

inetd(1M), tftp(1), inetd.conf(4), services(4).

WARNINGS

This server is known only to be self consistent (i.e. it operates with the user TFTP program, *tftp*(1)).

The search permissions of the directories leading to the files accessed are not checked if tftp runs as root. The default configuration runs tftpd as user "sync."

NAME

timed - time server daemon

SYNOPSIS

/etc/timed [**-t**] [**-M**] [**-n** *network*] [**-i** *network*]

DESCRIPTION

Timed is the time server daemon and is normally invoked at boot time from the TCP startup script. It synchronizes the host's time with the time of other machines in a local area network running *timed*(1M). These time servers will slow down the clocks of some machines and speed up the clocks of others to bring them to the average network time. The average network time is computed from measurements of clock differences using the ICMP timestamp request message.

The service provided by *timed* is based on a master-slave scheme. When *timed*(1M) is started on a machine, it asks the master for the network time and sets the host's clock to that time. After that, it accepts synchronization messages periodically sent by the master and calls *adjtime*(2) to perform the needed corrections on the host's clock.

It also communicates with *rdate*(1M) in order to set the date globally, and with *timedc*(1M), a timed control program. If the machine running the master crashes, then the slaves will elect a new master from among slaves running with the **-M** flag. A *timed* running without the **-M** flag will remain a slave. The **-t** flag enables *timed* to trace the messages it receives in the file /usr/adm/timed.log. Tracing can be turned on or off by the program *timedc*(1M). *Timed* normally checks for a master time server on each network to which it is connected, except as modified by the options described below. It will request synchronization service from the first master server located. If permitted by the **-M** flag, it will provide synchronization service on any attached networks on which no current master server was detected. Such a server propagates the time computed by the top-level master. The **-n** flag, followed by the name of a network which the host is connected to (see *networks*(4)), overrides the default choice of the network addresses made by the program. Each time the **-n** flag appears, that network name is added to a list of valid networks. All other networks are ignored. The **-i** flag, followed by the name of a network to which the host is connected (see *networks*(4)), overrides the default choice of the network addresses made by the program. Each time the **-i** flag appears, that network name is added to a list of networks to ignore. All other networks are used by the time daemon. The **-n** and **-i** flags are meaningless if used together.

FILES

/usr/adm/timed.log	tracing file for timed
/usr/adm/timed.masterlog	log file for master timed

SEE ALSO

rdate(1M), timedc(1M),
adjtime(2), gettimeofday(3), icmp(7) in the TCP Programmer's Guide and Reference,
date(1) in the UNIX User's Reference manual
Timed Installation and Operation Guide chapter in this manual.
Berkeley Time Syncronization Protocol chapter in the TCP Programmer's Guide and Reference manual.

NAME

timedc - timed control program

SYNOPSIS

timedc [command [argument ...]]

DESCRIPTION

Timedc is used to control the operation of the *timed* program. It may be used to:

- measure the differences between machines' clocks,
- find the location where the master time server is running,
- enable or disable tracing of messages received by *timed*, and
- perform various debugging actions.

Without any arguments, *timedc* will prompt for commands from the standard input. If arguments are supplied, *timedc* interprets the first argument as a command and the remaining arguments as parameters to the command. The standard input may be redirected causing *timedc* to read commands from a file. Commands may be abbreviated; recognized commands are:

? [command ...]

help [command ...]
: Print a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

clockdiff host ...
: Compute the differences between the clock of the host machine and the clocks of the machines given as arguments.

trace { on | off }
: Enable or disable the tracing of incoming messages to *timed* in the file /usr/adm/timed.log.

quit
: Exit from timedc.

Other commands may be included for use in testing and debugging *timed*; the help command and the program source may be consulted for details.

FILES

/usr/adm/timed.log	tracing file for timed
/usr/adm/timed.masterlog	log file for master timed

SEE ALSO

rdate(1M), timed(1M),
adjtime(2), icmp(7) in the TCP Programmer's Guide and Reference.
date(1) in the UNIX User's Reference manual.
the *Timed Installation and Operation Guide* chapter in this manual,
the *Berkeley Time Syncronization Protocol* chapter in the TCP Programmer's Guide and Reference manual.

DIAGNOSTICS

?Ambiguous command	abbreviation matches more than one command
?Invalid command	no match found
?Privileged command	command can be executed by root only

NAME

trace, query - routing tools

SYNOPSIS

trace [on|off] machines... **query [-n]** hosts...

