CONVEX

Managing Internet Services and NFS

First Edition

CONVEX Computer Corporation
3000 Waterview Parkway
P.O. Box 833851
Richardson, TX 75083-3851
United States of America
(214)497-4000

Managing Internet Services and NFS

Order No. DSW-108

First Edition
July 1994

CONVEX Press
Richardson, Texas
United States of America

Managing Internet Services and NFS

Order No. DSW-108

Printed in the United States of America

Revision information for Managing Internet Services and NFS

Edition	Document No.	Description
First	710-023930-000	Initial release July 1994. Replaces *CONVEX Internet Services System Manager's Guide* and *CONVEX NFS System Manager's Guide.*

Contents

Part 3 Configuring Internet Services

Figures

Tables

Preface

Purpose and audience

This guide provides system managers with information needed to configure, test, and maintain a TCP/IP local area network; it is not intended to serve as a tutorial on network management or on network software implementation. The reader is assumed to be an experienced system manager familiar with the ConvexOS operating system and with TCP/IP network administration.

Because TCP/IP networking and NFS are mature technologies, you can find information about the "real-world" uses for and internal workings of network services in the technical section of your local book store. Anyone responsible for managing TCP/IP and NFS software should acquire and peruse at least one of these off-the-shelf books.

We recommend the following books:

- *TCP/IP Network Administration*, by Craig Hunt (O'Reilly & Associates, 1992. ISBN 0-937175-82-X).
- *Managing NFS and NIS*, by Hal Stern (O'Reilly & Associates, 1992. ISBN 0-937175-75-7).
- *DNS and BIND in a Nutshell*, by Paul Albitz and Cricket Liu (O'Reilly & Associates, 1992. ISBN 1-56592-010-4).

You can find a concise introduction to basic networking concepts and terminology, an overview of CONVEX networking products, and a description of the CONVEX implementation of TCP/IP and NFS services in *CONVEX Networking Concepts*.

Notational conventions

This section describes notational conventions used in this book.

Command syntax

Consider this example:

```
COMMAND input_file [...] {a | b} [output_file]
   ①        ②       ③     ④        ⑤
```

1. `COMMAND` must be typed as it appears.
2. *input_file* indicates a file name that must be supplied by the user.
3. The horizontal ellipsis in brackets indicates that additional input file names may be supplied.
4. Either `a` or `b` must be supplied.
5. [*output_file*] indicates an optional file name.

General conventions

In general, the following conventions are used in this guide:

- **`Bold constant-width font`** identifies user input in examples.
- *Italic type*
  - Introduces new and important terms
  - Designates user-supplied variables in command-line examples
  - Indicates document titles
- `Constant-width font` designates input and output, including
  - Command names and options
  - System calls
  - Directives, program statements, display examples, printout examples, and error messages returned
- Horizontal ellipsis (...) shows repetition of the preceding item(s).
- Vertical ellipsis shows that lines have been left out of an example.

- Words and abbreviations that indicate keyboard keys you press are identified in a distinctive bold type. For example, **RETURN** refers to the carriage return key. Words separated by a hyphen indicate two keys that you must press simultaneously. For example, **CTRL-X** indicates that you must press and hold down the **CTRL** key and then press the **X** key.
- The word "enter" in a phrase such as "enter **ls**" means that you type the command and then press **RETURN**.
- References to the ConvexOS man pages appear in the form adb(1), where the name of the man page is followed by its section number enclosed in parentheses.
- The backslash (\) character at the end of a command line indicates a continuation line follows.

Note

A Note directs your attention to important information.

Associated documentation

Using this software may require more information than is contained in this guide.

The following documents contain information about installing, configuring, and troubleshooting network interfaces:

- *CONVEX VIOP/VBCU Service Guide* (DHW-051)
- *CONVEX VMEbus I/O Processor Diagnostic Manual* (DHW-241)
- *CONVEX Guide to Attaching Multibus Peripherals* (DHW-020)
- *CONVEX VMEbus Ethernet Controller Service Guide* (DHW-055)
- *CONVEX VMEbus Ethernet Controller Diagnostic Manual* (DHW-245)
- *CONVEX Multibus Ethernet Controller Diagnostic Manual* (DHW-237)
- *CONVEX VMEbus HYPERchannel Controller Diagnostic Manual* (DHW-284)
- *CONVEX VMEbus HYPERchannel Controller Service Guide* (DHW-265)
- *CONVEX Fiber Distributed Data Interface (FDDI) Service Guide* (DHW-275)
- *CONVEX Fiber Distributed Data Interface Diagnostic Manual* (DHW-276)
- *CONVEX HIPPI Service Guide* (DHW-281)
- *CONVEX High Performance Parallel Interface (HIPPI) Diagnostic Manual* (DHW-280)
- *CONVEX FDDI Release Notice*
- *CONVEX FDDI Installation Procedure*
- *CONVEX HIPPI/TCP Release Notice*
- *CONVEX HIPPI/TCP Installation Procedure*

The following document contains general information about configuring devices into ConvexOS:

- *Managing ConvexOS: Configuration Guide* (DSW-030)

For an introduction to basic networking concepts, refer to

- *CONVEX Networking Concepts* (DSW-128)

Detailed information about all of the commands, files, and utilities discussed in this guide can be found in the online ConvexOS man pages.

Ordering documents

To order the current edition of these or any other CONVEX document, send requests to:

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson TX 75083-3851 USA

Please include the order number (DSW or DHW number) or the exact title of the document.

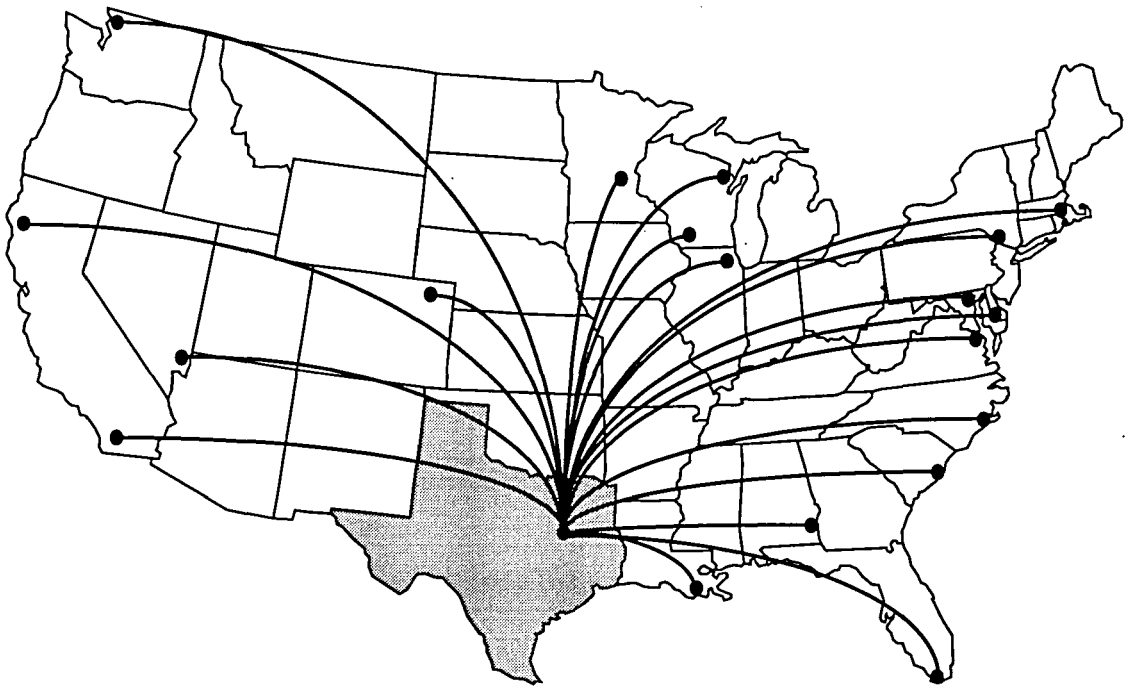
Technical assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC) at the following locations:

- Within the continental U.S., call 1 (800) 952-0379.
- From Canada, call 1 (800) 345-2384.
- All other locations, contact the local CONVEX office.

You can also use the `contact` utility, if you would like to report any problems you may have with ConvexOS or its associated documentation. For more information, refer to the contact(1) man page in *ConvexOS Man Pages for Users*, or the appendix "Reporting problems" in the *ConvexOS Primer* or *Managing ConvexOS: Operations Guide.*

Part 1 Introduction



Introduction 1

Because computer networks are complex, so is the task of administering one. To ease the task of administration, CONVEX Internet Services and NFS software includes utilities and databases used to configure, manage, monitor, and troubleshoot a local area network.

To help acquaint you with CONVEX Internet Services and NFS software, this chapter describes each of these products and lists prerequisites to configuring the software.

Product descriptions

This section describes major components of Internet Services and NFS software.

CONVEX Internet Services

CONVEX Internet Services provides transparent access and resource sharing capabilities among CONVEX systems and a wide variety of other machines, including personal computers, workstations, minicomputers, and supercomputers.

Internet Services software includes

- DARPA Internet protocols that run over Ethernet, HYPERchannel, FDDI, or HIPPI interfaces; or over serial lines via the Serial Line Internet Protocol (SLIP).
- Berkeley networking utilities, used to communicate with other ConvexOS or UNIX systems that run Berkeley Software Distribution (BSD) networking.
- DARPA Internet utilities, used to communicate with systems running TCP/IP, but not necessarily BSD networking.
- System management utilities and databases used to configure, maintain, monitor, and troubleshoot a TCP/IP network.

- Interprocess Communication (IPC) system calls and library routines that facilitate communication among application programs.
- The Berkeley Internet Name Domain server (BIND), a network service that enables clients to name resources and share this information with other hosts on the network.

CONVEX Network File System (NFS)

CONVEX Network File System (NFS) provides transparent access to file systems on remote machines. NFS, developed by Sun Microsystems, Inc., links together heterogeneous systems—including personal computers, workstations, supercomputers, and mainframes—to share resources and files over local area networks. File sharing is accomplished by mounting a remote file system, then reading or writing files in place.

NFS allows a variety of machines and operating systems to act as client or server. Because the environment is transparent, users can access remote files as if these files were on the local machine. Individual workstations can have access to information residing anywhere on the network. In addition to providing transparent access to files, NFS provides users with a single network-wide directory structure.

NFS software includes

- Remote Procedure Call (RPC) facilities that enable client processes to have another process execute a procedure call as if the caller had executed the procedure call in its own address space.
- Network File System (NFS), a distributed file system that enables users to access remote file systems as if they were local.
- The Secure NFS authentication system, used to enhance the security of network environments.
- The automounter utility, which automatically mounts and unmounts remote file systems on an as-needed basis.
- The Network Information Service (NIS), a distributed lookup system that centralizes administration of certain network configuration files.
- NETdisk, used to enable a CONVEX server machine to support booting diskless workstations.

- Lock manager facilities that enable cooperating processes to implement record locking.
- System management utilities and databases used to configure, maintain, monitor, and troubleshoot NFS and NIS services.

Before you begin

This guide provides instructions for configuring and managing NFS and NIS services; it does not discuss hardware and software installation.

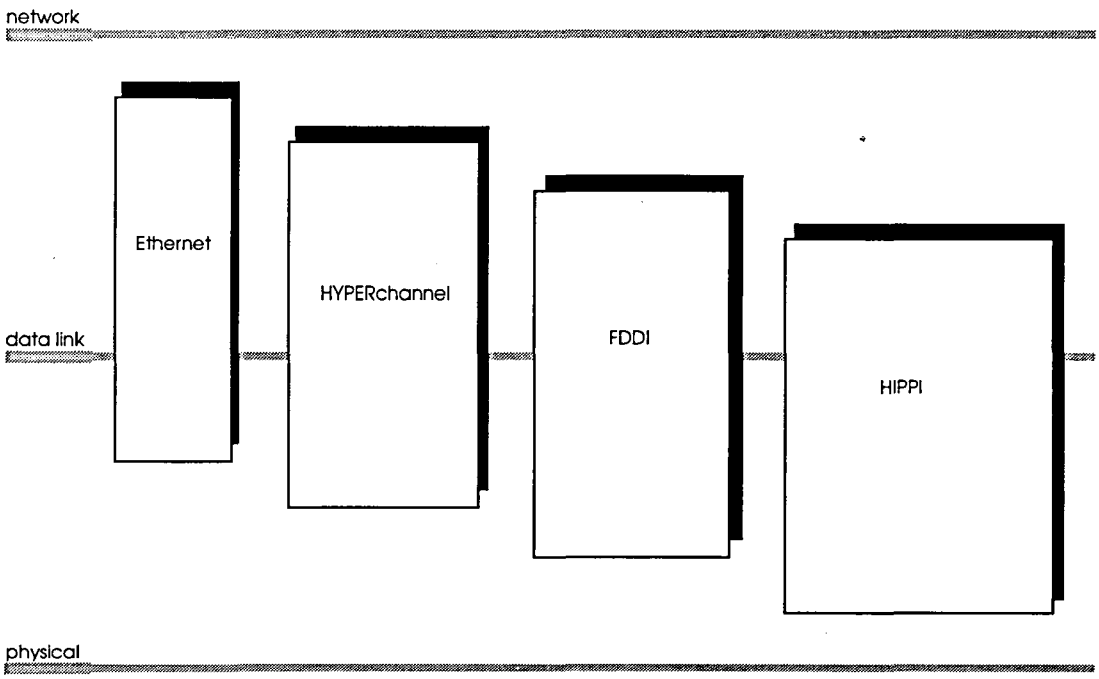
Before attempting any procedure described in this part, you must successfully complete the following tasks in the order in which they are listed:

- Install, configure, and verify operation of network hardware interfaces.
- Install ConvexOS and Utilities
- Install CONVEX Internet Services
- Install CONVEX NFS

For information about installing, configuring, and testing network interfaces, refer to the appropriate service guide and diagnostic manual.

For information about installing the software products listed above, refer to the installation procedures that accompanied the software distribution tape.

Part 2 Configuring network interfaces



Introduction

2

When you add network hardware to your system, you must integrate the new device into ConvexOS and customize the operating system for the interface's particular characteristics. This chapter describes how to integrate network devices into ConvexOS and define parameters for the operating system to use when the system is booted.

Before you begin

This guide covers configuration and management of network software; it does not explain how to install hardware interfaces. You must install, configure, and verify operation of the hardware before attempting any software configuration task.

For complete instructions on installing and testing the hardware, refer to the appropriate service guide and diagnostic manual. Each chapter in this part lists the titles of applicable hardware documents.

Where to find more information

You can find a concise introduction to basic networking concepts and terminology, as well an an overview of CONVEX networking products, in the first two chapters of *CONVEX Networking Concepts*.

Managing ConvexOS: Configuration Guide describes device configuration in detail. You can find supplemental information in the man pages for ex(4), hy(4), hv(4), and uv(4), and in the *ConvexOS Release Notice*.

Configuring an Ethernet interface

3

When you add a network interface to your system or change characteristics of an existing one, you must integrate the device into ConvexOS. This chapter explains how to integrate a newly installed or reconfigured Ethernet interface into ConvexOS. Specifically, it explains how to

- Define the interface in /ioconfig, the system I/O configuration description file located on the SPU.
- Configure a STREAMS protocol stack on systems running ConvexOS V11.0 or later.
- Use the `ifconfig` command to assign an address to a network interface, configure interface parameters, and enable or disable the interface.
- Tune network performance by modifying boot-time parameters.
- Use the `netstat` command to verify network configuration.

This chapter does not explain how to install network hardware; you must install the hardware before attempting any task described here. For complete instructions on installing and configuring Ethernet hardware, refer to the following Convex Press documents:

- *CONVEX VIOP/VBCU Service Guide*
- *CONVEX VMEbus Ethernet Controller Service Guide*
- *CONVEX VMEbus Ethernet Controller Diagnostic Manual*
- *CONVEX Guide to Attaching Multibus Peripherals*
- *CONVEX Multibus Ethernet Controller Diagnostic Manual*

Refer to *Managing ConvexOS: Configuration Guide* for general information about configuring devices into ConvexOS. You can find information specific to Ethernet in the ex(4) man page.

Defining an Ethernet in /ioconfig

Before ConvexOS can communicate over a network interface, it needs to know the physical location and characteristics of the hardware. To define a network interface's location and characteristics, you edit the /ioconfig file to include a description of the interface.

This section provides instructions for adding an Ethernet interface description to the /ioconfig file.

The /ioconfig file, located on the SPU disk, contains descriptions of all the Channel Control Units (CCUs), interfaces, controller boards, and peripheral devices on your system. When the system is booted, the boot process reads /ioconfig to determine the hardware configuration.

To display the /ioconfig file, enter the following command:

```
spu -r /ioconfig
```

Each entry in the /ioconfig file contains several levels of information, usually indented for readability. ConvexOS supports both Multibus and VMEbus Ethernet interfaces. The interface type determines the content of the /ioconfig file entry. Ethernet entries in the /ioconfig file contain the following information:

```
{iop | viop} ccu-slot-number
   {mbus | vme} chassis-number
      ctlr type csr 0xaddress int interrupt-level
         unit unit-number type unit-type
```

where

{iop | viop} *ccu-slot-number*
: Identifies the CCU type and slot number. Specify
iop for a Multibus Ethernet interface;
viop for a VMEbus Ethernet interface.
Slot numbers range from 0 to 7, corresponding to the I/O slot to which the CCU is connected.

{mbus | vme} *chassis-number*
: Identifies interface type and chassis number. Specify
mbus for a Multibus Ethernet Interface;
vme for a VMEbus Ethernet interface.
IOP- and VIOP-type CCUs each support two chassis, numbered zero and one.

ctlr *type*
: Identifies I/O controller type. Specify
LAN-001 for an Excelan Multibus Ethernet Controller;
LAN-007 for an Excelan VMEbus Ethernet Controller.

csr *0xaddress*
Identifies control and status register (CSR) address in hexadecimal format, corresponding to the board strapping for the specific controller type. Refer to the specific controller's documentation to obtain this value.

int *interrupt-level*
Identifies the controller's interrupt level. Interrupt levels range from 0 to 7. Refer to the specific controller's documentation to obtain this value.

unit *unit-number*
Units are numbered sequentially beginning with 0 for each controller type on a bus. The number of units supported by a controller depends on the controller type. Refer to the specific controller's documentation to obtain this information.

type *unit-type*
Identifies the type of unit connected to the controller. Specify **ex** for both Multibus and VMEbus Ethernet interfaces.

After installing Ethernet hardware and booting the system to the SPU level, use the following procedure to define an Ethernet interface in the /ioconfig file.

Step 1 Log in to the SPU as the superuser.

Edit /ioconfig to include an entry for the Ethernet interface. Figure 1 shows an example /ioconfig file entry for a Multibus Ethernet interface.

Figure 1 Example Multibus Ethernet /ioconfig entry

```
iop 6
   mbus 0
      ctlr LAN-001 csr 0x5c0 int 5
         unit 0 type ex
```

Figure 2 shows an example /ioconfig file entry for a VMEbus Ethernet interface.

Figure 2 Example VMEbus Ethernet /ioconfig entry

```
viop 3
   vme 0
      ctlr LAN-007 csr 0x4c0 int 4
         unit 0 type ex
```

Step 2 Changes made to the /ioconfig file take effect when you reboot the system. If you need to modify network-related boot-time parameters, do so before rebooting. Refer to "Tuning an Ethernet interface," page 21, for information about defining boot-time parameters.

Reboot the system by entering

```
/mnt/os/boot
```

Configuring an Ethernet STREAMS protocol stack

On systems running ConvexOS V11.0 or later, Internet Services TCP/IP protocols (TCP, UDP, and IP) are implemented as a STREAMS-based architecture. When the system is booted, the boot process runs the `knetdctl` utility, which causes the STREAMS network daemon, `knetd`, to use information in the knetd.conf file to build the STREAMS protocol stack.

The knetd.conf file included in the standard distribution provides a default configuration definition for a TCP/IP stack and one Ethernet interface stack. You need to modify knetd.conf to correctly build the protocol stack for additional interfaces.

There are two ways to define a STREAMS protocol stack in the knetd.conf file.

- Use the `io2knetcf` utility to generate the knetd.conf file from I/O descriptions found in the /ioconfig file.
- Manually edit the knetd.conf to add the appropriate definitions.

Changes made to the knetd.conf file take effect when the system is rebooted.