DESCRIPTION

trace sends a RIP_TRACE_ON or RIP_TRACE_OFF command to the specified machines. *machine must be specified as an*

query is used request routing information from the specified host. Any packets received in response to a query will be displayed.

These commands are useful for debugging *routed* (*1M*).

SEE ALSO

routed(1M), udp(4)

BUGS

RFC 1058 states that TRACE_ON and TRACE_OFF are not supposed to be supported anymore.

NAME

trpt - transliterate protocol trace

SYNOPSIS

trpt [**-a**] [**-s**] [**-t**] [**-f**] [**-j**] [**-p** hex-address] [system [core]]

DESCRIPTION

Trpt interrogates the buffer of TCP trace records created when a socket is marked for debugging (see *setsockopt*(2)), and prints a readable description of these records. When no options are supplied, *trpt* prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB). The following options may be used to alter this behavior.

-a in addition to the normal output, print the values of the source and destination addresses for each packet recorded.

-s in addition to the normal output, print a detailed description of the packet sequencing information.

-t in addition to the normal output, print the values for all timers at each point in the trace.

-f follow the trace as it occurs, waiting a short time for additional records each time the end of the log is reached.

-j just give a list of the protocol control block addresses for which there are trace records.

-p show only trace records associated with the protocol control block, the address of which follows.

The recommended use of *trpt* is as follows. Isolate the problem and enable debugging on the socket(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the **-A** option to *netstat*(1). Then run *trpt* with the **-p** option, supplying the associated protocol control block addresses. The **-f** option can be used to follow the trace log once the trace is located. If there are many sockets using the debugging option, the **-j** option may be useful in checking to see if any trace records are present for the socket in question.

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

FILES

/unix
/dev/kmem

SEE ALSO

setsockopt(2), netstat(1M)

DIAGNOSTICS

"no namelist" when the system image doesn't contain the proper symbols to find the trace buffer; others which should be self explanatory.

BUGS

Should also print the data for each input or output, but this is not saved in the trace record.

The output format is inscrutable and should be described here.

NAME

intro - introduction to formats of files used by networking commands

DESCRIPTION

This section outlines the formats of various files. The C struct declarations for the file formats are given where applicable. Usually, these structures can be found in header files under the directories **/usr/include, /usr/include/net, /usr/include/netinet,** or **/usr/include/sys.**

References of the type name(1M) refer to entries found in Section 1M in the Reference chapter of the *TCP Administrator's Guide and Reference* manual.

NAME

aliases - aliases file for sendmail

SYNOPSIS

/usr/lib/aliases

DESCRIPTION

This file describes user id aliases used by */usr/lib/sendmail.* It is formatted as a series of lines of the form

name: name_1, name2, name_3, . . .

The *name* is the name to alias, and the *name_n* are the aliases for that name. Lines beginning with white space are continuation lines. Lines beginning with '#' are comments.

Aliasing occurs only on local names. Loops can not occur, since no message will be sent to any person more than once.

After aliasing has been done, local and valid recipients who have a ".forward" file in their home directory have messages forwarded to the list of users defined in that file.

SEE ALSO

sendmail(1M)
the *Sendmail - An Internetwork Mail Router* chapter in the *TCP Programmer's Guide and Reference;*
the *Sendmail Installation and Operation Guide* chapter in the *TCP Adminstrator's Guide and Reference;*

NAME

hosts - list of hosts on network

DESCRIPTION

The file **/etc/hosts** is a list of hosts that share the network, including the local host. It is referred to by programs that need to translate between host names and DARPA Internet addresses when the name server [see *named*(1M)] is not being used. Each line in the file describes a single host on the network and consists of three fields separated by any number of blanks or tabs:

address name aliases ...

where

address is the DARPA Internet address. Unless another type of address is required by some host on the network, *address* should be a Class A address, which takes the form *net.node*, where *net* is the network number from **/etc/networks** (see *networks*(4)), which must be between 0 and 127; and *node* is a value which must be unique for each host and be between 0 and 16777215.

name is the official name of the host. If the host is a computer system running UNIX, it must claim this host name by executing *hostname*(1M) when it is initializing itself.

aliases... is a list of alternate names for the host. Aliases can be used in network commands in place of the official name.

It is suggested that you specify the *hostname* and the *node name* [see *hostname*(1) and *uname*(1)] as aliases of one another for each machine listed in the **/etc/hosts** file.

The routines which search this file ignore comments (portions of lines beginning with **#**) and blank lines.

Internet addresses can actually take one of four forms:

A *A* is a simple 32-bit integer.

A.B *A* is an eight-bit quantity occupying the high-order byte and *B* is a 24-bit quantity occupying the remaining bytes. This form is suitable for a Class A address of the form *net.node*.

A.B.C *A* is an eight-bit quantity occupying the high-order byte; *B* is an eight-bit quantity occupying the next byte; and *C* is a 16-bit quantity occupying the remaining bytes. This form is suitable for a Class B address of the form **128.***net.node*.

A.B.C.D The four parts each occupy a byte in the address.

EXAMPLE

```
#          Engineering network
192.35.53.1     bellhw.BellTech.COM bellhw      # Bell Hardware
192.35.53.2     bellsw.BellTech.COM bellsw      # Bell Software
192.35.53.85    bellfw.BellTech.COM bellfw      # Bell Firmware
```

FILES

/etc/hosts

SEE ALSO

hostname(1), uname(1) in the *TCP User's Guide and Reference*, networks(4), and inet(7) in the *TCP Programmer's Guide and Reference*.

NAME

hosts.equiv - list of trusted hosts

DESCRIPTION

Hosts.equiv resides in directory */etc* and contains a list of trusted hosts. When an *rlogin*(1) or *rcmd*(1) request from such a host is made, and the initiator of the request is in */etc/passwd*, then no further validity checking is done. That is, *rlogin* does not prompt for a password, and *rsh* completes successfully. So a remote user is "equivalenced" to a local user with the same user ID when the remote user is in *hosts.equiv*.

The format of *hosts.equiv* is a list of names, as in this example:

```
host1
host2
```

A line consisting of a simple host name means that anyone logging in from that host is trusted. The *.rhosts* file has the same format as *hosts.equiv*. When user *XXX* executes *rlogin* or *rcmd*, the *.rhosts* file from *XXX*'s home directory is conceptually concatenated onto the end of *hosts.equiv* for permission checking. In the special case when the user is the super-user then only the */.rhosts* file is checked.

It is also possible to have two entries (separated by a single space) on a line of these files. In this case, if the remote host is equivalenced by the first entry, then the user named by the second entry is allowed to log in as anyone, that is, specify any name to the **-l** flag (provided that name is in the */etc/passwd* file, of course). Thus

```
laidbak ez
```

allows *ez* to log in from laidbak as anyone. The usual usage would be to put this entry in the *.rhosts* file in the home directory for *derek*. Then *ez* may log in as *derek* when coming from laidbak.

FILES

/etc/hosts.equiv
$HOME/.rhost

SEE ALSO

rlogin(1), rcmd(1), rhosts(4).

NAME

inetd.conf - configuration file for inetd (internet "super-server")

DESCRIPTION

inetd.conf is the configuration file for the *inetd*(1M) System V/386 TCP internetworking "super-server".

The file consists of a series of single-line entries, each entry corresponding to a service to be invoked by *inetd*. These services are connection-based, datagram, or "internal".

Internal services are those supported by the *inetd* program: these services are "echo", "discard", "chargen" (character generator), "daytime" (human readable time), and "time" (machine readable time, in the form of the number of seconds since midnight, January 1, 1900). All of these services are tcp based. (For details of these services, consult the appropriate RFC from the DDN Network Information Center.)

Each service, including internal services, must have a valid entry in **/etc/services**(4). In the case of an internal service, its name must correspond to the official name of the service: that is, the first entry in **/etc/services**.

Each entry has a series of space- or tab-separated fields. (No field, except for the last one, may be omitted.) The fields are as follows:

service name
: Name of a valid service in **/etc/services**, as described above.

socket type
: One of "stream", "dgram", or "raw", depending on whether the socket type is stream, datagram, or raw [see *socket*(2)].

protocol
: Name of a valid protocol (for example, "tcp") specified in **/etc/protocols**(4).

wait/nowait
: Specifies whether the socket can be made available for new connections while there is still data waiting on the socket. The value is always "nowait" unless it is a datagram socket. If it is a datagram socket, the value is usually "wait", although "nowait" is possible in some cases. (Note that *tftpd* is an exception in that it must have "wait" specified, and yet the socket can continue to process messages on the port.)

user
: Name of the user as whom the server should run. This allows servers to be run with less permission than root.

server program
: Except in the case of internal services, full pathname of the server program to be invoked by *inetd* when a request is waiting on a socket. For an internal service, the value is "internal".

server program arguments
: Arguments to the server program, starting with *argv*[0], which is the name of the program. For an internal service, the value is "internal".