The knetd.conf file installed with the software includes explanatory comments; you should read this file before making any changes to it. For information on knetd.conf file syntax, read the knetd.conf(5) man page.

This section explains how to configure the STREAMS TCP/IP protocol stack in the knetdctl.conf file for an Ethernet interface.

Generating the knetd.conf file from /ioconfig

The `knetdctl` utility is used to control the `knetd` daemon, which configures a STREAMS protocol stack from information provided in the /etc/knetd.conf file. The knetd.conf file included in the standard distribution provides a default configuration for the TCP/IP stack and one Ethernet interface. If your system uses more than one network interface, you need to modify this file to correctly build the protocol stack.

You can use the `io2knetcf` utility to generate the /etc/knetd.conf file from information in the /ioconfig file. To use `io2knetcf`, enter

```
io2knetcf > /etc/knetd.conf
```

By default, `io2knetcf` uses the /ioconfig file as input and overwrites the existing knetd.conf file. You can specify alternate input and output files on the command line. For complete information about `io2knetcf`, refer to the io2knetcf(8) man page.

After running `io2knetcf`, you should see entries in the knetd.conf file similar to those shown in Figure 3.

Manually defining additional Ethernet stacks

Follow the steps in this section if your system uses additional Ethernet interfaces.

Step 1 Duplicate the lines shown in Figure 3 for each additional interface.

Figure 3 Ethernet definition in the knetd.conf file

```
mac:0   -         PORT={0}        # open stream to mac device, use unit 0
ip      mac:0     PORT={2048, IP}# link ip to mac stream, dlsap 2048
mac:1   -         PORT={0}        # open stream to mac device, use unit 0
ip      mac:1     PORT={2054, ARP}
mac:2   -         PORT={0}        # open stream to mac device, use unit 0
ip      mac:2     PORT={32821, REVARP}
```

Step 2 Modify the duplicated lines as follows:

- Increment the `mac:` number for each pair.
- Modify the `PORT={0}` number to reflect the controller number for each new interface.

Figure 4 shows an example of a second Ethernet controller definition.

Figure 4 Second Ethernet definition in the knetd.conf file

```
mac:3   -          PORT={1}        # open stream to mac device, use unit 1
ip      mac:3      PORT={2048, IP}# link ip to mac stream, dlsap 2048
mac:4   -          PORT={1}        # open stream to mac device, use unit 1
ip      mac:4      PORT={2054, ARP}
mac:5   -          PORT={1}        # open stream to mac device, use unit 1
ip      mac:5      PORT={32821, REVARP}
```

Changes made to the knetd.conf file take effect when the system is rebooted.

Configuring an Ethernet with `ifconfig`

Before ConvexOS can communicate over a network interface, it needs to know the interface's network address and operating characteristics. This section explains how to use the `ifconfig` command to assign a network address to an Ethernet interface and set parameters that affect its operation.

Step 1 Log in as the superuser.

Step 2 Use `ifconfig` to define the interface's network address and operating characteristics.

You will need to add the `ifconfig` command to your rc.local file so that the interface is automatically configured at boot time; however, it is good practice to first test the command by entering it on the command line.

The `ifconfig` command has the following syntax:

`/etc/ifconfig` *interface host-name* {**`up`** | **`down`**} [**`arp`**] [**`trailers`**] [**`netmask`** *mask*] [**`broadcast`** *address*]

where

interface	A string composed of interface type and unit number (`eth`*n*). Interfaces are numbered in the order in which they appear in the /ioconfig file. For example, the first Ethernet in /ioconfig is assigned the unit name, `eth0`, the second, `eth1`, and so forth.
host-name	Local host name or internet address in dot notation.
`up`	Enable the network interface.
`down`	Disable the interface. Used prior to reconfiguring certain parameters. For example, to change the interface's *mask*, you must disable the interface, define the new mask, and then enable the interface.
`arp`	Enable use of the Address Resolution Protocol (ARP), a mechanism for dynamically mapping between internet and Ethernet addresses. (Refer to "Using the Address Resolution Protocol (ARP)," page 20.)
`-arp`	Disable use of the Address Resolution Protocol (ARP).

trailers Enable use of *trailer* link-level encapsulation of outgoing messages (enabled by default). If a network interface supports trailers, the system, when possible, encapsulates outgoing messages in a way that minimizes the number of memory-to-memory copy operations performed by the receiver. On networks that support ARP, this option indicates that the system should request other systems to use trailers when sending to this host. Similarly, trailer encapsulations are sent to other hosts that have made such requests.

All machines that communicate with each other must use the same `trailers` setting; that is, they must all use trailers or they must all have the `trailers` option disabled.

-trailers Disable trailer link-level encapsulation of outgoing messages.

netmask *mask*
Specify the portion of the internet address to reserve for the combined network and subnet fields. (Refer to "Defining the subnet mask," page 20.)

broadcast *address*
Specify the address used to represent broadcasts to the network. By default, the broadcast address has a host part of all ones. (Refer to "Defining the broadcast address," page 19.)

Note

`ifconfig` has more parameters than those discussed here. Refer to the ifconfig(8C) man page for additional information.

Step 3 After testing the `ifconfig` command, verify that the system accepted the information by entering

ifconfig *interface*

In response, the system displays the network address and operating characteristics of the interface. Sample output is shown in Figure 5.

Figure 5 Using `ifconfig` to verify interface configuration

```
eth0: flags=43<UP,BROADCAST,RUNNING>
      inet 130.150.60.3 netmask ffffff00 broadcast 130.150.60.255
      hardware address aa.00.04.00.2e.28
```

Step 4 If the network appears to be configured properly, edit your rc.local file to add the `ifconfig` command. This will ensure that the network interface is enabled every time the system boots. Figure 6 shows a partial rc.local file containing `ifconfig` commands.

Figure 6 Sample /etc/rc.local file

```
#
/bin/hostname moose
#
# build the networking streams stack (ConvexOS V11.0 or later)
#
if [ "`/etc/knetdctl -q`" = "knetd not configured" ]; then
        /etc/knetdctl -c /etc/knetd.conf
        /etc/knetdctl -r
fi

# configure network interfaces
#
if [ -f /etc/ifconfig ]; then
        /etc/ifconfig eth0 `/bin/hostname` up arp -trailers netmask 0xffffff00
fi
#
```

Defining the broadcast address

The default broadcast address contains a host part of all ones. `ifconfig` enables you to change an interface's broadcast address. CONVEX systems accept broadcasts with a host part of all zeros (for compatibility with systems that use BSD 4.2 broadcasts), but transmits broadcasts with the destination address set to the broadcast address assigned with `ifconfig`.

If a machine on your network does not understand the broadcast address you select, some utilities, such as `rwho`, fail. If this happens, use `ifconfig` to set the broadcast address to the old broadcast address (all zeros) for all machines on the network.

Note

All machines that communicate with each other must use the same broadcast address, either all zeros or all ones. The preferred method is a broadcast address of all ones, as in broadcast 128.194.255.255.

Defining the subnet mask

Ones in the subnet mask indicate bit positions to use for the combined network and subnet fields; zeros mark the positions of bits in the host field. You specify the mask as a single hexadecimal number with a leading 0x, or in dot notation. For example, specifying

```
netmask 0xffffff00
```

or

```
netmask 255.255.255.0
```

both indicate that you want 24 bits of combined network and subnet fields, and 8 bits of host number. For a class B network, this mask logically partitions your 16 bits of host number into an 8-bit subnet field and an 8-bit host field. If you do not supply a `netmask`, the mask is set according to the network class (A, B, or C with 8, 16, or 24 bits of network part, respectively).

Note

To avoid confusion with broadcast addresses, do not use subnet numbers of all zeros or all ones.

Using the Address Resolution Protocol (ARP)

ARP maps logical internet addresses to physical Ethernet addresses by caching the logical mappings between dot-notation addresses (as in the /etc/hosts file) and physical Ethernet addresses. If an interface requests mapping for an address not cached, ARP queues the message that requires the mapping, then broadcasts a message on the associated network to request the address. If a response is received, the new mapping is cached, and the queued message is transmitted.

If you want to communicate with a network host that does not use the ARP protocol, you must use the `arp` program to manually add address mapping information to the local ARP table. The `arp` program forces caching of an ARP table entry for a specific host, so that the ARP protocol does not transmit a packet to a host to get its Ethernet address.

`arp` has the following syntax:

```
arp -s host_name e_address [temp] [pub]
```

where

-s	Adds an entry to the ARP table.
host_name	Remote host as listed in the /etc/hosts file.
e_address	Physical Ethernet address of the remote host. This address is displayed by most systems at boot time and is usually printed on the network controller. You specify it as six hexadecimal numbers separated by colons, as in 08:00:20:06:dd:42 The entry will be permanent unless you also specify **temp**.
temp	Specifies that the ARP table entry is temporary.
pub	Specifies that the entry will be "published"; that is, this system will act as an ARP server, responding to requests for *host_name* even though the host address is not its own.

You can check the current status of the ARP table by entering

```
arp -a
```

The `arp` command has more options than are discussed here. For a complete summary, refer to the arp(8C) man page.

Tuning an Ethernet interface

When the system is booted, the initialization process reads a file containing parameters that affect the way ConvexOS handles CPUs and peripheral devices. Among other uses, these boot-time parameters enable you to optimize network performance.

Most network-related boot-time parameters apply to networking in general, rather than to a specific type of network interface; a few are interface-specific. This section describes the boot-time parameter used to optimize Ethernet performance. For information about general networking parameters and instructions for modifying boot-time parameters, refer to Chapter 7, "Customizing boot-time parameters," page 57.

With the boot-time parameter `viop_enet_proc` you can set the number of send and receive processes for a particular VIOP Ethernet interface. This parameter is used to reduce the amount of memory needed by the VIOP when servicing multiple Ethernet interfaces. While lowering the number of processes helps the VIOP avoid running out of memory, it also slows network performance, so you should decrease the number of processes only if your system fails to boot because of insufficient memory.

The default for this parameter is 4 processes per Ethernet interface.

Verifying Ethernet configuration with `netstat`

Bringing the machine up in multiuser mode indicates that you have correctly configured network interfaces. Usually, the machine simply does not run in multiuser mode if you make a mistake during the configuration process. Of course, you should test the network after the system is up and running in multiuser mode.

To verify network configuration, enter

```
netstat -i
```

If the network is properly configured, the system displays output similar to that shown in Figure 7.

Figure 7 Checking Ethernet configuration with `netstat -i`

```
Name  Mtu   Network     Address   Ipkts    Ierrs  Opkts    Oerrs   Collis
eth0  1500  acme-net    acmes     1520512  0      1327049  0       293
```

The displayed host name and network name are specific to your installation. The name of the interface you just configured should appear. If it does not, you have a problem with your installation or your configuration of the network. In either case, use the troubleshooting procedures outlined in Part 4, "Part 5 Testing and troubleshooting the network," page 329 to find the problem. The troubleshooting part also contains more complete instructions for using `netstat`.

Configuring CONVEX FDDI 4

When you add a network interface to your system or change characteristics of an existing one, you must integrate the device into ConvexOS. This chapter explains how to integrate a newly installed or reconfigured CONVEX FDDI into ConvexOS. Specifically, it explains how to

- Define the interface in /ioconfig, the system I/O configuration description file located on the SPU.
- Configure a STREAMS protocol stack on systems running ConvexOS V11.0 or later.
- Use the `ifconfig` command to assign an address to a network interface, configure interface parameters, and enable or disable the interface.
- Tune network performance by modifying boot-time parameters.
- Use the `netstat` command to verify network configuration.

This chapter does not explain how to install network hardware; you must install the hardware before attempting any task described here. For complete instructions on installing and configuring Ethernet hardware, refer to the following Convex Press documents:

- *CONVEX VIOP/VBCU Service Guide*
- *CONVEX Fiber Distributed Data Interface (FDDI) Service Guide*
- *CONVEX Fiber Distributed Data Interface Diagnostic Manual*
- *CONVEX FDDI Release Notice*
- *CONVEX FDDI Installation Procedure*

Refer to *Managing ConvexOS: Configuration Guide* for general information about configuring devices into ConvexOS. You can find information specific to CONVEX FDDI in the fd(4) man page.

Defining an Ethernet in /ioconfig

Before ConvexOS can communicate over a network interface, it needs to know the physical location and characteristics of the hardware. To define a network interface's location and characteristics, you edit the /ioconfig file to include a description of the interface.

This section provides instructions for adding a CONVEX FDDI description to the /ioconfig file.

The /ioconfig file, located on the SPU disk, contains descriptions of all the Channel Control Units (CCUs), interfaces, controller boards, and peripheral devices on your system. When the system is booted, the boot process reads /ioconfig to determine the hardware configuration.

To display the /ioconfig file, enter the following command:

```
spu -r /ioconfig
```

Each entry in the /ioconfig file contains several levels of information, usually indented for readability. The interface type determines the content of the /ioconfig file entry. CONVEX FDDI entries in the /ioconfig file contain the following information:

viop *ccu-slot-number*
  vme *chassis-number*
    ctlr LAN-208 csr *0xaddress* **int** *interrupt-level*
      unit *unit-number* **type FD**

where

viop *ccu-slot-number*
: Identifies the CCU type and slot number. Slot numbers range from 0 to 7, corresponding to the I/O slot to which the CCU is connected.

vme *chassis-number*
: Identifies interface type and chassis number. VIOP-type CCUs support two chassis, numbered zero and one.

ctlr LAN-208
: Identifies I/O controller type.

csr *0xaddress*
: Identifies control and status register (CSR) address in hexadecimal format, corresponding to the board strapping for the specific controller type. Refer to the *CONVEX FDDI Service Guide* for information about selecting this value.

int *interrupt-level*
Identifies the controller's interrupt level. Interrupt levels range from 0 to 7. Refer to the *CONVEX FDDI Service Guide* for information about selecting this value.

unit *unit-number*
Units are numbered sequentially beginning with 0 for each controller type on a bus. The number of units supported by a controller depends on the controller type. Refer to the *CONVEX FDDI Service Guide* for information about selecting this value.

type FD
Identifies the type of unit connected to the controller.

After installing CONVEX FDDI hardware and booting the system to the SPU level, use the following procedure to define a CONVEX FDDI in the /ioconfig file.

Step 1 Log in to the SPU as the superuser.

Edit /ioconfig to include an entry for the CONVEX FDDI. Figure 8 shows an example /ioconfig file entry for a CONVEX FDDI.

Figure 8 Example FDDI /ioconfig file entry

```
iop 6
   mbus 0
      ctlr LAN-001 csr 0x5c0 int 5
         unit 0 type ex
```

Step 2 Changes made to the /ioconfig file take effect when you reboot the system. If you need to modify network-related boot-time parameters or configure an FDDI STREAMS protocol stack, do so before rebooting. Refer to "Tuning CONVEX FDDI," page 33, for information about defining boot-time parameters. Configuring an FDDI STREAMS protocol stack is explained in the next section.

Reboot the system by entering

```
/mnt/os/boot
```

Configuring an FDDI STREAMS protocol stack

On systems running ConvexOS V11.0 or later, Internet Services TCP/IP protocols (TCP, UDP, and IP) are implemented as a STREAMS-based architecture. When the system is booted, the boot process runs the `knetdctl` utility, which causes the STREAMS network daemon `knetd` to use information in the knetd.conf file to build the STREAMS protocol stack.

The knetd.conf file included in the standard distribution provides a default configuration definition for a TCP/IP stack and one Ethernet interface stack. You need to modify knetd.conf to correctly build the FDDI protocol stack.

There are two ways to define a STREAMS protocol stack in the knetd.conf file.

- Use the `io2knetcf` utility to generate the knetd.conf file from I/O descriptions found in the /ioconfig file.
- Manually edit the knetd.conf to add the appropriate definitions.

Changes made to the knetd.conf file take effect when the system is rebooted.

The knetd.conf file installed with the software includes explanatory comments; you should read this file before making any changes to it. For information on knetd.conf file syntax, read the knetd.conf(5) man page.

This section explains how to define the STREAMS TCP/IP protocol stack in the knetdctl.conf file for a CONVEX FDDI.

Generating the knetd.conf file from /ioconfig

The `knetdctl` utility is used to control the `knetd` daemon, which configures a STREAMS protocol stack from information provided in the /etc/knetd.conf file. The knetd.conf file included in the standard distribution provides a default configuration for the TCP/IP stack and one Ethernet interface. You need to modify this file to correctly build the FDDI protocol stack.

You can use the `io2knetcf` utility to generate the /etc/knetd.conf file from information in the /ioconfig file. To use `io2knetcf`, enter

```
io2knetcf > /etc/knetd.conf
```

By default, `io2knetcf` uses the /ioconfig file as input and overwrites the existing knetd.conf file. You can specify alternate

input and output files on the command line. For complete information on `io2knetcf`, refer to the io2knetcf(8) man page.

After running `io2knetcf`, you should see entries in the knetd.conf file similar to those shown in Figure 10.

Manually defining an FDDI stack

Follow the steps in this section to define a CONVEX FDDI in the knetd.conf file.

Step 1 Duplicate the lines shown in Figure 9 for each FDDI interface.

Figure 9 Ethernet STREAMS stack definition in knetd.conf

```
mac:0   -         PORT={0}        # open stream to mac device, use unit 0
ip      mac:0     PORT={2048, IP}# link ip to mac stream, dlsap 2048
mac:1   -         PORT={0}        # open stream to mac device, use unit 0
ip      mac:1     PORT={2054, ARP}
mac:2   -         PORT={0}        # open stream to mac device, use unit 0
ip      mac:2     PORT={32821, REVARP}
```

Step 2 Replace the `mac` keyword with `macf`, as shown in Figure 10, if your system also has an Ethernet interface.

Figure 10 CONVEX FDDI definition in the knetd.conf file

```
macf:0 -          PORT={0}        # open stream to macf device, use unit 0
ip      macf:0    PORT={2048, IP}# link ip to macf stream, dlsap 2048
macf:1 -          PORT={0}        # open stream to macf device, use unit 0
ip      macf:1    PORT={2054, ARP}
macf:2 -          PORT={0}        # open stream to macf device, use unit 0
ip      macf:2    PORT={32821, REVARP}
```

Step 3 At the beginning of the knetd.conf file in the Modules section, locate the line that reads

```
mac dc /dev/eth
```

Add a line after it that reads

```
macf dc /dev/fddi
```

Changes made to the knetd.conf file take effect when the system is rebooted.

Configuring an FDDI with `ifconfig`

Before ConvexOS can communicate over a network interface, it needs to know the interface's network address and operating characteristics. This section explains how to use the `ifconfig` command to assign a network address to a CONVEX FDDI and set parameters that affect its operation.

Step 1 Log in as the superuser.

Step 2 Use `ifconfig` to define the interface's network address and operating characteristics.

You will need to add the `ifconfig` command to your rc.local file so that the interface is automatically configured at boot time; however, it is good practice to first test the command by entering it on the command line.

The `ifconfig` command has the following syntax:

`/etc/ifconfig` *interface host-name* {**`up`** | **`down`**} [**`arp`**] [**`netmask`** *mask*] [**`broadcast`** *address*]

where

interface — A string composed of interface type and unit number (`fddi`*n*). Interfaces are numbered in the order in which they appear in the /ioconfig file. For example, the first CONVEX FDDI in /ioconfig is assigned the unit name, `fddi0`, the second, `fddi1`, and so forth.

host-name — Local host name or internet address in dot notation.

`up` — Enable the network interface.

`down` — Disable the interface. Used prior to reconfiguring certain parameters. For example, to change the interface's *mask*, you must disable the interface, define the new mask, and then enable the interface.

`arp` — Enable use of the Address Resolution Protocol (ARP), a mechanism for dynamically mapping between internet addresses and physical addresses. (Refer to "Using the Address Resolution Protocol (ARP)," page 32.)

`-arp` — Disable use of the Address Resolution Protocol (ARP).

netmask *mask*

Specify the portion of the internet address to reserve for the combined network and subnet fields. (Refer to "Defining the subnet mask," page 31.)

broadcast *address*

Specify the address used to represent broadcasts to the network. By default, the broadcast address has a host part of all ones. (Refer to "Defining the broadcast address," page 30.)

Note

ifconfig has more parameters than those discussed here. Refer to the ifconfig(8C) man page for additional information.

Step 3 After testing the `ifconfig` command, verify that the system accepted the information by entering

ifconfig *interface*

In response, the system displays the network address and operating characteristics of the interface. Sample output is shown in Figure 11.