Comments are denoted by a "#" at the beginning of a line.

The distribution **inetd.conf** file contains prototype entries; refer to these entries when editing the file.

EXAMPLE

```
.
.
.
ftp     stream  tcp     nowait  root    /etc/ftpd       ftpd
telnet  stream  tcp     nowait  root    /etc/telnetd    telnetd
login   stream  tcp     nowait  root    /etc/rlogind    rlogind
exec    stream  tcp     nowait  root    /etc/rexecd     rexecd
```

```
finger    stream   tcp    nowait   sync     /etc/fingerd   fingerd
echo      stream   tcp    nowait   root     internal
discard   stream   tcp    nowait   root     internal
chargen   stream   tcp    nowait   root     internal
daytime   stream   tcp    nowait   root     internal
time      stream   tcp    nowait   root     internal
echo      dgram    udp    wait     root     internal
discard   dgram    udp    wait     root     internal
chargen   dgram    udp    wait     root     internal
daytime   dgram    udp    wait     root     internal
time      dgram    udp    wait     root     internal
                  .
                  .
                  .
```

SEE ALSO

fingerd(1M), ftpd(1M), inetd(1M), rexecd(1M), rlogind(1M), rshd(1M), telnetd(1M), tftpd(1M), protocols(4), services(4).

NAME

netrc - login file for remote networks

DESCRIPTION

If the **.netrc** file exists, it will be used by *ftp*(1) for automatic login on the remote host. For each remote host, the file contains a one-line entry that describes the login data for the user on that host.

An entry may consist of up to three blank-separated fields introduced by keywords. The keyword is followed by the literal data needed for login. The following keywords are available:

machine The hostname of the machine.

login The user login name for that host.

password (Optional) The user's password on that host. **NOTE:** The literal password must be given in clear text; it is not encrypted.

If the **.netrc** file includes the password feature, permissions on the file must be set to prohibit reading by group and others; the file will not otherwise take effect.

EXAMPLE

The following example entry allows automatic login on the "admin" host by a user named "superuser" whose password is "open".

```
machine admin login superuser password open
```

FILES

$HOME/.netrc

SEE ALSO

ftp(1).

WARNING

For security reasons, use of the password feature is not recommended.

NAME

networks - names and numbers for the internet

DESCRIPTION

The file **/etc/networks** lists networks on the internet. Each line describes a single network and consists of the following blank separated fields:

name number aliases ...

where

name is the official name of the network. All hosts on the internet should use the same official name for a given network.

number is the network number, which serves as part of the DARPA Internet address for each host on the internet. All hosts on the internet must use the same number for a given network.

aliases ... is a blank-separated list of local aliases for the network.

The routines which search this file ignore comments (portions of lines beginning with #) and blank lines.

EXAMPLE

```
# Building 1 Internet
Engineering 1       #R&D
Production 2        #Administration, etc.
```

SEE ALSO

hosts(4).

FILES

/etc/networks

NAME

protocols - list of Internet protocols

DESCRIPTION

The file **/etc/protocols** lists known DARPA Internet protocols. Each line describes a single protocol and consists of the following blank separated fields:

name number aliases ...

where

name is the official name of the protocol.

number is the protocol number.

aliases ... is a blank-separated list of local aliases for the protocol.

The routines which search this file ignore comments (portions of lines beginning with **#**) and blank lines.

Protocol names and numbers are specified by the DDN Network Information Center. Do not change this file.

FILES

/etc/protocols

SEE ALSO

socket(2), slink(1M), ldsocket(1M)

NAME

resolver - resolver configuration file

SYNOPSIS

/etc/resolv.conf

DESCRIPTION

The resolver configuration file contains information that is read by the resolver routines the first time they are invoked by a process. The file is designed to be human readable and contains a list of name-value pairs that provide various types of resolver information.

On a normally configured system this file should not be necessary. The only name server to be queried will be on the local machine and the domain name is retrieved from the system.