Figure 11 Using `ifconfig` to verify interface configuration

```
fddi0: flags=43<UP,BROADCAST,RUNNING>
       inet 130.150.60.6 netmask ffffff00 broadcast 130.150.60.255
       hardware address 30.33.2f.39.32.20
```

Step 4 If the network appears to be configured properly, edit your rc.local file to add the `ifconfig` command. This will ensure that the network interface is enabled every time the system boots. Figure 12 shows a partial rc.local file containing `ifconfig` commands.

Figure 12 Sample /etc/rc.local file

```
#
/bin/hostname moose
#
# build the networking streams stack (ConvexOS V11.0 or later)
#
if [ "`/etc/knetdctl -q`" = "knetd not configured" ]; then
        /etc/knetdctl -c /etc/knetd.conf
        /etc/knetdctl -r
fi

# configure network interfaces
#
if [ -f /etc/ifconfig ]; then
        /etc/ifconfig ex0 `/bin/hostname` up arp -trailers netmask 0xffffff00
        /etc/ifconfig fddi0 dhostwo-f arp up netmask 0xffffff00
fi
#
```

Defining the broadcast address

The default broadcast address contains a host part of all ones. `ifconfig` enables you to change an interface's broadcast address. CONVEX systems accept broadcasts with a host part of all zeros (for compatibility with systems that use BSD 4.2 broadcasts), but transmits broadcasts with the destination address set to the broadcast address assigned with `ifconfig`.

If a machine on your network does not understand the broadcast address you select, some utilities, such as `rwho`, fail. If this happens, use `ifconfig` to set the broadcast address to the old broadcast address (all zeros) for all machines on the network.

Note

All machines that communicate with each other must use the same broadcast address, either all zeros or all ones. The preferred method is a broadcast address of all ones, as in broadcast 128.194.255.255.

Defining the subnet mask

Ones in the subnet mask indicate bit positions to use for the combined network and subnet fields; zeros mark the positions of bits in the host field. You specify the mask as a single hexadecimal number with a leading 0x, or in dot notation. For example, specifying

```
netmask 0xffffff00
```

or

```
netmask 255.255.255.0
```

both indicate that you want 24 bits of combined network and subnet fields, and 8 bits of host number. For a class B network, this mask logically partitions your 16 bits of host number into an 8-bit subnet field and an 8-bit host field. If you do not supply a `netmask`, the mask is set according to the network class (A, B, or C with 8, 16, or 24 bits of network part, respectively).

Note

To avoid confusion with broadcast addresses, do not use subnet numbers of all zeros or all ones.

Using the Address Resolution Protocol (ARP)

ARP maps logical internet addresses to physical addresses by caching the logical mappings between dot-notation addresses (as in the /etc/hosts file) and physical addresses. If an interface requests mapping for an address not cached, ARP queues the message that requires the mapping, then broadcasts a message on the associated network to request the address. If a response is received, the new mapping is cached, and the queued message is transmitted.

If you want to communicate with a host on the network that does not use the ARP protocol, you must use the `arp` program to manually add address mapping information to the local ARP table. The `arp` program forces caching of an ARP table entry for a specific host, so that the ARP protocol does not transmit a packet to a host to get its hardware address.

`arp` has the following syntax:

`arp -s` *host_name e_address* **`[temp] [pub]`**

where

`-s`	Adds an entry to the ARP table.
host_name	Remote host as listed in the /etc/hosts file.
e_address	Physical Ethernet address of the remote host. This address is displayed by most systems at boot time and is usually printed on the network controller. You specify it as six hexadecimal numbers separated by colons, as in 08:00:20:06:dd:42 The entry will be permanent unless you also specify **temp**.
`temp`	Specifies that the ARP table entry is temporary.
`pub`	Specifies that the entry will be "published"; that is, this system will act as an ARP server, responding to requests for *host_name* even though the host address is not its own.

You can check the current status of the ARP table by entering

`arp -a`

The arp command has more options than are discussed here. For a complete summary, refer to the arp(8C) man page.

Tuning CONVEX FDDI

When the system is booted, the initialization process reads a file containing parameters that affect the way ConvexOS handles CPUs and peripheral devices. Among other uses, these boot-time parameters enable you to optimize network performance.

Most network-related boot-time parameters apply to networking in general, rather than to a specific type of network interface; a few are interface-specific. This section describes the boot-time parameters used to optimize FDDI performance. For information about general networking parameters and instructions for modifying boot-time parameters, refer to Chapter 7, "Customizing boot-time parameters," page 57.

When you install CONVEX FDDI software, related boot-time parameters are preset to their default values. By modifying these parameters, you can tune VIOP resource requirements to achieve a balance between performance and resource usage.

You tune the FDDI by manipulating a pair of boot-time parameters, `fd_max_recv` and `fd_max_xmit`, which are defined as follows:

`fd_max_recv` — Specifies the number of buffers used to hold input packets between the time they are received by the FDDI driver and when they are passed to the IP layer for processing.

default = 28, min. = 2, max. = 128

`fd_max_xmit` — Specifies the number of buffers used to hold output packets between the time they are processed by the IP layer and when they are passed to the FDDI driver. When the supply of available buffers is exhausted, the FDDI driver ignores subsequent output packets passed to it from the IP layer.

default =28, min. = 4, max. = 64

Values of these parameters determine how many VIOP memory pages and VIOP windows the FDDI driver needs. The formula for calculating from these parameters the number of required VIOP memory pages is

10 + ((fd_max_recv + fd_max_xmit) * 88 + 511) / 4096

The formula for calculating the number of required VIOP windows is

7 + 2(fd_max_recv + fd_max_xmit)

For example, by using the default values for `fd_max_recv` and `fd_max_xmit` (28 and 28), we calculate VIOP resource requirements as follows:

$$10 + ((28 + 28) * 88 + 511) / 4096 = 11 \text{ VIOP memory pages}$$

$$7 + 2(28 + 28) = 119 \text{ VIOP windows}$$

In general, using more VIOP resources increases performance of the FDDI; however, you must be careful not to exceed available VIOP resources. The suggested minimum for `fd_max_recv` / `fd_max_xmit` is 12/12. Use of the maximum values, 128/64, could possibly cause the system to run out of VIOP memory pages.

We recommend that you use the default values for `fd_max_recv` and `fd_max_xmit`. You should tune these parameters only if greater performance is critical. If, after increasing either or both parameters, your system runs out of VIOP memory, either calculate a smaller set of values or revert to the defaults.

Verifying FDDI configuration with `netstat`

Bringing the machine up in multiuser mode is good evidence that you have correctly configured network interfaces. Usually, the machine will not run in multiuser mode if you make a mistake during the configuration process. You should test the network after the system is up and running in multiuser mode.

You can verify network configuration by entering

```
netstat -i
```

If the network is properly configured, the system displays output similar to that shown in Figure 13.

Figure 13 Checking FDDI configuration with `netstat -i`

```
Name    Mtu   Network    Address   Ipkts    Ierrs  Opkts    Oerrs    Collis
fddi0   4352  fddi2-net  attila    173031   0      150666   0        0
```

The displayed host name and network name are specific to your installation. The name of the interface you just configured should appear. If it does not, you have a problem with your installation or your configuration of the network. In either case, use the troubleshooting procedures outlined in Part 4, "Part 5 Testing and troubleshooting the network," page 329 to find the problem. The troubleshooting part also contains more complete instructions for using `netstat`.

Installing and configuring a HYPERchannel interface

5

Because the HYPERchannel interface supports Internet protocols (TCP/IP), you configure it in much the same way as you configure an Ethernet interface. However, HYPERchannel interface software has its own set of configuration prerequisites, as well as its own address resolution scheme. This chapter describes procedures used to install, configure, and test a HYPERchannel interface.

You can find supplemental information in the hy(4), tcp(4P), route(8C), and intro(4N) man pages, and in the following CONVEX Press documents:

- *CONVEX VMEbus HYPERchannel Controller Diagnostic Manual*
- *CONVEX VMEbus HYPERchannel Controller Service Guide*

Installing and configuring HYPERchannel hardware

This chapter assumes you have completed installation procedures that came with your ConvexOS software, and that you have installed CONVEX Internet Services software. In addition, before you can begin configuring HYPERchannel network software, you must complete the following hardware installation and configuration tasks:

- Install HYPERchannel hardware.
- Modify the /ioconfig file to integrate network hardware into ConvexOS.
- Test the hardware configuration.

Installing HYPERchannel hardware

CONVEX Internet Services provides a device driver or network interface (`hy`) for the Network Systems Corporation A400 or NB400 HYPERchannel Adapter through the IKON 10077-NSC Multibus or the IKON 10090 VMEbus Interface. To complete installation of network hardware, perform the following steps:

Step 1 Log in to the system as `root`.

Step 2 You must have an IKON 10077-NSC Multibus or IKON 10090 VMEbus Interface card installed in your system. Make sure the strappings are correct.

Note

On the IKON multibus board, output connector J1 connects to HYPERchannel adapter input. Input connector J2 connects to HYPERchannel adapter output. On the IKON VMEbus board, output connector J1 connects to the HYPERchannel adapter output. Input connector J2 connects to the HYPERchannel adapter input.

Step 3 Set the `timeout` value for the A400 Adapter to 78 by using thumbwheels on the back of the adapter. On the NB400, this setting is stored into EEPROM via the diagnostic console.

Step 4 Set the `unit` value into thumbwheels on the back of the A400 Adapter. Called the *adapter address*, this value either translates to a field in the internet address or is used in the `route` database. On the NB400, this setting is stored into EEPROM via the diagnostic console.

Note

Any change to the unit value on the thumbwheels requires you to reboot the system.

Configuring the VMEbus controller

The `csr` value is determined by controller switches U56, U57, and U58, as documented in the *IKON 10090 Hardware/Software Manual*. The interrupt level (`int`) is selected by jumpers W41-54 and W55-68. Other important controller switch settings are shown in Table 1.

Table 1 VMEbus controller switch settings

Controller switch	Setting
Address Modifier	3D
Bus Priority Level	3
DMA Burst Length	16
Range Counter Extension	16 bits wide
Device Flag	0
TEST ENB	off
SWAP DMA BYTES	on
SWAP P I/O BYTES	off

When you use configuration programs such as `ifconfig` and `route`, specify the device prefix as `hyper`. For example, to configure the device, `hostname-h`, enter the command

```
ifconfig hyper0 hostname-h up
```

Integrating HYPERchannel into ConvexOS

After installing network hardware, you must reconfigure ConvexOS to recognize the new device. Edit the /ioconfig file on the SPU to insert the device definition under the appropriate IOP and bus numbers. To edit the /ioconfig file for your system, first boot to the SPU prompt, `(spu)>`. Refer to the *CONVEX Processor Operation Guide* for booting instructions.

For a multibus HYPERchannel interface, insert an entry similar to:

```
iop 6
    mbus 0
        ctlr LAN-004 csr 0x5c0 int 5
            unit 0 type hy
```

Set the interrupt level (`int`) according to the IKON board's interrupt dipswitch setting.

A software-only release of the HYPERchannel driver for the IKON 10090 VMEbus Network Adapter Interface is available. The interface supports the NSC A400 and NB400 Adapters. Installation of this controller is similar to that of the multibus

controller. To integrate the VIOP HYPERchannel driver, add an entry similar to the following to the /ioconfig file:

```
viop 6
    vme 0
        ctlr LAN-204 csr 0x2000 int 4
            unit 0 type hy
```

Reboot the CONVEX system and bring it up in multiuser mode. This forces the system to autoconfigure the newly-installed network hardware.

Testing the hardware configuration

Bringing the machine up in multiuser mode indicates that you have correctly installed network hardware. Usually, I/O configuration fails and the system sends error messages to the /mnt/errlog file on the SPU if you make a mistake during the installation and configuration processes.

After modifying the /ioconfig file and rebooting the system, test the hardware configuration by performing the steps below.

Step 1 If the system does not `autoconfig` correctly (as indicated by a timeout on the device following the HYPERchannel adapter in the /ioconfig file), the I/O cables from the A400 Adapter to the IKON board may have been switched (that is, output connected to input and vice versa). Check connections on both the IKON board and the A400 Adapter.

Step 2 Bring up any network interfaces by running `ifconfig`. Until you run `ifconfig` to bring up a specific network interface (for example, `eth0`, `lo0`, or `hyper0`), you cannot display its status.

Step 3 Enter **`netstat -i`** to verify that the entry, `hyper0`, exists for the HYPERchannel interface. If **`netstat -i`** does not display an entry for the HYPERchannel interface, you will likely find that either the /ioconfig file is incorrect or the IKON board is not installed properly. Recheck the hardware installation and the /ioconfig file.

Configuring HYPERchannel software

Once you are confident that you have properly installed and configured network hardware, the next task is to configure network software.

Configuring HYPERchannel TCP/IP software is similar to configuring a TCP/IP Ethernet. Among the differences are the ability to specify the maximum transmission unit (MTU) for communication with remote hosts and the method used to map internet addresses to hardware addresses. If you wish to use a larger MTU, or if your HYPERchannel network includes internet addresses that do not conform to the structure shown in Figure 14, you must configure the software to use `route` after you assign host names and addresses. `route` is discussed in the section, "Mapping Internet Addresses to HYPERchannel Hardware Addresses."

To configure CONVEX HYPERchannel Interface software, you will perform the following tasks:

- Assign host names and addresses.
- Run `ifconfig` to configure the HYPERchannel network interface.
- Map internet addresses to HYPERchannel hardware addresses.

Assigning host names and Internet addresses

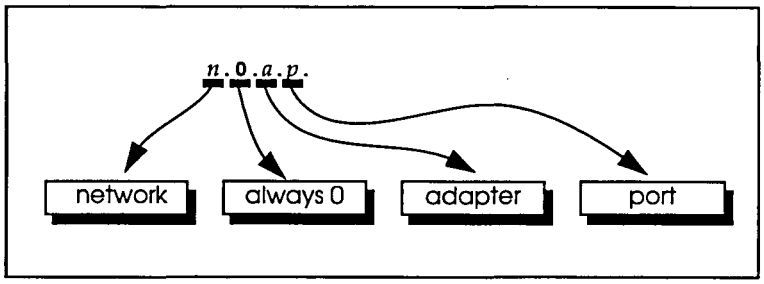
As with an Ethernet LAN, you can use either the host table lookup or name server method to assign HYPERchannel host names and associate them with internet addresses. The difference is in the format of the addresses themselves, not in the method of name resolution.

Regardless of which method you choose, read this section for an understanding of how HYPERchannel hardware addresses are structured. Then, if you choose to use the name server, turn to Chapter 15, "Configuring the name server," page 111, for a complete discussion on how to configure the name server.

If you choose the table-lookup method for associating host names and addresses, modify the /etc/hosts file to include hosts on the HYPERchannel network. Add an entry to the /etc/hosts file for each host connection on the HYPERchannel network.

Unless you structure your addresses as shown in Figure 14, you must use `route` to translate the internet addresses you assign into HYPERchannel hardware addresses.

Figure 14 HYPERchannel hardware address structure



Entries in the /etc/hosts file take one of the following forms:

n.0.*a*.*p*. *local_host_name*
n.0.*a*.*p*. *remote_host_name*

where:

n	Network number (1-63, 64-127, 128-255).
0	Second octet must be zero.
a	HYPERchannel adapter address, normally expressed in hexadecimal to match the `unit` value selected with thumbwheels on the back of the A400 Adapter to which the host is connected, or via the diagnostic console for an NB400 Adapter. (See Figure 14 for examples.)
p	HYPERchannel port number (0-3).
local_host_name	Name of the local CONVEX host.
remote_host_name	Name of a remote host.

Figure 15 shows typical /etc/hosts file entries for a CONVEX system with both an Ethernet and a HYPERchannel interface.

Figure 15 Typical hosts file with HYPERchannel interface

```
#
# Host Database
#
127.1          localhost
#
#
# The format is:
# internet-addressofficial-hostnamealiases...
#
# Ethernet
140.50.4       acme1    dopey
140.50.5       acme2    sleepy
140.50.6       acme3    sneezy
#
# HYPERchannel
#
128.0.0x12.1   acme4
128.0.0xc2.0   cray2
128.0.0x05.3   vax-1
#
```

You must also modify the /etc/networks file to include the address of the HYPERchannel network. To do so, add an entry similar to

netname 128

where *netname* is any name you assign to the HYPERchannel network. The network address you assign (for example, 128) must match the network address field of the HYPERchannel hosts identified in the /etc/hosts file.

Note

If your HYPERchannel addresses do not conform to the structure shown in Figure 14, or if you wish to change the MTU, you must configure the software to use `route` after you assign host names and addresses.

For more information about /etc/hosts and /etc/networks, refer to Chapter 10, "Setting up the host name database," page 69.

Configuring the HYPERchannel network interface

You define a HYPERchannel TCP/IP interface and customize network software the same way you define an Ethernet interface, by using the `ifconfig` command. Use a command similar to the following:

`ifconfig hyper0` *host_name* **`up`**

If the system responds with the error message, "Could not assign requested address," one of the following is probably true:

- The CONVEX system is not connected to the correct HYPERchannel adapter and port. Verify that the address you assigned to the interface matches the one selected with thumbwheels on the back of the A400 Adapter. If your system uses an NB400 Adapter, verify that the address you assigned to the interface matches the one stored in EEPROM. Correct the address if necessary.
- You assigned an incorrect network number to the HYPERchannel interface. Verify that the network number you entered is what you intended.
- HYPERchannel adapter hardware is malfunctioning. Consult the *NSC A400 HYPERchannel Adapter Hardware Reference Manual* or the *NSC NB400 HYPERchannel Adapter Hardware Reference Manual* for information about diagnosing the problem.

If `ifconfig` is successful, test the network configuration by entering

`/usr/etc/ping` *host_name*

You should get a series of messages that indicate data is being sent and received over the network. If not, one of the following may be true:

- *host_name* is not defined in the host name database.
- HYPERchannel adapter hardware is malfunctioning. Consult the *NSC A400 HYPERchannel Adapter Hardware Reference Manual* or the *NSC NB400 HYPERchannel Adapter Hardware Reference Manual* for information about diagnosing the problem.

Once you are confident that the network functions properly, add the `ifconfig` command to your /etc/rc.local script so that network software is configured each time the system starts. Then complete the software configuration by following the steps outlined in Chapter 7 through Chapter 11.

Mapping Internet addresses to HYPERchannel hardware addresses

You must configure network software to use `route` if your HYPERchannel host addresses do not conform to the structure shown in Figure 14, or if you wish to change the MTU.

`route` allows you to assign arbitrary internet addresses (that is, addresses that do not conform to the HYPERchannel hardware address structure), or to assign an MTU other than the default of 4096. For example, you could use `route` to map internet address `128.4.2.13` to `128.0.0x44.3` so that the host is assigned to adapter number 0x44, port 3.

HYPERchannel network software uses a default MTU of 4096 bytes. You can use `route` to change the MTU for particular routes.

To use `route`, follow these steps:

Step 1 Add commands to the /etc/rc.local script similar to

```
/etc/ifconfig hyper0 acme1 up
/etc/route addhyper host acme1 acme1 0 4096 2400 0000 0
```

Step 2 Modify the /etc/hosts file to include HYPERchannel host names and addresses. Because you are using `route`, these addresses do not need to conform to HYPERchannel hardware addresses. Figure 16 shows a typical /etc/hosts file for a HYPERchannel network.

Figure 16 hosts file configured for HYPERchannel

```
#
# Host Database
#
127.1            localhost
#
#
# The format is
#    internet-addressofficial-hostnamealiases...
#
128.50.0.0    acme1      # map to adapter 0x24, port 0 with route
128.50.0.1    acme2      # map to adapter 0x24, port 1 with route
128.50.0.2    acme3      # map to adapter 0x48, port 0 with route
98.50.0.1     acme4      # map to adapter 0xc2, port 3 with route
                         # (this computer is not directly accessible
                         #  because it's on a different network (98))
#
```

Step 3 Define host name to hardware address mappings. Figure 17 shows what the mappings might be for the hosts in Figure 16.