The different configuration options are:

nameserver
followed by the Internet address (in dot notation) of a name server that the resolver should query. At least one name server should be listed. Up to MAXNS (currently 3) name servers may be listed, in that case the resolver library queries them in the order listed. If no **nameserver** entries are present, the default is to use the name server on the local machine. (The algorithm used is to try a name server, and if the query times out, try the next, until out of name servers, then repeat trying all the name servers until a maximum number of retries are made).

domain followed by a domain name, that is the default domain to append to names that do not have a dot in them. If no **domain** entries are present, the domain returned by *gethostname* (2) is used (everything after the first '.'). Finally, if the host name does not contain a domain part, the root domain is assumed.

The name value pair must appear on a single line, and the keyword (e.g. **nameserver**) must start the line. The value follows the keyword, separated by white space.

EXAMPLE

```
domain      BellTech.COM
nameserver  192.35.52.1
nameserver  192.35.52.2
```

FILES

/etc/resolv.conf

SEE ALSO

gethostent(3) and resolver(3) in the *TCP Programmer's Guide and Reference* manual, named(1M), hosts(4).
Name Server Operations Guide for BIND chapter in this manual.

NAME

rhosts - remote equivalent users

DESCRIPTION

These files grant permission for remote users to use local user names without knowing the corresponding user passwords. This is known as making the remote user "equivalent" to the local user, and is convenient, for example, when one person owns user names on more than one host.

If a user's home directory contains a file named **.rhosts**, remote users specified in the file are equivalent to the local user. Each user specification in the file consists of the remote user host name and user name, separated by a space. (If an asterisk is substituted for either name, any name will match.) For security reasons, **.rhosts** must belong to the user granting the equivalence or to root.

The file **/etc/hosts.equiv** is a list of remote hosts with matching-name equivalence. The file lists remote hosts one per line. On each host listed in **/etc/hosts.equiv**, a remote user with the same name as a local user is equivalent to the local user. In effect, the users are the same if the names are the same.

FILES

$HOME/.rhosts
/etc/hosts.equiv

SEE ALSO

rcmd(1), rcp(1), rlogin(1).

WARNINGS

When a system is listed in **/etc/hosts.equiv**, its security must be as good as local security. One insecure system mentioned in **/etc/hosts.equiv** can compromise the security of an entire network.

NAME

services - list of Internet services

DESCRIPTION

The file **/etc/services** lists known DARPA Internet services. Each line describes a single service and consists of the following blank separated fields:

name number/protocol aliases ...

where

name is the official name of the service.

number is the service number.

protocol is the name of the protocol (see *protocols*(4)) used by the service.

aliases ... is a blank-separated list of local aliases for the service.

The routines which search this file ignore comments (portions of lines beginning with #) and blank lines.

Service names and numbers are specified by the DDN Network Information Center. Do not change this file unless you are familiar with DARPA Internet internals.

FILES

/etc/services

SEE ALSO

inetd(1M), inetd.conf(4).

NAME

/etc/sockcf - socket configuration file

DESCRIPTION

/etc/sockcf contains information about the protocols that are to be accessed via the socket interface. This file is read by *ldsocket*(1M) at boot time.

/etc/sockcf contains one line per protocol which specifies the address family, protocol type, protocol number, flags, and STREAMS device for the protocol. The flags describe the behavior of the protocol.

The format of a protocol line is:

Family Type Protocol Flags Device

Family can be an address family name or an integer. The following address family names are recognized:

Name	Value	Description
UNSPEC	0	Unspecified
UNIX	1	Local to host (pipes, portals)
INET	2	Internetwork: TCP, UDP, etc.
IMPLINK	3	Arpanet IMP addresses
PUP	4	PUP protocols, e.g. BSP
CHAOS	5	MIT CHAOS protocols
NS	6	XEROX NS protocols
NBS	7	NBS protocols
ECMA	8	European Computer Manufacturers
DATAKIT	9	Datakit protocols
CCITT	10	CCITT protocols: X.25, etc.
SNA	11	IBM SNA
DECnet	12	DECnet
DLI	13	Direct Data Link Interface
LAT	14	LAT
HYLINK	15	NSC Hyperchannel
APPLETALK	16	Apple Talk

Type can be a type name or an integer. The following type names are recognized:

Name	Value	Description
STREAM	1	Stream socket
DGRAM	2	Datagram socket
RAW	3	Raw protocol interface
RDM	4	Reliably delivered message
SEQPACKET	5	Sequenced packet stream

Protocol is the protocol number associated with the protocol.