Figure 17 Host name to HYPERchannel address mapping in /etc/rc.local

```
# Add HYPERchannel routes
# Lines starting with  "#"  are comments
# The format is
# /etc/routeaddhyper [net|host] destinationgateway metric MTU  dest  control
access
/etc/route addhyper  host       acme1       acme1      0     4096  2400   0000       0
/etc/route addhyper  host       acme2       acme2      0     4096  2401   1100       0
/etc/route addhyper  host       acme3       acme3      0     163844801    2200       0
/etc/route addhyper  host       acme4       acme2      1     4096  2401   1100       0
/etc/route addhyper  host       acme4       acme3      1     163844801    2200       0
```

The entries in Figure 17 specify the following:

1. Any messages going to acme2 go out over trunk 1 and any messages coming from acme2 are received from trunk 1. This information is encoded in the high-order octet of the control field, where bits 7-4 encode transmit trunks 0-3, and bits 3-0 encode receive trunks 0-3.
2. acme3 maps to port 1 on adapter 0x48 instead of port 0 as specified in the /etc/hosts file in Figure 16. This entry also causes all messages to be transmitted over trunk 2 of the HYPERchannel adapter.
3. acme4 can be reached by going through either acme2 or acme3. The metric field shows that it is one hop away.

The access field allows the local host to send the message to the receiving adapter. This access code must match the receiving adapter's access code as selected on the thumbwheels on the back of the adapter. Generally, the access code is not used, but it still must be included in the 64-byte header of the HYPERchannel protocol.

Specify the MTU field in bytes. The actual MTU will be slightly larger than this.

Note

Because of the limited amount of buffer space available on the A400 Adapter, it is recommended that you do not set the MTU greater than 4096 for transfers between hosts directly connected to a single A400 Adapter.

If you need to change the mappings after the system has been booted, delete and readd the route with the `route` command. For example,

```
/etc/route delete host acme1 acme1
/etc/route/ addhyper host acme1 acme1 ...
```

Installing a non-TCP/IP device driver

You may install a device driver to communicate with networks that use protocols other than TCP/IP. Refer to the *CONVEX Guide to Writing Device Drivers* for information about how to write your own drivers.

To install the device driver, complete the steps listed below.

Step 1 Insert the following lines in the /ioconfig file on the SPU under the appropriate IOP and multibus entry:

```
ctlr LAN-002 csr 0x5c0 int 5
unit 0 type HYP_001
```

Step 2 Reboot the system to start autoconfiguration.

Step 3 Enter the command

```
mknod /dev/hyper0 c 15 0
```

to create a device entry in the operating system for the HYPERchannel device.

To use the port (`/dev/hyper0`), you must have the correct software. The HYPERchannel device driver can only send and receive messages that contain a 64-byte HYPERchannel header block. Refer to the *NSC A400 HYPERchannel Adapter Hardware Reference Manual* or the *NSC NB400 HYPERchannel Adapter Hardware Reference Manual* for specifications.

Solving installation problems

After completing software configuration, you should be able to use `rlogin`, `ftp`, or `telnet` to any host on the TCP/IP LAN. If you experience problems, use this section to help diagnose and solve them.

- Sometimes, the HYPERchannel adapter itself may be faulty. Try more than one adapter when you are having trouble getting a HYPERchannel link established.
- Make sure that the strappings on the IKON board are correct based on the most current documentation.

Configuring CONVEX HIPPI/TCP

6

When you add a network interface to your system or change characteristics of an existing one, you must integrate the device into ConvexOS. This chapter explains how to integrate a newly installed or reconfigured HIPPI/TCP interface into ConvexOS. Specifically, it explains how to

- Define the interface in /ioconfig, the system I/O configuration description file located on the SPU.
- Configure a STREAMS protocol stack on systems running ConvexOS V11.0 or later.
- Use the `ifconfig` command to assign an address to a network interface, configure interface parameters, and enable or disable the interface.
- Set up HIPPI/TCP routing.
- Tune network performance by modifying boot-time parameters.
- Use the `netstat` command to verify network configuration.

This chapter does not explain how to install network hardware; you must install the hardware before attempting any task described here. For complete instructions on installing and configuring CONVEX HIPPI hardware, refer to the following Convex Press documents:

- *CONVEX VIOP/VBCU Service Guide*
- *CONVEX HIPPI Service Guide*
- *CONVEX High Performance Parallel Interface Diagnostic Manual*
- *CONVEX HIPPI/TCP Release Notice*
- *CONVEX HIPPI/TCP Installation Procedure*

Refer to *Managing ConvexOS: Configuration Guide* for general information about configuring devices into ConvexOS.

Defining HIPPI/TCP in /ioconfig

Before ConvexOS can communicate over a network interface, it needs to know the physical location and characteristics of the hardware. To define a network interface's location and characteristics, you edit the /ioconfig file to include a description of the interface.

This section provides instructions for adding a HIPPI/TCP interface description to the /ioconfig file.

The /ioconfig file, located on the SPU disk, contains descriptions of all the Channel Control Units (CCUs), interfaces, controller boards, and peripheral devices on your system. When the system is booted, the boot process reads /ioconfig to determine the hardware configuration.

To display the /ioconfig file, enter the following command:

```
spu -r /ioconfig
```

Each entry in the /ioconfig file contains several levels of information, usually indented for readability. The interface type determines the content of the /ioconfig file entry. HIPPI/TCP entries in the /ioconfig file contain the following information:

```
hippitcp ccu-slot-number
   drvr LAN-205
      unit unit-number type HID-001
```

where

hippitcp *ccu-slot-number*
: Identifies the CCU type and slot number. Slot numbers range from 0 to 7, corresponding to the I/O slot to which the CCU is connected.

drvr LAN-502
: Identifies I/O driver type.

unit *unit-number*
: Units are numbered sequentially beginning with 0 for each controller type on a bus. The number of units supported by a controller depends on the controller type. Refer to the *CONVEX HIPPI Service Guide* for information about selecting this value.

type HID-001
: Identifies the type of unit connected to the controller.

After installing HIPPI/TCP hardware and booting the system to the SPU level, use the following procedure to define a HIPPI/TCP interface in the /ioconfig file.

Step 1 Log in to the SPU as the superuser.

Edit /ioconfig to include an entry for the HIPPI/TCP interface. Figure 18 shows an example /ioconfig file entry for CONVEX HIPPI/TCP.

Figure 18 CONVEX HIPPI/TCP /ioconfig entry

```
hippitcp 0
  drvr LAN-502
    unit 0 type HID-001
```

Step 2 Changes made to the /ioconfig file take effect when you reboot the system. If you need to perform other tasks that require a reboot to take effect, such as modifying network-related boot-time parameters or configuring a HIPPI/TCP STREAMS protocol stack, do so before rebooting. Refer to "Setting up HIPPI/TCP routing," page 54, for information about defining boot-time parameters. The next section explains how to configure a HIPPI/TCP STREAMS protocol stack.

Reboot the system by entering

```
/mnt/os/boot
```

Configuring a HIPPI/TCP STREAMS stack

On systems running ConvexOS V11.0 or later, Internet Services TCP/IP protocols (TCP, UDP, and IP) are implemented as a STREAMS-based architecture. When the system is booted, the boot process runs the `knetdctl` utility, which causes the STREAMS network daemon `knetd` to use information in the knetd.conf file to build the specified STREAMS stack.

The knetd.conf file included in the standard distribution provides a default configuration for the TCP/IP stack and one Ethernet interface stack. You need to modify this file to correctly build the HIPPI/TCP STREAMS protocol stack.

There are two ways to configure a HIPPI/TCP protocol stack in the knetd.conf file.

- Use the `io2knetcf` utility to generate the knetd.conf file from I/O descriptions found in the /ioconfig file.
- Manually edit the knetd.conf to add the appropriate entries.

Changes made to the knetd.conf file take effect when the system is rebooted.

The knetd.conf file installed with the software includes explanatory comments; you should read this file before making any changes to it. For information on knetd.conf file syntax, read the knetd.conf(5) man page.

This section explains how to define the STREAMS TCP/IP protocol stack in the knetd.conf file for a CONVEX HIPPI/TCP.

Generating the knetd.conf file from /ioconfig

The `knetdctl` utility is used to control the `knetd` daemon, which configures a STREAMS protocol stack from information provided in the /etc/knetd.conf file. The knetd.conf file included in the standard distribution provides a default configuration for the TCP/IP stack and one Ethernet interface. You need to modify this file to correctly build the HIPPI/TCP protocol stack.

You can use the `io2knetcf` utility to generate the /etc/knetd.conf file from information in the /ioconfig file. To use `io2knetcf`, enter

```
io2knetcf > /etc/knetd.conf
```

By default, `io2knetcf` uses the /ioconfig file as input and overwrites the existing knetd.conf file. You can specify alternate input and output files on the command line. For complete information on `io2knetcf`, refer to the io2knetcf(8) man page.

Manually defining a HIPPI/TCP stack

Follow the steps in this section to define CONVEX HIPPI/TCP in the knetd.conf file.

Step 1 In the Modules section, add the following line:

```
maci dc /dev/hippi
```

Step 2 In the Streams section, add the following statements for each HIPPI controller. Be sure to change the `maci:` number and `PORT={0}` appropriately for each controller.

```
maci:0 -          PORT={0}
ip     maci:0     PORT={2048, IP}
```

Changes made to the knetd.conf file take effect when the system is rebooted.

Configuring HIPPI/TCP with `ifconfig`

Before ConvexOS can communicate over a network interface, it needs to know the interface's network address and operating characteristics. This section explains how to use the `ifconfig` command to assign a network address to a HIPPI/TCP and set parameters that affect its operation.

Step 1 Log in as the superuser.

Step 2 Use `ifconfig` to define the interface's network address and operating characteristics.

You will need to add the `ifconfig` command to your rc.local file so that the interface is automatically configured at boot time; however, it is good practice to first test the command by entering it on the command line.

The `ifconfig` command has the following syntax:

`/etc/ifconfig` *interface host-name* {**`up`** | **`down`**} [**`arp`**] [**`trailers`**] [**`netmask`** *mask*] [**`broadcast`** *address*]

where

interface — A string composed of interface type and unit number (`hippi`*n*). Interfaces are numbered in the order in which they appear in the /ioconfig file. For example, the first HIPPI in /ioconfig is assigned the unit name, `hippi0`, the second, `hippi1`, and so forth.

host-name — Local host name or internet address in dot notation.

`up` — Enable the network interface.

`down` — Disable the interface. Used prior to reconfiguring certain parameters. For example, to change the interface's *mask*, you must disable the interface, define the new mask, and then enable the interface.

`netmask` *mask* — Specify the portion of the internet address to reserve for the combined network and subnet fields. (Refer to "Defining the subnet mask," page 53.)

`broadcast` *address* — Specify the address used to represent broadcasts to the network. By default, the broadcast address has a host part of all ones. (Refer to "Defining the broadcast address," page 52.)

Note

`ifconfig` has more parameters than those discussed here. Refer to the ifconfig(8C) man page for additional information.

Step 3 After testing the `ifconfig` command, verify that the system accepted the information by entering

`ifconfig` *interface*

In response, the system displays the network address and operating characteristics of the interface. Sample output is shown in Figure 19.

Figure 19 Using `ifconfig` to verify interface configuration

```
hippi0: flags=43<UP,BROADCAST,RUNNING>
        inet 130.150.70.3 netmask ffffff00 broadcast 130.150.70.255
        hardware address aa.00.04.00.2e.28
```

Step 4 If the network appears to be configured properly, edit your rc.local file to add the `ifconfig` command. This will ensure that the network interface is enabled every time the system boots. Figure 20 shows a partial rc.local file containing an `ifconfig` command.

Figure 20 Sample /etc/rc.local file

```
#
/bin/hostname moose
#
# build the networking streams stack (ConvexOS V11.0 or later)
#
if [ "`/etc/knetdctl -q`" = "knetd not configured" ]; then
        /etc/knetdctl -c /etc/knetd.conf
        /etc/knetdctl -r
fi

# configure network interfaces
#
if [ -f /etc/ifconfig ]; then
        /etc/ifconfig ex0 `/bin/hostname` up arp -trailers netmask 0xffffff00
fi
#
```

Defining the broadcast address

The default broadcast address contains a host part of all ones. `ifconfig` enables you to change an interface's broadcast

address. CONVEX systems accept broadcasts with a host part of all zeros (for compatibility with systems that use BSD 4.2 broadcasts), but transmits broadcasts with the destination address set to the broadcast address assigned with `ifconfig`.

If a machine on your network does not understand the broadcast address you select, some utilities, such as `rwho`, fail. If this happens, use `ifconfig` to set the broadcast address to the old broadcast address (all zeros) for all machines on the network.

Note

All machines that communicate with each other must use the same broadcast address, either all zeros or all ones. The preferred method is a broadcast address of all ones, as in broadcast 128.194.255.255.

Defining the subnet mask

Ones in the subnet mask indicate bit positions to use for the combined network and subnet fields; zeros mark the positions of bits in the host field. You specify the mask as a single hexadecimal number with a leading 0x, or in dot notation. For example, specifying

```
netmask 0xffffff00
```

or

```
netmask 255.255.255.0
```

both indicate that you want 24 bits of combined network and subnet fields, and 8 bits of host number. For a class B network, this mask logically partitions your 16 bits of host number into an 8-bit subnet field and an 8-bit host field. If you do not supply a `netmask`, the mask is set according to the network class (A, B, or C with 8, 16, or 24 bits of network part, respectively).

Note

To avoid confusion with broadcast addresses, do not use subnet numbers of all zeros or all ones.

Setting up HIPPI/TCP routing

If HIPPI/TCP is used for point-to-point connections, you do not need to use the `route` command. When a switch is involved, however, the `route` command will indicate how to get to each destination. The format of the `route` command for HIPPI/TCP is

```
route addhippi host dest gateway metric mtu ifield
```

where

dest Destination host

gateway Address of the host on which the HIPPI interface is installed

metric A count that indicates the number of hops to the destination

mtu The maximum transmission unit (65280)

ifield A 32–bit entity (I-field) that is defined in RFC 1374. Bits relevant to the `route` command are as follows:

Path selection (bits 26, 25) can be 00, 01, or 11 (binary). 00 (source route mode) indicates that the I-field bits 23-00 contain a 24–bit source route; 01 or 11 (logical address mode) indicate that bits 23-00 contain 12–bit source and destination Addresses. The value 11 is meaningful when more than one route exists from a source to a destination; it allows the switch to choose the route. Use of 01 forces the switch to always use the same route for the same source/destination pair.

If logical address mode is used, the source address field (bits 23-12) is not used, and the destination address field (bits 11-0) contains the switch address of the destination. If source route mode is used, the routing control field (bits 23-00) contains the route to the destination.

For example, assume host A and host B are connected to port 0 and port 1 of a switch, respectively. The path selection bits should be 00. The `route` commands for host A would look similar to the following:

```
route addhippi host addr_A addr_A 0 65280 0
route addhippi host addr_B addr_A 0 65280 1
```

If there was more than one switch between host A and host B, each port would be listed and 01 would be used for the path selection bits. Assume host A is on port 0 of switch 1, switch 2 is on port 1 of switch 1 and host B is on port 4 of switch 2. The route command from host A to host B would look similar to the following:

route addhippi host addr_B addr_A 1 65280 0x2000041

As the I-field is passed from switch 1, the routing control bits will be shifted right by 4 bits.

Tuning CONVEX HIPPI/TCP

When the system is booted, the initialization process reads a file containing parameters that affect the way ConvexOS handles CPUs and peripheral devices. Among other uses, these boot-time parameters enable you to optimize network performance.

Most network-related boot-time parameters apply to networking in general, rather than to a specific type of network interface; a few are interface-specific. This section describes boot-time parameters used to optimize HIPPI/TCP performance. For information about general networking parameters and instructions for modifying boot-time parameters, refer to Chapter 7, "Customizing boot-time parameters," page 57.

When you install CONVEX HIPPI/TCP, related boot-time parameters are set to their default values. By modifying these parameters, you can tune resource requirements to achieve a balance between performance and resource usage.

You tune HIPPI/TCP by manipulating the boot-time parameters, `hpi_recv_max`, `hpi_xmit_max`, and `str_msg_sz`, which are defined as follows:

`hpi_recv_max` Specifies the number of buffers used to hold input packets between the time they are received by the HIPPI/TCP driver and when they are passed to the IP layer for processing.

default = 50, min. = 32, max. = 100

`hpi_xmit_max` Specifies the number of buffers used to hold output packets between the time they are processed by the IP layer and when they are passed to the HIPPI/TCP driver. When the supply of available buffers is exhausted, the HIPPI/TCP driver ignores subsequent output packets passed to it from the IP layer.

default =50, min. = 32, max. = 100

`str_msg_sz` Specifies the maximum amount of data in a message in bytes. This value needs to be at least 64k for HIPPI/TCP. For the best

performance this number should be as high as possible. The install script will increase this tunable to 1048576 if necessary.

default =131072, min. = 16384, max. = 10485760

The values of these parameters determine how much memory the HIPPI/TCP driver will allocate for buffer space. Each buffer is 64k, large enough to hold the maximum packet size. If the values are set for 100 receive buffers and 100 send buffers the driver will allocate 200 * 64k bytes or 12.5 megabytes.

Verifying HIPPI/TCP configuration with `netstat`

Bringing the machine up in multiuser mode indicates that you have correctly configured network interfaces. Usually, the machine will not run in multiuser mode if you make a mistake during the configuration process. You should test the network after the system is up and running in multiuser mode.

To verify network configuration, enter

```
netstat -i
```

If the network is properly configured, the system displays output similar to that shown in Figure 21.

Figure 21 Checking HIPPI/TCP configuration with `netstat -i`

```
Name    Mtu   Network    Address   Ipkts    Ierrs  Opkts    Oerrs   Collis
hippi0  1500  acme-net   acmes     1520512  0      1327049  0       3
```

The displayed host name and network name are specific to your installation. The name of the interface you just configured should appear. If it does not, you have a problem with your installation or your configuration of the network. In either case, use the troubleshooting procedures outlined in Part 4, "Part 5 Testing and troubleshooting the network," page 329 to find the problem. The troubleshooting part also contains more complete instructions for using `netstat`.

Customizing boot-time parameters

7

When you boot your system, the boot process reads a file containing parameters that control the way ConvexOS handles CPUs and CCUs at your site. You can change these parameters to optimize performance and behavior of your system without recompiling the system image. This chapter describes each parameter, and explains how to modify boot-time parameters.

Locating boot-time parameters

Two system files contain boot-time parameters: the /mnt/os/bootcmd file and the /mnt/os/bootcmd.local file. Both files are located on the SPU disk.

The /mnt/os/bootcmd file is provided with your ConvexOS release tape. It contains information about how to boot your system, where the root partition resides, and commands that specify certain system parameters. You should not alter this file, as it is subject to change with each release of ConvexOS. Instead, you should use the /mnt/os/bootcmd.local file to set boot-time parameters specific to your site.

The /mnt/os/bootcmd.local file is not part of the ConvexOS release tape. You must create it to change the default values for boot-time parameters. Commands in /mnt/os/bootcmd.local take precedence over those in /mnt/os/bootcmd, allowing you to customize your system.

Modifying parameters

Follow this procedure to modify boot-time parameters:

Step 1 Log in as the superuser.

Step 2 If the bootcmd.local file already exists in the /mnt/os directory on the SPU, copy this file from the SPU to the /tmp directory on the CONVEX machine by entering

```
spu -r /mnt/os/bootcmd.local > /tmp/bootcmd.local
```

If bootcmd.local does not already exist, create a file named /tmp/bootcmd.local on the CONVEX system.

Step 3 Edit /tmp/bootcmd.local to change existing parameters or add new ones. Table 2 describes boot-time parameters used to tune network performance.

Boot-time parameters are set using the `tune` command. You can specify one parameter for each `tune` command, with each specification on a separate entry in the file. Entries have the following format:

tune *processor parameter=value*

where

processor The processor for which the parameter is specified. Each boot-time parameter affects only a single processor. Choices for processor include cpu, hsp, iop, and viop.

parameter The name of the parameter you are changing. Table 2 lists networking boot-time parameters.

value A value assigned to the parameter. This value must be within the minimum and maximum values. Default values, as well as the minimum and maximum values for each parameter are listed in Table 2.

An example of the contents of the bootcmd.local file is shown in Figure 22.

Figure 22 Example bootcmd.local

```
tune cpu hpi_recv_max        = 50
tune cpu hpi_xmit_max        = 50
tune cpu str_msg_sz          = 131072
```

Step 4 Copy the modified bootcmd.local file back to the SPU by entering

```
spu -w /mnt/os/bootcmd.local < /tmp/bootcmd.local
```

Step 5 Reboot the system.