Flags is a string of flag characters describing the protocol. The recognized flag characters are:

M This protocol supports atomic messages only.

C Connections are required.

A Messages contain addresses.

R Rights can be passed with this protocol.

P The protocol number must be bound to the stream. This is required to support raw IP sockets.

/etc/sockcf may contain comments, which are delimited by '#' and newline.

The standard **/etc/sockcf** contains the following entries:

```
INET    STREAM 6     C     /dev/inet/tcp
INET    DGRAM  17    AM    /dev/inet/udp
INET    RAW    1     AM    /dev/inet/icmp
INET    RAW    0     AMP   /dev/inet/rip
```

Because of the way the kernel builds the protocol switch table, the last protocol specified for a type, *e.g.* RAW will become the default. For this reason, it is important to ensure that the default protocol is the last one listed.

FILES

/etc/sockcf

SEE ALSO

ldsocket(1M), intro(7), socket(2).

NAME

/etc/strcf - STREAMS Configuration File for TCP/IP

DESCRIPTION

/etc/strcf contains the script that is executed by *slink*(1M) to perform the STREAMS configuration operations required for TCP/IP.

The standard **/etc/strcf** file contains several functions that perform various configuration operations, along with a sample **boot** function. Normally, only the **boot** function must be modified to customize the configuration for a given installation. In some cases, however, it may be necessary to change existing functions or add new functions.

The following functions perform basic linking operations:

Function **tp** is used to set up the link between a transport provider, such as TCP, and IP.

```
#
# tp - configure transport provider (i.e. tcp, udp, icmp)
# usage: tp devname
#
tp {
        p = open $1
        ip = open /dev/inet/ip
        link p ip
}
```

Function **linkint** links the specified streams and does a **sifname** operation with the given name.

```
#
# linkint - link interface to ip or arp
# usage: linkint top bottom ifname
#
linkint {
        x = link $1 $2
        sifname $1 x $3
}
```

Function **aplinkint** performs the same function as **linkint** for an interface that uses the **arpproc** module.

```
#
# aplinkint - like linkint, but arpproc is pushed on dev
# usage: aplinkint top bottom ifname
#
aplinkint {
        push $2 arpproc
        linkint $1 $2 $3
}
```

The following functions are used to configure different types of Ethernet interfaces:

Function **uenet** is used to configure an Ethernet interface for a cloning device driver that uses the *unit select* ioctl to select the desired interface. The interface name is constructed by concatenating the supplied prefix and the unit number.

```
#
# uenet - configure ethernet-type interface for cloning driver using
#       unit select
# usage: uenet ip-fd devname ifprefix unit
```

```
#
uenet {
        ifname = strcat $3 $4
        dev = open $2
        unitsel dev $4
        aplinkint $1 dev ifname
        dev = open $2
        unitsel dev $4
        arp = open /dev/inet/arp
        linkint arp dev ifname
}
```

Function **denet** performs the same function as **uenet**, except that *DL_ATTACH* is used instead of *unit select*.

```
#
# denet - configure ethernet-type interface for cloning driver using
#       DL_ATTACH
# usage: denet ip-fd devname ifprefix unit
#
denet {
        ifname = strcat $3 $4
        dev = open $2
        dlattach dev $4
        aplinkint $1 dev ifname
        dev = open devname
        dlattach dev $4
        arp = open /dev/inet/arp
        linkint arp dev ifname
}
```

Function **cenet** is used to configure an Ethernet interface for a cloning device driver that uses a different major number for each interface. The device name is formed by concatenating the supplied device name prefix and the unit number. The interface name is formed in a similar manner using the interface name prefix.

```
#
# cenet - configure ethernet-type interface for cloning driver with
#       one major per interface
# usage: cenet ip-fd devprefix ifprefix unit
#
cenet {
        devname = strcat $2 $4
        ifname = strcat $3 $4
        dev = open devname
        aplinkint $1 dev ifname
        dev = open devname
        arp = open /dev/inet/arp
        linkint arp dev ifname
}
```

Function **senet** is used to configure an Ethernet interface for a non-cloning device driver. Two different device nodes must be specified for IP and ARP.

```
#
```

```
# senet - configure ethernet-type interface for non-cloning driver
# usage: senet ip-fd ipdevname arpdevname ifname
#
senet {
        dev = open $2
        aplinkint $1 dev $4
        dev = open $3
        arp = open /dev/inet/arp
        linkint arp dev $4
}
```

Function **senetc** is like **senet**, except that it allows the specification of a convergence module to be used with the ethernet driver (e.g. for the 3B2 emd driver).

```
#
# senetc - configure ethernet-type interface for non-cloning driver
#       using convergence module
# usage: senetc ip-fd convergence ipdevname arpdevname ifname
#
senetc {
        dev = open $3
        push dev $2
        aplinkint $1 dev $5
        dev = open $4
        push dev $2
        arp = open /dev/inet/arp
        linkint arp dev $5
}
```

Function **loopback** is used to configure the loopback interface.

```
#
# loopback - configure loopback device
# usage: loopback ip-fd
#
loopback {
        dev = open /dev/llcloop
        linkint $1 dev lo0
}
```

Function **slip** is used to configure a SLIP interface. This function is not normally executed at boot time. Rather, the *slattach* (1M) command runs *slink* specifying **slip** on the command line.

```
#
# slip - configure slip interface
# usage: slip unit
#
slip {
        ip = open /dev/inet/ip
        s = open /dev/slip
        ifname = strcat sl $1
        unitsel s $1
        linkint ip s ifname
}
```

Function **boot** is called by default when *slink* is executed. Normally, only the *interfaces* section and possibly the *queue params* section will have to be customized for a given installation. Examples are provided for the various Ethernet driver types.

```
#
# boot - boot time configuration
#
boot {
        #
        # queue params
        #
        initqp /dev/inet/udp rq 8192 40960
        initqp /dev/inet/ip muxrq 8192 40960 rq 8192 40960
        #
        # transport
        #
        tp /dev/inet/tcp
        tp /dev/inet/udp
        tp /dev/inet/icmp
        #
        # interfaces
        #
        ip = open /dev/inet/ip
        senetc ip eli /dev/emd0 /dev/emd1 en0
#       uenet ip /dev/abc en 0
#       denet ip /dev/def en 0
#       cenet ip /dev/ghi en 0
#       senet ip /dev/jkl0 /dev/jkl1 en0
        loopback ip
}
```

FILES

/etc/strcf

SEE ALSO

slink(1M), intro(7). eroff -man intro.4 eroff -man aliases.4 eroff -man hosts.4 eroff -man hosts.eq.4 eroff -man inetd.conf.4 eroff -man netrc.4 eroff -man networks.4 eroff -man protocols.4 eroff -man resolver.4 eroff -man rhosts.4 eroff -man services.4 eroff -man sockcf.4 eroff -man strcf.4 sc aliases.4 < sc.input sc hosts.4 < sc.input sc hosts.eq.4 < sc.input sc inetd.conf.4 < sc.input sc intro.4 < sc.input sc netrc.4 < sc.input sc networks.4 < sc.input sc protocols.4 < sc.input sc resolver.4 < sc.input sc rhosts.4 < sc.input sc services.4 < sc.input sc sockcf.4 < sc.input sc strcf.4 < sc.input

NAME

intro - introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities and concepts such as mail address formats, etc.

NAME

mailaddr - mail addressing description

DESCRIPTION

Mail addresses are based on the ARPANET protocol listed at the end of this manual page. These addresses are in the general format

user@domain

where a domain is a hierarchical dot separated list of subdomains. For example, the address

stevea@laiter.lachman.com

is normally interpreted from right to left: the message should go to the Lachman gateway, after which it should go to the local host laiter. When the message reaches laiter it is delivered to the user "stevea".

Unlike some other forms of addressing, this does not imply any routing. Thus, although this address is specified as an RFC822 address, it might travel by an alternate route if that were more convenient or efficient. For example, at Lachman, the associated message would probably go directly to laiter over the Ethernet rather than going via the Lachman mail gateway.

Abbreviation.

Under certain circumstances it may not be necessary to type the entire domain name. In general, anything following the first dot may be omitted if it is the same as the domain from which you are sending the message. For example, a user on "laisagna.Lachman.COM" could send to "stevea@laiter" without adding the "Lachman.COM" since it is the same on both sending and receiving hosts.

Certain other abbreviations may be permitted as special cases. For example, at Lachman, Internet hosts may be referenced without adding the "Lachman.COM" as long as their names do not conflict with a local host name.

Compatibility.

Certain old address formats are converted to the new format to provide compatibility with the previous mail system. In particular,

user@host.ARPA

is allowed and

host:user

is converted to

user@host

to be consistent with the *rcp*(1) command.