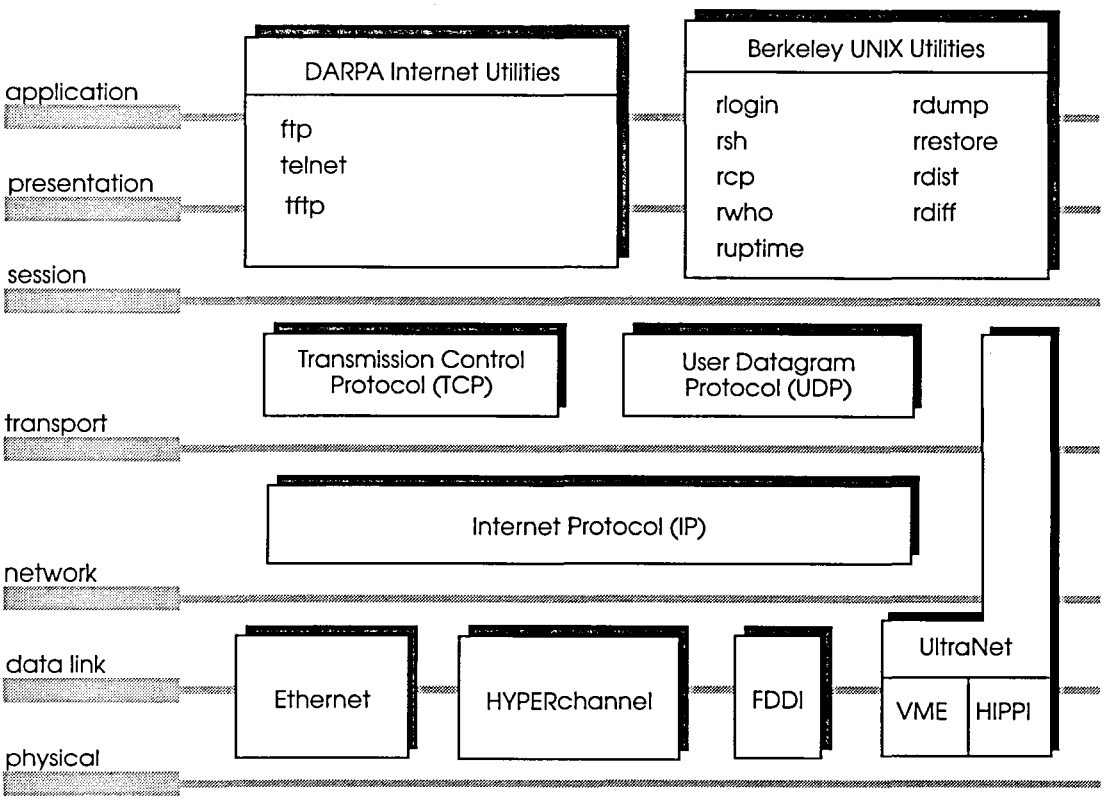
Table 2 Networking boot-time parameters

Parameter	Description
`fd_max_recv`	Specifies the number of buffers used to hold input packets between the time they are received by the FDDI driver and when they are passed to the IP layer for processing. default = 28, min. = 2, max. = 128
`fd_max_xmit`	Specifies the number of buffers used to hold output packets between the time they are processed by the IP layer and when they are passed to the FDDI driver. When the supply of available buffers is exhausted, the FDDI driver ignores output packets passed to it from the IP layer. default =28, min. = 4, max. = 64
`gateway`	Enables sending Internet Control Message Protocol (ICMP) errors if both of the following conditions are true: • The system has a single network interface, or IP forwarding is disabled (see also the `ipforwarding` parameter). • The received IP packet is not for the system that has `gateway` enabled. If `gateway` is disabled under these circumstances, errors are not sent to the source host, and the packet is dropped. See also the `ipforwarding`, `ipsendredirects`, and `subnetsarelocal` parameters. default = disabled, disabled = 0, enabled = 1
`hpi_recv_max`	Specifies the number of buffers used to hold input packets between the time they are received by the HIPPI/TCP driver and when they are passed to the IP layer for processing. default = 50, min. = 32, max. = 100
`hpi_xmit_max`	Specifies the number of buffers used to hold output packets between the time they are processed by the IP layer and when they are passed to the HIPPI/TCP driver. When the supply of available buffers is exhausted, the HIPPI/TCP driver ignores subsequent output packets passed to it from the IP layer. default =50, min. = 32, max. = 100

Table 2 Networking boot-time parameters **(continued)**

Parameter	Description
`ipforwarding`	Enables Internet Protocol (IP) packet forwarding. Packets are forwarded when the IP address does not correspond to any of the Internet addresses for the host's network interfaces. If this parameter is disabled, packets can be dropped. See also the `ipsendredirects`, `gateway`, and `subnetsarelocal` parameters. default = enabled, disabled = 0, enabled = 1
`ipsendredirects`	Enables sending Internet Control Message Protocol (ICMP) redirects. Redirects inform the sending host of the direct address of the host to which packets were forwarded. If this option is disabled, redirects are not sent when packets are forwarded. See `ipforwarding`, `gateway`, and `subnetsarelocal` parameters. default = enabled, off = 0, enabled = 1
`str_msg_sz`	Specifies the maximum number of bytes of data in a message. This value needs to be at least 64k for HIPPI/TCP. For the best performance, this number should be as high as possible. The install script will increase this tunable to 1048576 if necessary. default =131072, min. = 16384, max. = 10485760
`subnetsarelocal`	Considers an Internet address local even if it belongs to another subnet. A different network number means the address is remote. This option is used only during determination of the TCP maximum segment size. TCP uses a small maximum segment size (536 bytes) for remote destinations. For local destinations, TCP uses the maximum transmission unit of the outgoing network interface. See the `ipforwarding`, `ipsendredirects`, and `gateway` parameters. default = 1, off = 0, on = 1
`updcksum`	Enables checksumming of UDP datagrams. UDP checksumming incurs substantial overhead because each transmitted and received UDP datagram is checksummed. Turning the parameter off increases the risk of allowing bad packets farther up in the protocols. default = 0, off = 0, on = 1
`viop_enet_proc`	Sets the number of send and receive processes for a particular VIOP Ethernet interface. This parameter is used to reduce the amount of memory needed by the VIOP when servicing multiple Ethernet interfaces. While lowering the number of processes helps the VIOP avoid running out of memory, it also lowers network performance, so you should decrease the number of processes only if your system fails to boot because of insufficient memory. default = 4 processes per Ethernet interface

Part 3 Configuring Internet Services



Introduction

8

This chapter lists prerequisites to configuring CONVEX Internet Services software and presents an overview of tasks used to configure and test the network.

This part provides instructions for configuring and managing Internet Services; it does not explain how to install the hardware or software. Before beginning any procedure described in this part, make sure you have completed all the prerequisite hardware and software installation and configuration tasks described below.

Hardware prerequisites

Before beginning network configuration, make sure that you can answer "yes" to the following questions:

1. Has the hardware (cables, controllers, adapters) been completely installed and tested?
2. Have you run diagnostics on the SPU? In particular, if you have installed an Ethernet, has the Ethernet Controller Functional Test (`dev4500`) run successfully?

   Instructions for running diagnostic tests are included in the diagnostic manual for each type of network interface. Each chapter in Part 2, "Part 2 Configuring network interfaces," page 7, includes a list of appropriate hardware service guides and diagnostics manuals.
3. Have you started the system in multiuser mode without problems?

Software prerequisites

Before attempting any procedure described in this part, you must successfully complete the following tasks, in the order in which they are listed:

- Install ConvexOS and Utilities
- Install CONVEX Internet Services

If you intend to use NFS and NIS services, you should also install CONVEX NFS before beginning network configuration.

For information about installing the products listed above, refer to the installation procedures that accompanied the software distribution tape.

Task summary

Network management components include commands, daemons, and system files. You use these commands to define network interfaces, assign host names and addresses, test the network, and set up routing; you configure daemons to service network communication requests; and you modify system files to reflect network configuration.

To configure CONVEX Internet Services, you must complete the following tasks, in the order in which they are listed:

- Integrate network hardware into ConvexOS by editing the /ioconfig file on the SPU.
- Install network software according to the installation procedure shipped with the distribution tape.
- Customize boot-time parameters for your site and type of network interface.
- Choose local host names and internet addresses and set up the host name database.
- Configure network interfaces with the `ifconfig` command.
- Optionally, configure Serial Line Internet Protocol (SLIP).
- Configure and start the internet superserver daemon, `inetd`.
- Set up and enable routing.
- Optionally, configure the Berkeley Internet Name Domain server (BIND).
- Test the configured network and troubleshoot problems if necessary.

Subsequent chapters describe these tasks in detail.

Where to find more information

This guide focuses on the "how to" of TCP/IP system management—the tasks that you, as the system manager, perform to set up, maintain, and troubleshoot Internet Services. This book does not explain fundamental networking concepts and theories, or the design and implementation of the software. It also does not attempt to provide you with all the information you may need to help you decide whether or not to configure a particular service on a particular host.

Because TCP/IP networking is a mature technology, you can find information about the "real-world" uses for and inner workings of TCP/IP network services in the technical section of your local book store. Anyone responsible for managing a TCP/IP network should acquire and peruse at least one of these off-the-shelf books.

We recommend the following books:

- *TCP/IP Network Administration*, by Craig Hunt (O'Reilly & Associates, 1992. ISBN 0-937175-82-X).
- *DNS and BIND in a Nutshell*, by Paul Albitz and Cricket Liu (O'Reilly & Associates, 1992. ISBN 1-56592-010-4).

You can find a concise introduction to basic networking concepts and terminology, an overview of CONVEX networking products, and a description of the CONVEX implementation of Internet Services in *CONVEX Networking Concepts*.

Controlling access to the network

9

This chapter describes how to control users' access to network services.

Each machine on the network can accept or refuse logins by remote users. The system manager specifies whether users logging in from other machines must supply a password, or even if remote logins are permitted on a machine. Two files control this type of access: */etc/hosts.equiv* and *.rhosts*. (If your system runs NFS, the Network Information System (NIS) uses the /etc/netgroup file to control user access. NFS is an optional product. For information about configuring NIS, refer to Part 3, "Part 3 Configuring Internet Services" page 61.)

The hosts.equiv file

The /etc/hosts.equiv file contains a list of machine names that you can access from the local machine without using a password (using the same login name on the remote machines). When you use `rlogin` to remotely log in to a machine that is not listed in the /etc/hosts.equiv file, you are prompted for a password unless the local machine is listed in the ~/.rhosts file. Figure 23 shows a sample /etc/hosts.equiv file.

Figure 23 Sample hosts.equiv file

```
acme1
dopey
sleepy
sneezy
doc
bashful
grumpy
```

Note

For security reasons, access to the root account is not controlled by the /etc/hosts.equiv file. You must either have an appropriate entry in root's /.rhosts file, or you must supply the root password when you log in remotely. For security reasons, it is suggested that read access to the /etc/hosts.equiv file be limited to root.

The .rhosts file

Each user's home directory may also contain a private equivalence list in a file called .rhosts for use by `rlogin`. Entries in this file contain remote host names (not aliases) and user names specifying which users on which machines are permitted to log in without supplying passwords. For example, if you use different login names on different machines, add entries to your .rhosts file containing your other login names and names of machines from which you want to access your local account. The example in Figure 24 shows a .rhosts file for a user named `rsmith` on the local machine who uses login names `smith` and `bobsmith` on remote machines.

Figure 24 Sample .rhosts file

```
aries smith
aries bobsmith
```

In the above example, both `smith` and `bobsmith` may access `rsmith's` account on `aries` without entering a password.

Note

Your .rhosts file will not be honored if it has either group or world write permissions.

You must use official host names (the first name listed in /etc/hosts file entries), not aliases, in the .rhosts file.

Setting up the host name database

10

This chapter describes how to set up a host name database for your local network. It contains three sections:

- An introduction to host naming and addressing concepts, as well as descriptions of ways to represent host names and addresses.
- A discussion on choosing an address structure.
- A step-by-step guide to creating the host name database.

Readers familiar with internet naming and addressing may want to skip to the section,"Creating the host name database" on page 78, which guides you through steps for implementing an address structure.

Identifying hosts on a network

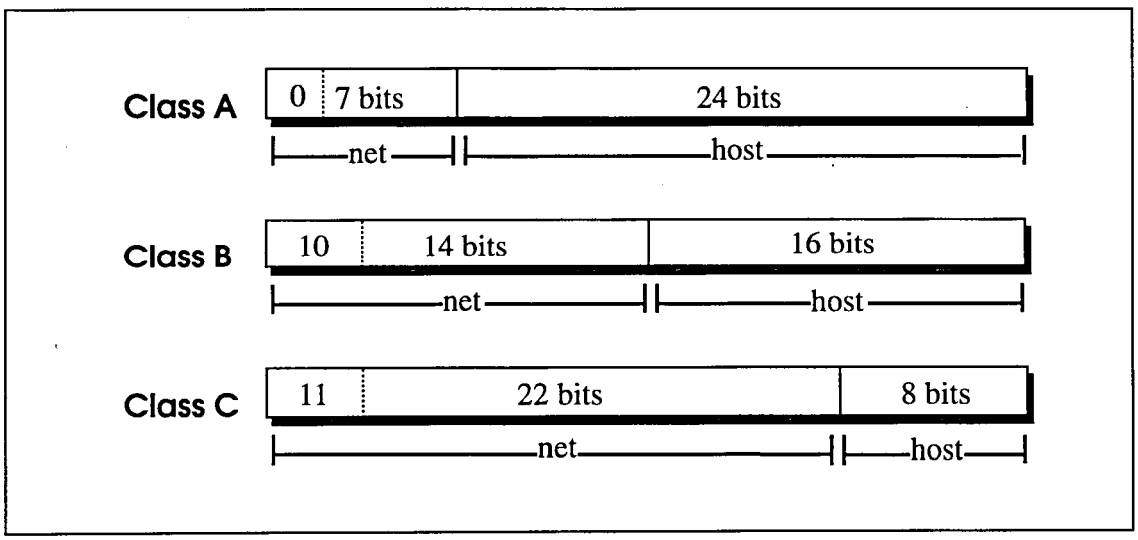
Internet addresses, also known as *network addresses* or *host addresses*, are numeric, universal identifiers assigned to machines on a TCP/IP network. While software works well with these compact addresses, people find them cumbersome and difficult to remember. *Host names* also identify machines on a network, but in a text form most people find easier to work with. Network software translates host names to internet addresses before using them internally.

Internal representation of Internet addresses

Internet addresses are expressed as 32-bit values representing two fields: *network number*, which includes *address class*, and *host number*. Address class determines the size (often called *address space* or *name space*) of the network in terms of how many hosts it can support. Network number identifies the interface used by the host to communicate on the network. Host number refers to a specific machine on the network interface.

As Figure 25 illustrates, distribution of the 32 bits of address depends upon the address class defined in the high-order bits.

Figure 25 Internet addresses by class



Users typically designate hosts by name, and need not work with addresses. When configuring the network, however, the system manager must work with both names and addresses.

Dot notation

In situations where you identify hosts or networks by address, as in the /etc/hosts and /etc/networks files, you normally use *dot notation*. In dot notation, you specify an address as a series of decimal numbers (usually four) separated by periods, as in

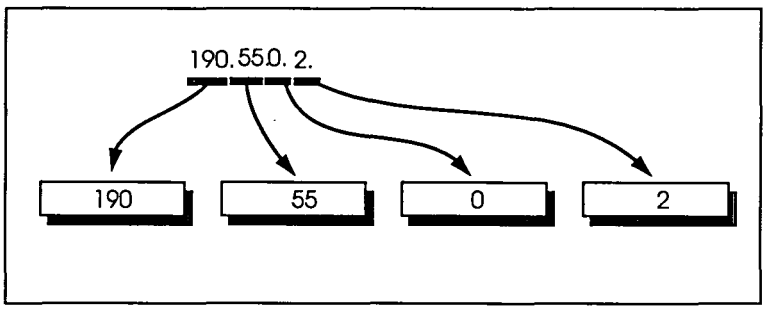
```
128.50.10.1
```

Note

In addition to dot notation, you can specify internet addresses as decimal, octal, or hexadecimal numbers. Use a leading 0x or 0X to signify a hexadecimal address, and a leading 0 to signify an octal address. Numbers without prefixes are interpreted as decimal.

When the address is specified in four parts, each 8-bit value is encoded into the corresponding octet of the internet address. Figure 26 illustrates this correspondence.

Figure 26 Four-part dot notation address



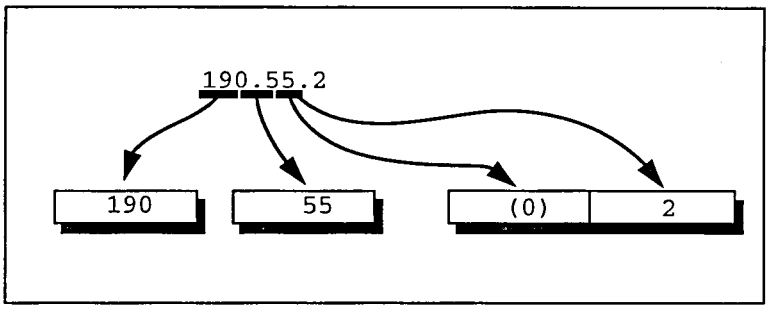
Address `190.55.0.2` has binary value `10` in its address class to show that it is a class B address. Network software interprets the first two octets of a class B address as the network number and the second two octets as the host number.

Three-part dot notation conveniently represents class B host addresses in the form

```
net.net.host
```

Figure 27 shows how host address, `190.55.2`, is encoded into a four-octet internet address.

Figure 27 Three-part dot notation address



As you can see from the previous two figures, internal representation of host address `190.55.2` is equivalent to that of host address `190.55.0.2`.

Note

If you specify a network address in three-part dot notation, as in /etc/networks, the address will be left-justified in four octets. For example, for network address 190.50.5, the first octet receives 190, the second receives 50, the third receives 5, and the rightmost octet is set to zero.

You can also use three-part dot notation to distinguish subnet numbers from network numbers. Subnet addresses are discussed in the section, "Dividing your address space Into subnets," page 75.

Reserved addresses

Certain addresses have special meanings to network software and to other hosts on the network.

Broadcast addresses

By default, a host number field of all ones signifies a *broadcast* address. A broadcast is a message sent to all hosts on the network. For example, networked hosts share routing information with each other by broadcasting their routing tables.

Note

Because some network interfaces, for example, HYPERchannel, do not support broadcasts, using a broadcast address does not guarantee that broadcasts will actually take place.

Network addresses

An internet address with a host number of all zeros refers to the network itself, rather than to a particular host. Class A network

addresses are normally expressed in one-part dot notation. Class B network addresses are normally expressed in two-part dot notation, as in `140.55`. Class C network addresses are normally written in three parts, as in `140.55.20`, with each decimal value mapped to an octet of the network field; the host number field is set to zero.

Host naming conventions

To keep matters simple, this discussion of names and addresses has implied that a host address identifies a unique computer. This is only partially true, because a host can communicate over more than one network interface. In such cases, a single address is not enough.

Termed *multihomed hosts*, machines connected to multiple network interfaces have multiple addresses, one per interface. This situation makes it apparent that an internet address does not actually refer to a host computer, but to a connection—a pathname that specifies a particular network interface on a particular machine.

Naming conventions help minimize confusion caused by multihomed hosts. One such convention consists of a root name identifying the host computer (for example, `hamlet`) followed by suffixes for the network connection (for example, `hamlet-ex` for accessing `hamlet` over an Ethernet; and `hamlet-hy` for accessing it via a HYPERchannel connection). This naming convention is illustrated in Figure 28.

Figure 28 Naming hosts with multiple network connections

```
#
# Host Database
#
127.0.0.1     localhost
#
# Local Ethernet
#
190.55.10.2     ariel-ex air
190.55.10.3     puck-ex
190.55.11.1     hamlet-ex son
#
# HYPERchannel
#
190.56.10.2     ariel-hy  wind
190.56.11.1     hamlet-hy pop
#
```

Unfortunately, this naming convention sometimes results in names that are not user-friendly. To compensate for this awkwardness, assign aliases that are both meaningful and friendly, as in the example above.

Choosing an address structure

The first step in configuring your network is to select names and addresses for hosts on your LAN.

If you were setting up a LAN that you did not want to connect to any other network, you could pick addresses at random or invent your own numbering scheme. However, if you choose addresses randomly and later, after your small, simple network has grown in size and complexity, you find that you want to connect to other networks, you face the difficult task of changing your address structure to conform to the expectations of external networks. For this and other reasons, you should consider several factors before assigning addresses to hosts on your LAN. Among these factors are

- Whether your LAN will connect to external networks.
- The number of host addresses you need, with room for expansion.
- Whether you will divide your network address space into subnets.

Access to other networks

Internet protocols require each host to have a unique address. Unless you use "official" internet addresses, you cannot guarantee uniqueness. Therefore, it pays to obtain official addresses before you set up your network. To do so costs nothing, and it permits you to expand the service area of your network as your needs grow.

After you have determined which address class you need, your site administrator should contact the organization in charge of the network and request the appropriate domain registration form. An organization that connects to multiple networks (such as CSNET, DARPA Internet and BITNET) should register with only one network. Contacts are as follows:

- **DARPA Internet**—Sites that are already on the DARPA Internet and need information about setting up a domain should contact `HOSTMASTER@NIC.DDN.MIL`. You can do so by writing to DDN Network Information Center, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025, or by sending `email` to the network address below.

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

bind-request@ucbarpa.Berkeley.EDU.

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

An organization that already has a registered domain name should keep the CIC informed about how it would like its mail routed. In general, the CSNET relay prefers to send mail via CSNET (as opposed to BITNET or the Internet) if possible. For an organization on multiple networks, this may not always be the preferred method. The CIC can be reached via electronic mail at `cic@sh.cs.net`, or by phone at (617) 873-2777.

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

Determining address space requirements

The potential size of your network influences how you choose addresses. Because your choice of address class imposes a hard limit on address space, you should choose a class generous enough to meet your needs as your network grows. Few networks require class A addresses (16,777,214 hosts), but many need a larger address space than the 254 hosts available on class C networks (host numbers of all zeros and all ones are reserved, as discussed above, under "Reserved Addresses"). Choosing an address class is a relatively easy decision to make before the network is set up and a difficult one to change later.

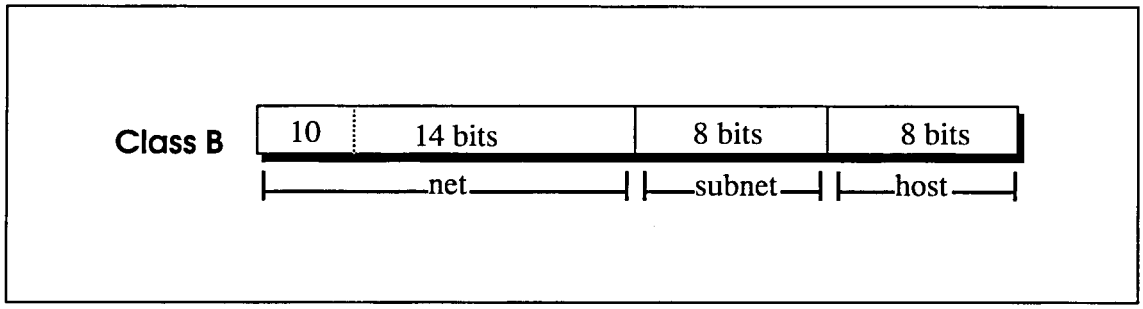
Because the total number of available network numbers in each class is limited (127 class A; 16,383 class B; and 4,194,303 class C), you should choose the largest class that can support your present needs and your expected growth.

Dividing your address space into subnets

Before you assign addresses, you should decide whether you want to use *subnetting*. Subnetting is the partitioning of the network address space defined by a single network number into multiple virtual networks called *subnets* or *subnetworks*. With subnetting, local networks can include multiple physical

network interfaces, yet appear as a single entity to remote networks. Figure 29 illustrates a typical way in which a class B network address is partitioned into subnet addresses.

Figure 29 Subnet address partitioning



While this division into 8 bits for subnet number and 8 bits for host number is common for class B networks, you can partition the address any way you choose. The number of bits you give each field depends upon your needs. Will you have just a few large subnets, or is your installation more suited to having many small subnets?

Subnetted class B host addresses are commonly written in four-part dot notation as

```
net.net.subnet.host
```

When setting up subnets, assign addresses to your subnets as well as to the hosts they serve. Three-part dot notation conveniently represents a subnet address as

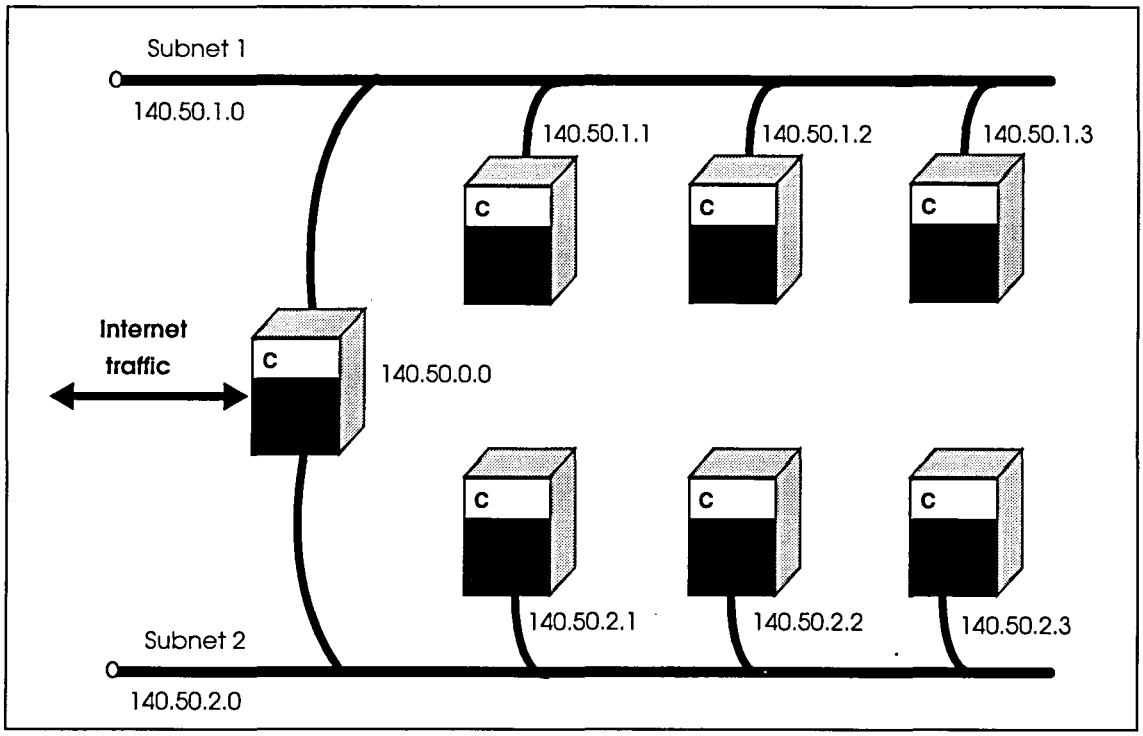
```
net.net.subnet
```

You might choose to logically divide an Ethernet LAN into subnets so that you can switch to multiple cables relatively easily if the network grows beyond the capacity of a single Ethernet cable.

More commonly, subnets correspond to separate physical networks. For example, an installation may use one Ethernet cable to connect machines used by engineering, another to serve the accounting department's machines, another for marketing, and so forth, each one assigned its own subnet number. A single gateway, assigned a single network number, could then be used for communication between hosts on different subnets, as well as between hosts on all subnets and the rest of the internet. (Gateways are discussed further in Chapter 12, "Setting up routing," page 95.")

Figure 30 depicts one possible use of subnets.

Figure 30 Subnets connected to an internet by a gateway



You could achieve the same logical partitioning of a large local network by assigning different network numbers to each department's LAN. However, this method has disadvantages. Take the situation where a network consists of several physical LANs. You could obtain one network number and partition it into subnets, or you could request a different network number for each LAN.

Though the decision to use subnets or multiple network numbers is yours to make, be aware that it has an impact on systems beyond your own. Each address class has a fixed number of networks it can support, limited by the number of bits in the network number field. If you use multiple network addresses, you may unnecessarily deplete the pool of network numbers.

Once you have made the decision to divide your address space into subnets, you must tell network software which bits to use as the network and subnet numbers. You do this with the `netmask` parameter, when you configure the network interface with `ifconfig`.

Methods for partitioning address space into subnets are discussed later in this chapter. Chapter 3, "Configuring an Ethernet interface," page 11, explains how to set the subnet mask with `ifconfig`.

Other considerations

How you structure your addresses also affects routing, because routing decisions are based on the network number portion of the address. All hosts on the same LAN should have the same network number.

Creating the host name database

Previous sections described ways in which hosts on a TCP/IP network are identified and discussed points to consider before deciding upon an address structure.

After choosing an address structure, create the host name database for your name space and enable a method of associating host names with addresses. Host names are associated with internet addresses through a mechanism called *name resolution*. Network software uses one of two methods to resolve host names:

- Host table lookup method, whereby network software uses a static host name database containing explicit name-to-address translations. If you choose to use the host table lookup method to resolve names, your primary task is to maintain the /etc/hosts and /etc/networks files. This method requires you to update the /etc/hosts file on every networked host each time you add, remove, or change its address. Consequently, this method is practical only when your network is relatively small and stable.
- Name server method, whereby dynamic host name and address information is shared across the network. CONVEX Internet Services provides the Berkeley Internet Name Domain (BIND) server system, consisting of a name server and an address resolver. When you use the name server method, you need only set up the host name database on a single host and configure other machines on the LAN to send queries to that host.

Regardless of which method you choose, you must create and maintain the /etc/hosts and /etc/networks files on at least one host on your LAN. The host name database must include all machines on your network.

To use the name server, you must also configure hosts on your LAN as described in Chapter 15, "Configuring the name server," page 111.

Modifying the /etc/hosts file

The /etc/hosts file contains the database of host names and addresses for all hosts on your local network, as well as hosts on networks interconnected with yours.

The first few lines of the /etc/hosts file distributed with CONVEX Internet Services are similar to those in Figure 31.

Figure 31 Sample /etc/hosts file

```
#
# Host Database
#
127.0.0.1    localhost
#
#
# The format is:
#    Internet-addressofficial-hostnamealiases...
#
# Local Net -- Ethernet
#
140.50.0.4      acme1 dopey
140.50.0.5      acme2 sleepy
140.50.0.6      acme3 sneezy
140.50.0.7      acme4 doc
140.50.0.9      acme5 bashful
140.50.0.10     acme6 happy cad
#
```

The /etc/hosts file contains two types of data: dot-notation addresses (`140.50.0.9`) and logical host names (`acme5 and bashful`).

The first number of the addresses inFigure 31 is between 128 and 191, signifying that the hosts belong to a class B network. This means that the 32-bit internet address is divided into 16 bits of network number and 16 bits of host number.

Note

In addition to dot notation, you can use decimal, octal, or hexadecimal numbers to specify internet addresses. Use a leading 0x or 0X to signify a hexadecimal address, and a leading 0 to signify an octal address. Numbers without prefixes are interpreted as decimal.

Host names in Figure 31 are specified two ways: official host name and aliases. The official host name is listed first; aliases, if any, follow on the same line. For example, in the entry:

```
acme6 happy cad
```

`acme6` is the official name, and `happy` and `cad` are aliases. Aliases enable you to reference hosts by symbolic names. For example, the command sequences `rlogin cad` and `rlogin acme6` connect you to the same machine.

The version of /etc/hosts initially installed on your system is large, containing many groups of host names. It is intended to serve as an example and to provide addresses of hosts you may want to access if you connect to the DARPA Internet. Because most of these host names have little to do with names and addresses you choose for hosts on your LAN, feel free to delete those you do not need. Then, modify /etc/hosts to reflect your own network configuration.

Add addresses, names, and aliases for each host to which your hosts need access. Internet Requests for Comments (RFCs) require all networks to register with NIC.DDN.MIL. However, if your LAN is not connected to external networks, you are free to choose any network and host numbers; just make sure that each host on any given network has the same network number and a unique host number.

Note

If an official host name appears on more than one entry in the /etc/hosts file, the system uses only the first name. For example, if the /etc/hosts file contains the entries

```
140.50.0.4    acme1
140.50.1.4    acme1
```

networking software will send messages destined for `acme1` to Internet address 140.50.0.4.

An example of a customized host database is shown in Figure 32. The file contains names and addresses for machines on two LANs.

Figure 32 Customized /etc/hosts file

```
#
# Host Database
#
127.0.0.1  localhost
#
# local Ethernet; network number 140.50
#
140.50.0.1  acmeq   obewan    acmeq-ex
140.50.0.2  acmer   vader     acmer-ex
140.50.0.3  acmes   leia      acmes-ex
140.50.0.4  acmet   luke      acmet-ex
#
# local Ethernet; network number 140.51
#
140.51.0.1  acmea   jason     acmea-ex
140.51.0.2  acmeb   medea     acmeb-ex
140.51.0.3  acmec   argos     acmec-ex
140.51.0.4  acmed   athens    acmed-ex
#
```

Host names shown in Figure 32 consist of a machine name (for example, `acmeq`), followed by two aliases (`obewan` and `acmeq-ex`). The second alias has an appended interface designator (`-ex`). Though you are not required to adhere to this naming convention, you may find it useful if you have multiple network interfaces, because it serves to identify both the machine and the interface.

Read more about the /etc/hosts file in Appendix A of this manual and in the hosts(5) man page.

Note

The /etc/hosts file will probably be identical for all hosts on the local network. If this is the case on your network, you can create the /etc/hosts file once and transfer it to all the other hosts.

Modifying the /etc/networks file

Just as the /etc/hosts file contains logical names, addresses, and aliases for each host to which your network is connected, the /etc/networks file contains logical names, addresses, and aliases for each network with which your LAN communicates. Several network utilities, including `netstat` and `routed`, use

this file for mapping between network numbers and names. A typical /etc/networks file is shown in Figure 33.

Figure 33 Sample /etc/networks file

```
#
# Network Database
#
# The format is:
#   network-name Internet-address aliases...
#
acme-ex0        140.50   starnet
acme-ex1        140.51   greeknet
loopback-net    127      localnet
#
```

Network numbers shown in the example correspond to network numbers given in the sample /etc/hosts file in Figure 32.

Modify the /etc/networks file to include a logical name and network number for each network to be referenced from your machine. Be sure to use the same network numbers that you used in the /etc/hosts file. If your LAN has connections to other large networks such as the DARPA Internet, you must use officially assigned network numbers.

Read more about the /etc/networks file in Appendix A of this manual and in the networks(5) man page.

Creating subnets

Subnetting allows interconnected local networks to share a single network number, thus reducing the demand on the available pool of network numbers and simplifying routing for external hosts and gateways. With subnets, local networks can include multiple physical network interfaces, yet appear as a single entity to remote networks.

To implement subnets, divide the host number field of your LAN's internet address into subnet and host portions by specifying a *subnet mask* with the `ifconfig` command used to configure the network interface. The subnet mask determines which part of the 32-bit internet address is combined with the network number to identify the subnet. By default, the mask is set to the size of the normal network number field for the particular network class (A, B, or C with 8, 16, or 24 bits of network part, respectively).

Within a subnetted local network, the subnet field is considered part of the network number. Outside the local network, the subnet field is not included in the network number.

Set the subnet mask at boot time in /etc/rc.local with `ifconfig`. Chapter 3, "Configuring an Ethernet interface," page 11, explains how to use `ifconfig` for this purpose. The /etc/networks and /etc/hosts files in Figure 34 illustrate how a network number is logically partitioned into three subnets.

Figure 34 Subnet partitioning in the /etc/networks and /etc/hosts files

```
#
# Network Database (/etc/networks)
#
# The format is:
#network-name Internet-address aliases. . .
#
# Class B Network - 140.50.
# subnet mask - 255.255.255.0 (0xffffff00)
# 255 subnets with 255 hosts available
#
acme-ex0        140.50.100    fishnet
acme-ex1        140.50.200    astronet
acme-ex2        140.50.300    faulknet
#
loopback-net    127           localnet
#


//////////////////////////////////////////////////////////////////
# Host Database (/etc/hosts)
#
# The format is:
#               Internet-address official-hostname aliases. . .
#
127.1           localhost
#
# Engineering Dept. (fishnet 140.50.100.hh)
#
140.50.100.10   eng1a         carp
140.50.100.20   eng1b         tuna
140.50.100.30   eng1c         sushi         squishy
140.50.100.40   eng1d         squid         inky
#
# Marketing Dept. (astronet 140.50.200.hh)
#
140.50.200.10   eng2a         altair
140.50.200.20   eng2b         sirius
140.50.200.30   eng2c         betelgeuse    beetle
#
# Documentation Dept. (faulknet 140.68.300.hh)
#
140.50.300.10   doc1a         caddy
140.50.300.20   doc1b         benjy
140.50.300.30   doc1c         jason
140.50.300.40   doc1d         quentin
#
```

Regenerating /etc/hosts and /etc/networks files

Internet host name databases are normally derived from a file retrieved from the Internet Network Information Center at SRI. The *gettable* program retrieves the NIC host database, and the *htable* program converts the data to the format library routines use to resolve addresses. Both programs are located in the /usr/etc directory.

To retrieve and reformat the internet host database table, change to the directory where you maintain local additions to the host table and execute the commands shown in Figure 35.

Figure 35 Using `gettable` and `htable` commands

```
# /usr/etc/gettable nic.ddn.mil
Connection to sri-nic.arpa opened.
Host table received.
Connection to sri-nic.arpa closed.
# /usr/etc/htable hosts.txt
Warning, no localgateways file.
#
```

The `htable` program generates three data files: /etc/hosts, /etc/networks, and /etc/gateways. (Gateways are discussed in Chapter 12, "Setting up routing," page 95.) If a file named localhosts is present in the working directory, it is copied to the output file preceding the /etc/hosts file created by the `htable` program. Similarly, a file named localnetworks is copied to the /etc/networks file preceding the network data created by `htable`, and the localgateways file is copied to the front of the /etc/gateways data. Before installing new host and network databases in the /etc directory, run the `diff` file comparator on the old and new host databases to ensure that the files are complete.

If you use the name server for address resolution, you need only install the /etc/networks file and a small /etc/hosts file describing your local hosts. You may want to place the full host table elsewhere for reference by users.

If you are connected to the DARPA Internet, it is recommended that you use the name server, because it provides access to a much larger set of hosts than are in the host table. Many large organizations on the network that run the name server currently have only a small percentage of their hosts listed in the host table retrieved from the NIC.

Completing host name database setup

If your system uses the host table lookup method of name resolution, your host name database is configured and ready to use upon completion of procedures outlined in this chapter. To use the name server, however, you must also complete tasks described in Chapter 15, "Configuring the name server," page 111.

Configuring STREAMS 11

STREAMS is a message-passing system consisting of system calls, kernel resources, and kernel routines. On systems running ConvexOS V11.0 or later, Internet Services TCP/IP protocols (TCP, UDP, and IP) are implemented as a STREAMS-based architecture.

This chapter explains how to define the STREAMS TCP/IP protocol stack in the knetd.conf file for each type of network interface supported by ConvexOS, and how to start and stop the STREAMS stack.

Defining STREAMS protocol stacks

On systems running ConvexOS V11.0 or later, Internet Services TCP/IP protocols (TCP, UDP, and IP) are implemented as a STREAMS-based architecture. When the system is booted, the boot process runs the `knetdctl` utility, which causes the STREAMS network daemon `knetd` to use information in the knetd.conf file to build the STREAMS protocol stack.

The knetd.conf file included in the standard distribution provides a default configuration definition for a TCP/IP stack and one Ethernet interface stack. You need to modify knetd.conf to correctly build the protocol stack for additional interfaces.

There are two ways to define a STREAMS protocol stack in the knetd.conf file.

- Use the `io2knetcf` utility to generate the knetd.conf file from I/O descriptions found in the /ioconfig file.
- Manually edit the knetd.conf to add the appropriate definitions.

Changes made to the knetd.conf file take effect when the system is rebooted.

The knetd.conf file installed with the software includes explanatory comments; you should read this file before making any changes to it. For information on knetd.conf file syntax, read the knetd.conf(5) man page.

The following sections describe changes needed in the supplied knetd.conf file to define different types of network interfaces.