Also, the syntax

host!user

is converted to:

user@host.UUCP

This is normally converted back to the "host!user" form before being sent on for compatibility with older UUCP hosts.

The current implementation is not able to route messages automatically through the UUCP network. Until that time you must explicitly tell the mail system which hosts to send your message through to get to your final destination.

Case Distinctions.

Domain names (i.e., anything after the "@" sign) may be given in any mixture of upper and lower case with the exception of UUCP hostnames. Most hosts accept any combination of case in user names, with the notable exception of MULTICS sites.

Route-addrs.

Under some circumstances it may be necessary to route a message through several hosts to get it to the final destination. Normally this routing is done automatically, but sometimes it is desirable to route the message manually. Addresses which show these relays are termed "route-addrs." These use the syntax:

<@hosta,@hostb:user@hostc>

This specifies that the message should be sent to hosta, from there to hostb, and finally to hostc. This path is forced even if there is a more efficient path to hostc.

Route-addrs occur frequently on return addresses, since these are generally augmented by the software at each host. It is generally possible to ignore all but the "user@domain" part of the address to determine the actual sender.

Postmaster.

Every site is required to have a user or user alias designated "postmaster" to which problems with the mail system may be addressed.

Other Networks.

Some other networks can be reached by giving the name of the network as the last component of the domain. *This is not a standard feature* and may not be supported at all sites. For example, messages to CSNET or BITNET sites can often be sent to "user@host.CSNET" or "user@host.BITNET" respectively.

BUGS

The RFC822 group syntax ("group:user1,user2,user3;") is not supported except in the special case of "group:;" because of a conflict with old berknet-style addresses.

Route-Address syntax is ugly.

UUCP- and RFC822-style addresses do not coexist politely.

SEE ALSO

mailx(1), sendmail(1M). RFC822.

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

2. Does the publication cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating)._____

NAME _____ DATE _____
TITLE _____
COMPANY NAME/DEPARTMENT _____
ADDRESS _____ PHONE ( ) _____
CITY _____ STATE _____ ZIP CODE _____
(COUNTRY)

Please check here if you require a written reply. ☐

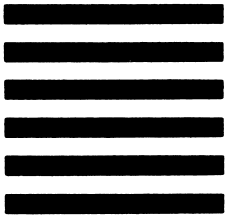
WE'D LIKE YOUR COMMENTS . . .

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

If you are in the United States, use the preprinted address provided on this form to return your comments. No postage is required. If you are not in the United States, return your comments to the Intel sales office in your country. For your convenience, international sales office addresses are printed on the last page of this document.

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 79 HILLSBORO, OR



POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
OMSO Technical Publications, MS: HF3-72
5200 N.E. Elam Young Parkway
Hillsboro, OR 97124-9978

INTERNATIONAL SALES OFFICES

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051

BELGIUM
Intel Corporation SA
Rue des Cottages 65
B-1180 Brussels

DENMARK
Intel Denmark A/S
Glentevej 61-3rd Floor
dk-2400 Copenhagen

ENGLAND
Intel Corporation (U.K.) LTD.
Piper's Way
Swindon, Wiltshire SN3 1RJ

FINLAND
Intel Finland OY
Ruosilante 2
00390 Helsinki

FRANCE
Intel Paris
1 Rue Edison-BP 303
78054 St.-Quentin-en-Yvelines Cedex

ISRAEL
Intel Semiconductor LTD.
Atidim Industrial Park
Neve Sharet
P.O. Box 43202
Tel-Aviv 61430

ITALY
Intel Corporation S.P.A.
Milandfiori, Palazzo E/4
20090 Assago (Milano)

JAPAN
Intel Japan K,K.
Flower-Hill Shin-machi
1-23-9, Shinmachi
Setagaya-ku, Tokyo 15

NETHERLANDS
Intel Semiconducto (Nethderland B.V.)
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam

NORWAY
Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013, Skjetten

SPAIN
Intel Iberia
Calle Zurbaran 28-IZQDA
28010 Madrid

SWEDEN
Intel Sweden A.B.
Dalvaegen 24
S-171 36 SOLNA

SWITZERLAND
Intel Semiconductor A.G.
Talackerstrasse 17
8125 Glattbrugg
CH-8065 Zurich

WEST GERMANY
Intel Semiconductor G.M.B.H.
Seidlestrasse 27
D-800 Munchen