Generating the knetd.conf file from /ioconfig

For the following types of network interfaces, the `io2knetcf` utility generates the /etc/knetd.conf file from I/O descriptions found in the /ioconfig file:

- LAN-001 and LAN-007 Ethernet interfaces
- LAN-208 FDDI
- LAN-502 HIPPI/TCP
- LAN-002, LAN-004, and LAN-204 UltraNet interfaces

Before running `io2knetcf`, make sure your /ioconfig file contains entries for each network interface installed on the system. To use `io2knetcf`, enter

```
io2knetcf > /etc/knetd.conf
```

By default, `io2knetcf` uses the /ioconfig file as input and overwrites the existing knetd.conf file. You can specify alternate input and output files on the command line. For complete information on `io2knetcf`, refer to the io2knetcf(8) man page.

Manually defining additional Ethernet stacks

Follow the steps in this section if your system uses additional Ethernet interfaces.

Step 1 Duplicate the lines shown in Figure 36 for each additional interface.

Figure 36 Ethernet definition in the knetd.conf file

```
mac:0   -         PORT={0}        # open stream to mac device, use unit 0
ip      mac:0     PORT={2048, IP}# link ip to mac stream, dlsap 2048
mac:1   -         PORT={0}        # open stream to mac device, use unit 0
ip      mac:1     PORT={2054, ARP}
mac:2   -         PORT={0}        # open stream to mac device, use unit 0
ip      mac:2     PORT={32821, REVARP}
```

Step 2 Modify the duplicated lines as follows:

- Increment the `mac:` number for each pair.
- Modify the `PORT={0}` number to reflect the controller number for each new interface.

Figure 37 shows an example of a second Ethernet controller definition.

Figure 37 Second Ethernet definition in the knetd.conf file

```
mac:3   -         PORT={1}        # open stream to mac device, use unit 1
ip      mac:3     PORT={2048, IP}# link ip to mac stream, dlsap 2048
mac:4   -         PORT={1}        # open stream to mac device, use unit 1
ip      mac:4     PORT={2054, ARP}
mac:5   -         PORT={1}        # open stream to mac device, use unit 1
ip      mac:5     PORT={32821, REVARP}
```

Manually defining an FDDI stack

Follow the steps in this section to define a CONVEX FDDI in the knetd.conf file.

Step 1 Duplicate the lines shown in Figure 36 for each FDDI interface.

Step 2 Replace the `mac` keyword with `macf`, as shown in Figure 38, if your system also has an Ethernet interface.

Figure 38 CONVEX FDDI definition in the knetd.conf file

```
macf:0 -          PORT={0}        # open stream to macf device, use unit 0
ip     macf:0     PORT={2048, IP}# link ip to macf stream, dlsap 2048
macf:1 -          PORT={0}        # open stream to macf device, use unit 0
ip     macf:1     PORT={2054, ARP}
macf:2 -          PORT={0}        # open stream to macf device, use unit 0
ip     macf:2     PORT={32821, REVARP}
```

Step 3 At the beginning of the knetd.conf file in the Modules section, locate the line that reads

```
mac dc /dev/eth
```

Add a line after it that reads

```
macf dc /dev/fddi
```

Defining a HYPERchannel stack

The `io2knetcf` utility does not check for HYPERchannel interfaces; therefore, you must define a HYPERchannel stack by modifying the knetd.conf file. Follow the steps in this section to define a HYPERchannel interface in the knetd.conf file.

Step 1 In the Modules section, remove the # (pound sign) from the following line:

```
#mach dc /dev/hyper
```

Step 2 In the Streams section, add the following statements for each controller. Be sure to change the `mach:` number and `PORT={0}` appropriately for each controller.

```
mach:0 -          PORT={0}
ip     mach:0     PORT={5, IP}
```

Defining a SLIP stack

To cause `io2knetcf` to generate a Serial Line Internet Protocol (SLIP) stack, enter

```
io2knetcf -slip > /etc/knetd.conf
```

To manually define SLIP in the knetd.conf file, follow these steps:

Step 1 In the Modules section, add the following line:

```
macs   dc         /dev/sl
```

Step 2 In the Streams section, add the following statements for each SLIP connection. Be sure to change the `macs:` number and `PORT={0}` appropriately for each controller.

```
macs:0 -          PORT={0}
ip     macs:0     PORT={5, IP}
```

Manually defining a HIPPI/TCP stack

To define a HIPPI/TCP interface in the knetd.conf file, follow these steps:

Step 1 In the Modules section, add the following line:

```
maci dc /dev/hippi
```

Step 2 In the Streams section, add the following statements for each HIPPI controller. Be sure to change the `maci:` number and `PORT={0}` appropriately for each controller.

```
maci:0 -          PORT={0}
ip     maci:0     PORT={2048, IP}
```

Manually defining an UltraNet stack

To define an UltraNet interface in the knetd.conf file, follow these steps:

Step 1 In the Modules section, remove the # (pound sign) from the following line:

```
#un    dc         /dev/un
```

Step 2 In the Streams section, add the following statements for each UltraNet controller. Be sure to change the `un:` number and `PORT={0}` appropriately for each controller.

```
un:0   -          PORT={0}
ip     un:0       PORT={2048, IP}
```

Stopping and starting the STREAMS stack

The `knetdctl` utility causes `knetd` to read the configuration specified in the knetd.conf file and build the specified STREAMS protocol stack. When the system is booted, the rc.local script runs the `knetdctl` utility to build the STREAMS stack, but, in some situations, you may need to issue `knetdctl` commands to shut down, reconfigure, and restart the STREAMS stack.

Note

The shutdown and restart options to `knetdctl` should be issued only when the system is in single-user mode.

Stopping the STREAMS stack

To stop the STREAMS stack, follow this procedure:

Step 1 Go to single-user mode.

Step 2 If your system is using the name server, `named`, remove the /etc/use_nameserver file. Otherwise, the `umount` command will hang while trying to talk to `named`, which is killed when the system goes into single-user mode.

Step 3 Unmount all NFS mounts by entering

```
umount -at nfs
```

Step 4 Stop the STREAMS stack with the command

```
knetdctl -s
```

Note

Shutting down the STREAMS stack without unmounting NFS mounts and then booting multiuser will cause the system to hang because the network is down.

Running `knetdctl` at boot time

To cause `knetd` to build the STREAMS stack when the system is booted, ensure that the lines shown in Figure 39 are in your rc.local file. These lines must come before any `ifconfig` statement, or any statement for any other network operation.

Figure 39 Starting STREAMS from rc.local

```
# build the networking streams stack
if ["`/etc/knetdctl -q`" = "knetd not configured"]; then
                /etc/knetdctl -c /etc/knetd.conf
                /etc/knetdctl -r
fi
```

`knetdctl` command options can also be issued from the command line. Options have the following effects:

`-q` (query) Queries `knetd` for its current configuration status. If `knetd` has not built the STREAMS protocol stack, `knetdctl` displays the message "`knetd not configured.`" If `knetd` has built the STREAMS protocol stack, `knetdctl` displays the message "`knetd configured.`"

`-c` (set configuration file) Causes `knetd` to use the specified configuration file during any subsequent restart commands.

`-r` (restart) Causes `knetd` to read the configuration from the current knetd.conf file and build the specified STREAMS protocol stack.

`-t` (trace) Causes `knetd` to display trac ing information while processing the configuration file to build the STREAMS protocol stack. `knetd` tracing information appears only on the system console.

Setting up routing

12

Previous chapters discussed host names and internet addresses as the means by which hosts on a network are identified. Addresses tell networking software where to send packets. Sometimes, however, the sender does not know where to find the destination host because the two machines are connected to different physical networks. In such cases, the sender needs a way of determining how to get the packet to its destination. This chapter discusses methods of *routing*, the task of finding the path over which to send packets to their destination.

Routes, bridges, and gateways

The simplest *route* connects one host to another on the same LAN. Packets never leave the physical network where they originate; they reach their destination directly over the LAN hardware. The process of routing becomes complicated when a host needs to send a packet to a machine on a different physical network. Transferring packets across physical network boundaries requires services provided by gateways and bridges.

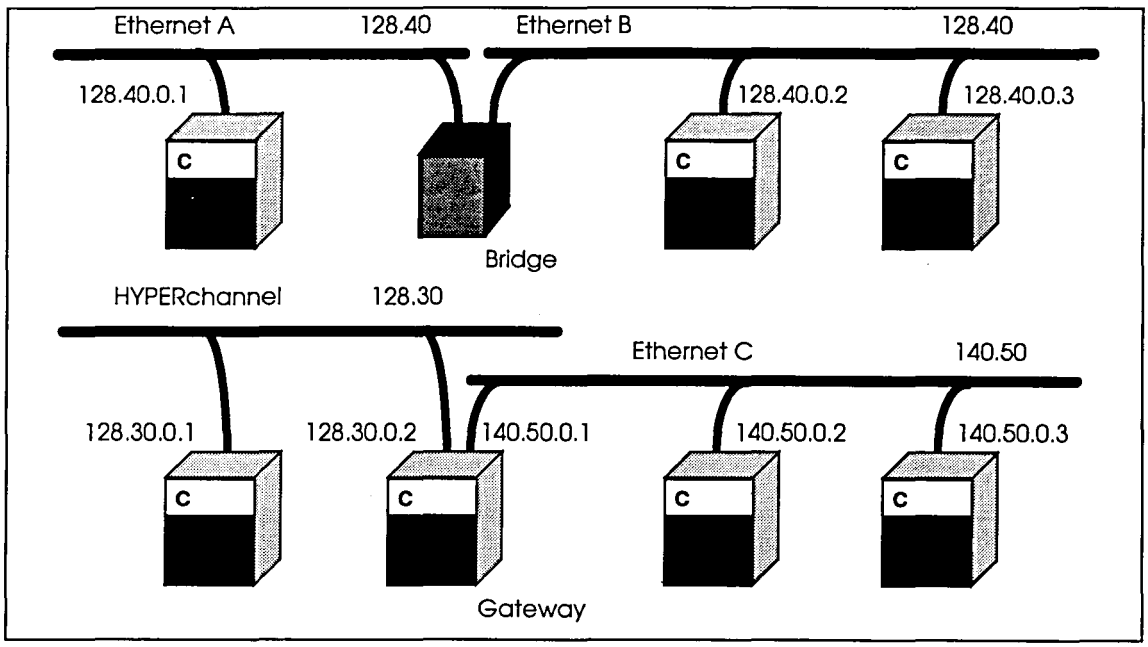
Gateways connect networks. Some are special-purpose packet-switching computers; others are simply hosts connected to multiple network interfaces, which transfer packets across networks in addition to serving as general purpose computers. Gateways can connect similar or different types of networks. For example, you might use a gateway to connect an Ethernet to a HYPERchannel.

Gateways that exchange routing information are called *active*, or *smart*, gateways; those that do not are called *passive* gateways. Active gateways keep dynamically-created routing tables up to date for use by themselves and by passive gateways.

Bridges also connect networks, but at the physical level. They normally connect networks of the same type. For example, you can use a bridge to catenate Ethernet cables so that they appear to network software as a single physical entity. Unlike gateways, bridges are transparent to network software.

Figure 40 illustrates the difference between bridges and gateways, and the purpose each serves in connecting networks.

Figure 40 The role of bridges and gateways



As you can see from the diagram, gateways have two internet addresses (one for each network interface), whereas bridges have none. Because bridges are transparent to networking software, hosts on Ethernet A and Ethernet B view the LAN as a single physical network. Hosts on Ethernet C, however, must use different network numbers to reach hosts on the HYPERchannel or the Ethernet network.

Setting up Internet routing

As installed, CONVEX Internet Services software can route packets only to networks directly attached to your host. If you plan to connect your LAN to other networks, you must set up a method for determining routes to hosts on those other networks.

For your network to successfully route packets that originate within it, each machine needs to know the route to all possible destinations. Network software bases routing decisions on information stored in *routing tables*. Routing table entries include destination internet address, address of the gateway to the destination network, address of the network interface, and other information.

You can use one of three methods to define and maintain routing tables:

- Configure the system to use the network routing daemon, `routed`, to maintain routes dynamically.
- Use the `route` command to create a set of static routes the system loads into routing tables at startup. Using static routes has the same disadvantage as using the table-lookup method to resolve host names—it requires that you update information on all networked hosts each time any route changes.
- Define a *default*, or *wildcard*, route to a smart gateway and rely on that gateway for routing information. This method allows you to use one host on your LAN to handle all traffic to and from other networks. The disadvantage of this method is that if the smart gateway goes down, your network becomes isolated from other networks.

Note

Regardless of which routing method you choose, you must configure network interfaces before enabling routing.

Using the routing daemon

Unless your network is small and relatively stable, your best option is to use the routing table management daemon `routed` to maintain system routing tables. The routing daemon maintains up-to-date routing tables in a cluster of LANs. If you add suitable entries to /etc/gateways, you can also use `routed` to initialize static routes to distant networks.

You normally start `routed` at boot time from the /etc/rc.local file. To use `routed`, your /etc/rc.local must include the following lines (Chapter 3, "Configuring an Ethernet interface," page 11, describes how to set up your /etc/rc.local file to configure the system at boot time.):

```
if [ -f /etc/routed ]; then
   /etc/routed & echo -n ' routed'
fi
```

These lines are included in the /etc/rc.local file created as a function of installing CONVEX Internet Services.

When `routed` first starts, it reads /etc/gateways and installs routes defined there. The routing daemon then attempts to find peer routing daemons on other hosts by sending out a broadcast message on each network interface to which the host is attached. If `routed` receives any responses, the local daemon cooperates with remote daemons to maintain a globally consistent view of

routing in the local environment. You can also add entries to /etc/gateways if you want to communicate with remote sites running the routing daemon. If you find that /etc/gateways does not exist on your system, create it following the format described in Appendix A of this document and in the routed(8) man pages.

Creating static routes

Another way to route packets is to create static routes using the `route` command. To do this, edit your /etc/rc.local file, adding the `route` commands you require. You can add and delete routes to specific hosts or to networks. Figure 41 illustrates typical `route` commands.

Figure 41 Sample `route` commands

```
#
# Internet routes
#
# the format is:
# /etc/route [add|delete] [net|host] destination
gateway
#[metric]
#
/etc/route add host    128.70.50.1   129.50.1.1  2
/etc/route add net     128.68.1      128.68.4.1  1
/etc/route add default 129.50.1.2
#
```

Entries specify the internet address of the gateway used to reach the host or network. You can also identify a *default* gateway to send packets to if no other route is known (the next section discusses default routes more thoroughly).

Note

If you decide to create and maintain static routes, be aware that as your network grows, so will the task of updating routing tables on all computers it serves.

Defining default routes

The third approach to routing combines `route` and `routed`. You define a default or wildcard route to a smart gateway, then depend on that gateway to provide routing redirect information used to dynamically create and update routing tables. Define a

wildcard route by adding an entry similar to the following to /etc/rc.local:

```
/etc/route add default smart_gateway 1
```

The variable, *smart_gateway,* is the host name of the gateway you wish to access; **1** specifies the number of hops (gateway machines) in the path to the destination. Specifying **`default`** tells networking software to take the route if no other routes to the destination are known. The route(8C) man page contains a more complete explanation of this command and its syntax.

The system uses the default route as a "last resort." If the gateway to which packets are directed is able to generate the proper routing redirect messages, the system updates routing table entries based on the information supplied.

You cannot use the wildcard method in an environment that contains only bridges (which do not generate routing-redirect messages) and no gateways. Furthermore, if the smart gateway crashes, there is no way to maintain service other than manually altering the routing table entry.

Because the system always listens for and processes routing-redirect messages, it is possible to use `routed` to maintain up-to-date information about routes to geographically local networks and the wildcard method for routing to "distant" networks.

Setting boot-time routing options

Three boot-time parameters control additional routing options: *gateway, ipsendredirects,* and *ipforwarding*.

If you set the `gateway` option, network software uses a larger routing table and forwards packets even if the host has only a single, non-loopback interface. If the `ipsendredirects` option is set, network software sends Internet Control Message Protocol (ICMP) redirect messages for use in routing. Setting `ipforwarding` enables IP packets to be forwarded when the internet address does not correspond to any of the internet addresses for the machine's network interfaces. Refer to the section "Networking Boot-Time Parameters," in this book and the chapter "Customizing Kernel Boot-Time Parameters," in *Managing ConvexOS: Configuration Guide* for a complete explanation of these parameters and how to change them.

Verifying routes with `netstat`

Use the *netstat* program to display routing table contents and statistics. For example, use the following command to display the contents of the routing tables:

```
netstat -r
```

Typical output is shown in Figure 42.

Figure 42 Sample output from `netstat -r`

```
Routing tables
Destination     Gateway    Flags  Refcnt  Use       Interface
localhost       localhost  UH     2       3736      lo0
acme-net        acmes      U      49      1050480   ex0
dragon-net      acmee      UG     0       20834     ex0
mktg-net        acme1-ex0  UG     1       1095      ex0
```

The following command displays the number of routing tables dynamically created as the result of routing redirect messages:

```
netstat -r -s
```

Typical output is shown in Figure 43.

Figure 43 Sample output from `netstat -r -s`

```
routing:
   0 bad routing redirects
   9 dynamically created routes
   3 new gateways due to redirects
   0 destinations found unreachable
   16 uses of a wildcard route
```

Configuring inetd 13

This chapter describes procedures for configuring network services handled by **inetd**.

Configuring the Internet superserver daemon

Each internet utility, such as `ftp`, `rcp`, and `rlogin`, uses a pair of processes to communicate over the network. A *client* process running in the sending host initiates a request for a connection to the destination host. Its corresponding *server* running on the destination host awaits such requests and performs necessary actions—initiate the transfer of a source program, list the contents of a file, execute a command, etc.—to complete the request. This section describes procedures for configuring internet servers to process requests for networking services.

To minimize the number of networking processes running on the system, CONVEX Internet Services uses a single daemon to service multiple network requests. This daemon is called *inetd*, or the "internet superserver." `inetd` is started automatically at boot time from the /etc/rc script. Once started, `inetd` reads a configuration file called /etc/inetd.conf and monitors a set of ports associated with each internet service defined there. When `inetd` receives a connection request on one of its ports, it activates the appropriate daemon to service the request.

Use the following procedure to ensure that necessary network daemons start at boot time:

Step 1 All daemons except those beginning with `rpc` should be listed in the /etc/inetd.conf file. `rpc` daemons are started only if your site runs NFS. If you have any `rpc` daemons listed in the configuration file, but the `rpc` superserver `portmap` has not been started, `inetd` logs an error message using `syslog` and continues without starting up any `rpc` servers. Once `portmap` has been started, `inetd` sets up the daemons listed in the configuration file if it receives a SIGHUP signal.

If sent a SIGHUP signal, `inetd` rereads the configuration file and reconfigures itself. If you do not want a particular daemon started, edit the /etc/inetd.conf file, turning the appropriate line into a comment by adding a "#" at the beginning of the line.

The format for records in the /etc/inetd.conf file is

service_name {**stream** | **dgram** | **raw**} *protocol* {**wait** | **nowait**} *user server_program* [*arguments*]

where:

service_name	Name of a valid service in the /etc/services or /etc/rpc file (if your site runs NFS.) See below.
stream	Socket type.
dgram	
raw	
protocol	Name of a valid protocol listed in the /etc/protocols file.
wait	Flag indicating whether the server processes all incoming
nowait	messages on the socket (**wait**), or frees the socket after receiving a message (**nowait**). This parameter applies only to datagram (**dgram**) sockets.
user	User name under which the server runs. This parameter allows the system manger to give servers less permission than `root`.
server_program	Pathname of the program `inetd` executes when it receives a request on the socket. If `inetd` provides the service itself, this field should be "internal."
arguments	Arguments passed to the server program.

Figure 44 shows the contents of the /etc/inetd.conf file installed as a function of installing network software.

(NFS is an optional product. For more information about configuring `rpc` daemons, refer to Part 3, "Part 3 Configuring Internet Services," page 61, and the *CONVEX Network Programming Guide*.)

Figure 44 Sample inetd.conf file

```
#
# Copyright 1990 Convex Computer Corp.
#
# Internet server configuration database
#
ftp         stream  tcp    nowait  root    /usr/etc/in.ftpd            ftpd
telnet      stream  tcp    nowait  root    /usr/etc/in.telnetd         telnetd
shell       stream  tcp    nowait  root    /etc/in.rshd                rshd
login       stream  tcp    nowait  root    /etc/in.rlogind             rlogind
exec        stream  tcp    nowait  root    /usr/etc/in.rexecd          rexecd
syslog      dgram   udp    wait    root    /usr/etc/in.syslog          syslog
comsat      dgram   udp    wait    root    /usr/etc/in.comsat          comsat
talk        dgram   udp    wait    root    /usr/etc/in.talkd           talkd
crashdump   dgram   udp    wait    root    /usr/etc/in.crashreceive    crashrcv
rusers      dgram   udp    wait    root    1-2 /usr/etc/rpc.rusersd    rusersd
rup         dgram   udp    wait    root    1-3 /usr/etc/rpc.rstatd     rstatd
rwall       dgram   udp    wait    root    1 /usr/etc/rpc.rwalld       rwalld
mount       dgram   udp    wait    root    1 /usr/etc/rpc.mountd       mountd
quota       dgram   udp    wait    root    1 /usr/etc/rpc.rquotad      rquotad
spray       dgram   udp    wait    root    1 /usr/etc/rpc.sprayd       sprayd
rex         stream  tcp    wait    root    1 /usr/etc/rpc.rexd         rexd
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
```


For more information about `inetd` and the /etc/inetd.conf file, refer to the inetd(8C) man page.

Step 2 If you have added any daemons, you must start them by running from the command line or shutting down to single-user mode (using `reboot` or `shutdown -r`) and rebooting according to instructions in the *CONVEX Processor Operation Guide.*

After completing the steps above, you should be able to use the network and its associated services. As a check, exercise utilities associated with each daemon you have installed. For example, enter the command

`telnet` ***hostname***

The host you specified should respond with a login prompt. If it does not, use the troubleshooting procedures outlined in Part 5, "Part 5 Testing and troubleshooting the network," page 329.

The /etc/services file

The /etc/services file defines services provided by the network. Entries in the /etc/services file specify a service name and its aliases, associated port number, and protocol name.The format for records in the /etc/services file is

official_service_name port_number/protocol_name [aliases] [**#***comment*]

where:

service_name — Name of the command associated with the service.

port_number/protocol_name — Number of the port associated with the service and name of the protocol it uses. These parameters are expressed as a single item with a "/" separating them, as in `512/tcp`.

aliases — Any other names you wish to call the server.

#*comment* — All characters right of the "#" are ignored by routines that use the file.

For more information about the services file, refer to the services(5) man page.

Figure 45 shows a sample /etc/services file.

Figure 45 Sample /etc/services file

```
#
# Network services, Internet style
#
echo            7/udp
echo            7/tcp
discard         9/udp           sink null
discard         9/tcp           sink null
systat          11/tcp
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp          mail
uucp            33/tcp          uucpd
time            37/tcp          timserver
time            37/udp          timserver
name            42/tcp          nameserver
name            42/udp          nameserver
whois           43/tcp          nicname
hostnames       101/tcp         hostname        # usually from sri-nic
#
# Host specific functions
#
tftp            69/udp
finger          79/tcp
#
# UNIX specific services
#
exec            512/tcp
login           513/tcp
shell           514/tcp         cmd             # no passwords used
biff            512/udp         comsat
who             513/udp         whod
syslog          514/udp
talk            517/udp
route           520/udp         router routed # 521 also
timed           525/udp         timed           # 4.3 timed
llockmgr        536/udp         lockf           # for lockf
#
# service for time delivery system
#
batch           666/tcp         batchd
nqs             607/tcp                         # nqs batch (tcp)
nqs             607/udp                         # nqs batch (udp)
crashdump       670/udp
```

Configuring anonymous ftp accounts

14

This chapter describes an optional procedure for enabling remote users to transfer files using ftp.

The ftp file server provides support for an anonymous ftp account. This account enables users to transfer files between machines without using a password. Files transferred are written to a common user account, enabling users to transfer files to a secure or heavily protected host without gaining unrestricted access to the host's file system.

Note

There are inherent security problems with any user account of this type. Therefore, before you install this account, make sure that benefits to your users outweigh possible security risks. If you decide to install this account, read this chapter carefully. By following the procedures described here, you will make the installation much safer than it might otherwise be.

Enable the anonymous account by creating a user named ftp. (You can use nu to create the ftp user. Refer to the nu(8) man page for details.) When a client uses the anonymous account, the ftp server performs a chroot system call. In this way, the user is restricted from moving outside the part of the file system containing the common ftp directory.

For this system to work properly, you must ensure that all directories and executable images associated with the ftp account are restricted to reading and executing (not writing), except for the directory into which you allow anonymous users to transfer files (pub). Furthermore, you must ensure that certain programs and files are supplied to the server. Enter the following sequence of commands to allocate files and directories needed. The subdirectory, ~ftp/pub, is allocated as the common ftp directory.

Figure 46 shows the sequence of commands used to set up an anonymous `ftp` account.

Figure 46 Setting up anonymous `ftp` accounts

```
# cd ~ftp
# chmod 555.
# chown ftp .
# chgrp ftp .
# mkdir bin etc pub
# chown root bin etc
# chmod 555 bin etc
# 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
```

When performing the last command in Figure 46, be sure to edit the `passwd` and `group` files to contain only `root` and `ftp`. Doing so helps minimize security risks.

The system described here is compromised if you allow access to this account by users logged in as `uucp` or `root`.

You may also run into problems if you allow logins by accounts with nonstandard shells and no passwords (for example, `rwho` or `finger`). To avoid these types of problems, make sure to add these names to the /etc/ftpusers file (or create the file if it does not exist). This file is checked each time `ftp` is used. If the requested user name is in the file, the request for service is denied. Read more about the ftpusers file in the ftpd(8) man page.

Configuring the name server

15

This chapter describes how to configure a system to use the Berkeley Internet Name Domain server (BIND). If you choose to use the name server, your primary task as system manager is to configure the system to use BIND as a replacement for /etc/hosts table lookup. Configuring the name server is not a simple task; it involves setting up several complex files. The benefit comes from having to do this on only one machine and letting the name server keep the files up-to-date on other networked machines.

This chapter contains three sections:

- An introduction to name server concepts for the reader who is unfamiliar with the internet name server.
- Descriptions of files used by the name server system.
- A step-by-step guide to configuring each type of name server.

Specifications for the name server are defined by Requests for Comments (RFCs), which you can find in /usr/lib/conf/bind. Familiarity with the material in these documents will aid you in configuring the name server for your system. It is also recommended that you read related man pages, named(8), resolver(3), and resolver(5).

Introduction

The Berkeley Internet Name Domain (BIND) server implements the DARPA Internet name server. A name server is a network service that enables clients to name resources or hosts and share this information with other hosts in the network. This is a distributed database system for hosts in a computer network.

Internet domain names

The internet name server views the network as a hierarchy of *domains*. A domain is a tree structure that can be organized according to organizational or administrative boundaries. Each domain in the hierarchy has a *label*. A *fully-qualified domain name* is the concatenation of all labels of domains from the root (highest level in the hierarchy) to the current domain, listed from right to left and separated by dots. A label need only be unique within its domain.

Name space is partitioned into areas called *zones*, usually represented by administrative boundaries. Each zone starts at a domain and extends down to the leaf domain (lowest level in the hierarchy) or to domains where other zones start. A given name server is said to have *authority* over a particular zone, meaning that the server maintains all data corresponding to that zone.

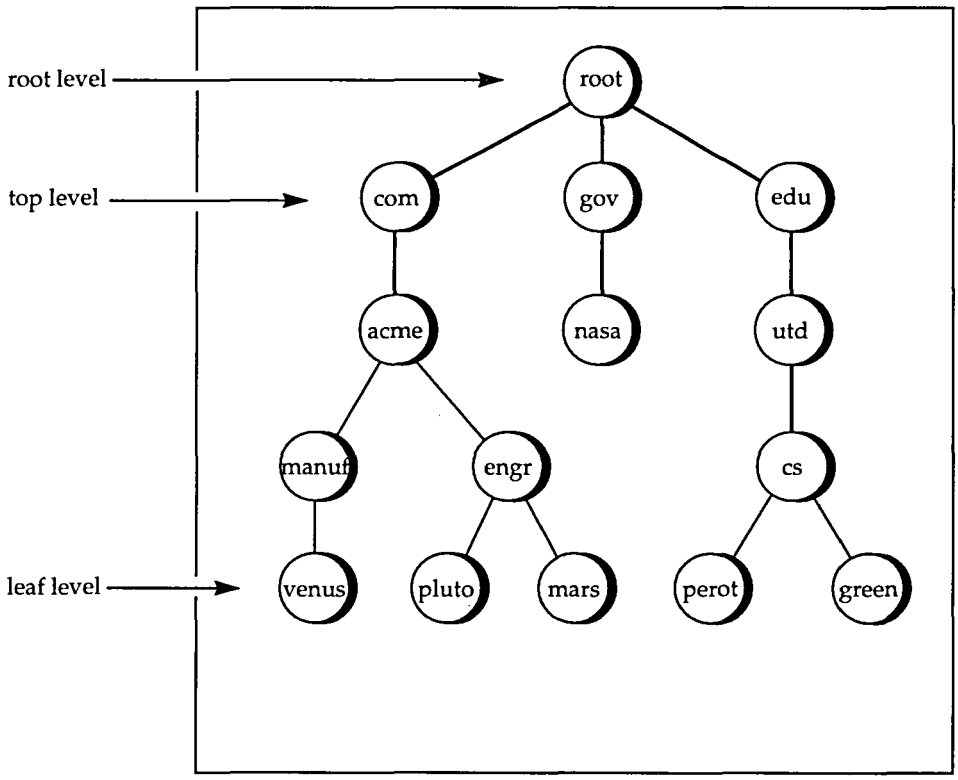
A domain name for a host at a university might be as follows:

```
perot.cs.utd.edu
```

The *top level* domain for educational organizations is `edu`; `utd` is a *subdomain* of `edu`, and `perot` is the name of the host used by the computer science (`cs`) department. A name server running on `perot` services all names in its zone of authority, which may be all the names for machines on the campus network or a subset of those names.

Figure 47 shows a hierarchical name structure.

Figure 47 Domain name tree structure



In the example above, fully-qualified domain names for hosts at the leaf level are venus.manuf.acme.com, pluto.engr.acme.com, mars.engr.acme.com, perot.cs.utd.edu, and green.cs.utd.edu.

Internet domain name server software

BIND software comprises two parts: `named` and the `resolver`. `named` is a daemon that services queries on a given network port (as defined in the /etc/services file). The `resolver` consists of a few routines that build query packets and exchange them with the name server.

You can configure your system to use any of several types of servers. The type you choose to run on a given system depends upon factors such as the amount of disk and memory space available on each system and the extent to which you want users on the system to have access to the network.

BIND provides the following types of servers:

- **Master server**—A *master server* for a domain has authority over that domain. This server maintains all data within its domain. Each domain should have at least two master servers, a primary master and at least one secondary master to provide backup service if the primary is unavailable or overloaded. A server may be a master for multiple domains; it may serve as primary for some domains and secondary for others.
  - **Primary**—A *primary master server* loads its data from a file on disk. This server may also delegate authority to other servers in its domain.
  - **Secondary**—The primary master server delegates authority to and provides data for a secondary master server. At boot time, the secondary server requests all data for the given zone from the primary master server. This server then periodically checks with the primary server to see if it needs to update its data.
- **Caching–only server**—A *caching* server is one that caches information it receives. Master servers and *caching-only* servers both cache information; however, a *caching-only server* is not authoritative for any domain. It services queries and asks other servers who have authority for the information it needs. All caching servers keep data in their caches until the data expires, based on a time–to–live field attached to the data when it is received from another server.
- **Resolving-only server**—You can use a name server on a workstation or on a machine that has a limited amount of memory and CPU cycles by choosing the *resolving-only* server option. This option allows you to run all networking programs that use the name server without running the name server on the local machine. Queries are serviced by a name server that is running on another machine on the network.

  Queries by resolving-only servers are recursive. The resolving-only server sends queries to each server in a list of available servers until the list is exhausted.

  You typically use a resolving-only configuration when you do not wish all servers at a given site to interact with the rest of the internet servers. For example, suppose your network consists of a number of workstations and a departmental timesharing machine with internet access. You might want to prohibit workstations from having internet access. To give workstations the appearance of access to the internet domain system, you could configure them as resolving-only servers

to the timesharing machine. The timesharing machine would then forward queries and interact with other name servers.

An added benefit of using the forwarding feature is that the central machine develops a much more complete cache of information of which all workstations can take advantage. The use of slave servers and forwarding is discussed further under the description of the named boot file commands.

System files

Configuring the name server requires you to create or modify several control and data files. Control files influence execution of named. Data files make up the name server's database. This section describes the content and format of files used by the name server system.

File summary

The name server system uses the files listed below. Refer to the sections, "Control file descriptions" page 116 and "Data file descriptions" page 120 for full path names, detailed descriptions, and examples of each file.

named.boot	Name server configuration boot file
named.reload	Script used to reload and restart named
named.restart	Script used to restart named
named.pid	Contains the process id of named
resolv.conf	Resolver configuration file
use_nameserver	File to indicate that named is in use
hosts	Host name database
named.hosts	Contains data about all hosts in this zone
named.local	Contains the address of the local loopback interface
named.rev	Contains internet host addresses in reverse order
root.cache	Contains addresses of authoritative name servers for the root domain
/usr/lib/conf/bind	Contains referenced RFCs and other documentation

Control file descriptions

This section contains detailed descriptions and formats for files used to control the execution of the name server.

named.boot file

The boot file, /etc/named.boot, contains name server configuration information. The name server daemon named reads this file when it first starts up. The file specifies the server's type, zones of authority, and location of its initial data. You can change the path name for the boot file from the default, /etc/named.boot, by specifying another path name on the command line that starts named. (See Figure 48 for an example of the commands used to start up named.)

Figure 48 shows a sample named.boot file.

Figure 48 Sample named.boot file

```
; named.boot
;
directory /usr/local/domain

; type       domain                 source host/file          backup file

cache        .                      root.cache
primary      acme.com               named.hosts
primary      32.128.IN-ADDR.ARPA    named.rev
secondary    mars.acme.com          128.32.137.8 128.32.137.3 mars.hosts.bak
secondary    6.32.128.IN-ADDR.ARPA 128.32.137.8 128.32.137.3 mars.rev.bak
primary      0.0.127.IN-ADDR.ARPA   localhost.rev
forwarders   10.0.0.78 10.2.0.78
;
```

The boot file may contain the following command lines, depending on how you wish to configure your name server:

- **directory**—Specifies the root directory for the name server. The directory command allows you to specify other file names in the boot file relative to this path name. You should always include a directory command in your boot file. The format of the directory command is:

  directory */root_directory*

  If you have more than a couple of named data files to maintain, you can place files in a directory such as /usr/local/domain and specify that path name with the directory command.

This command also ensures that named is in the proper directory to locate $INCLUDE and source/host files specified with relative path names.

- **primary**—Designates the server as a primary master server. The format of the primary command is

  primary *domain_name* *hosts_file*

  The first field specifies that the server is a primary master for the domain named in the second field. The third field is the path name of the file containing the domain data, relative to the path name specified with the directory command.

  In the example in Figure 48, the first primary command specifies that the database for domain acme.com is located in the file, named.hosts. Records in named.hosts contain data in the master file format described in the "Standard resource record format" page 121.

  The second primary command in Figure 48 specifies that authoritative data for the domain 32.128.IN-ADDR.ARPA is located in the file, named.rev (reverse). The name server uses data in named.rev to translate addresses in network 128.32 to host names. This file is described in the section "named.rev" page 131.

- **secondary**—Designates the server as a secondary master server. The secondary command is similar to the primary command except that it lists addresses of other servers (usually primary master servers) from which the name server running on the local host obtains its domain data. The format of the secondary command is:

  secondary *domain_name* *primary_servers* *backup*

  The first field identifies the server as a secondary master for the domain named in the second field. The *primary_servers* field specifies names of primary servers for the domain. The secondary server obtains its data across the network from the listed servers. Secondary masters try each primary server in the order listed until they successfully receive data from a listed server. If you specify a file name after the list of primary servers, the name server will dump data for the domain into that file as a backup. When the server first starts, it loads data from the backup file if possible, then consults a primary server to ensure that data is still up-to-date.

- **cache**—Specifies the name of the file used to prime the name server's cache. All servers should have at least one `cache` command in the boot file. The format of the command is

  cache . *root.cache*

  At boot time, the name server reads the files listed and reinstates in the cache all values that are still valid.

- **sortlist**—Lists in order of preference addresses of other name servers to query. Queries for host addresses from hosts on the same network as the local server will receive responses with local network addresses listed first, then addresses in the sort list, then other addresses. This line has meaning only at initial start-up. When the name server is reloaded with a SIGHUP signal, it ignores this line. The format of the `sortlist` command is

  sortlist *preferred_servers*

named.pid

When `named` successfully starts, it writes its process id into the /etc/named.pid file. In addition to `named.reload` and `named.restart`, programs that need to send signals to `named` use this file. You can change the name of this file by defining the `PIDFILE` parameter before compiling `named`. (Refer to the section "Building a system with a name server" page 134.)

named.reload

The shell script, /usr/etc/named.reload, causes the server to read the boot file and reload the database. All previously cached data is lost.

If the `named.reload` file does not exist on your system, create it as shown in Figure 49.

Figure 49 Sample named.reload file

```
#!/bin/csh
#
# named.reload
#
kill -HUP `cat /etc/named.pid`
```

named.restart

The shell script, /usr/etc/named.restart, kills the active named process, and causes the server to start without reloading the database. Previously cached data is preserved.

If named.restart does not exist on your system, create it as shown in Figure 50.

Figure 50 Sample named.restart file

```
#!/bin/csh
#
# named.restart
#
kill -9 `cat /etc/named.pid`
/usr/etc/named
```

resolv.conf

To set up a host that will use a remote server instead of a local server to answer queries, create the file, /etc/resolv.conf. This file specifies the name servers on the network that should be sent queries. You should always create the resolv.conf file to include, at minimum, the domain directive.

The resolv.conf file may contain two types of directives: domain and nameserver. The domain directive allows you to specify the default domain name to append to names that are not fully qualified.

The nameserver directive allows you to specify the internet address of a name server for the local resolver to query. If no nameserver is specified, the system uses the name server on the local machine.

use_nameserver

The existence of this file on a system activates the name server resolver for that system. Create /etc/use_nameserver by using the touch command. To deactivate the name server, remove this file. (The /etc/use_nameserver file is discussed further under the section "Building a system with a name server" page 134.)

Note

In the ConvexOS implementation, /etc/use_nameserver is the key to using the name server. Create it to activate the server and remove it to deactivate the server.

Data file descriptions

The name server processes two kinds of data: *authoritative* and *cached*. Authoritative data is the complete database for a particular domain over which the name server has authority. The name server loads this data from master files or obtains it from another name server. A local resolver acquires and maintains cached data.

Three standard files specify data for a domain: named.local, named.hosts, and named.rev. Name server data files contain directives and resource records. A description of each of these files, as well as the name server cache file, root.cache, follows the discussion on standard resource record formats.

Data file directives

In addition to records in the standard resource record format, name server data files may contain the following directives:

- **$INCLUDE**—Requests that the name server load the data in the named file. This directive begins with **`$INCLUDE`** in column 1, followed by the name of files to load.

  This feature is primarily used to separate different types of data into multiple files. For example, you might want to keep mailbox data separate from host data. You could do this by specifying

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

  The name server interprets this line as a request to load the /usr/named/data/mailboxes file. `$INCLUDE` does not cause data to be loaded into a different domain.

- **$ORIGIN**—Change the origin in a data file. This directive begins with **`$ORIGIN`** in column 1, followed by a domain origin. The `$ORIGIN` feature is used for putting more than one domain in a data file. Each domain in the file should have a corresponding origin specified for it.

Standard resource record format

Data records in the name server data files are called resource records (RR). The standard resource record format is specified in detail in RFCs. The following is a general description of resource records:

```
[record_name]    ttl addr_class  record_type data
```

where:

record_name — Specifies the name of the domain record. If present, it must start in column 1. If you leave the name blank, the record takes on the name of the previous RR.

ttl — Specifies an optional time-to-live value. This value defines how long data will be stored in the database. If you leave this field blank, the name server uses the default time-to-live specified on the *Start of authority* (SOA) resource record.

addr_class — Specifies the address class. Currently, only the internet (IN) class is supported.

record_type — Specifies the resource record type. See "Record Types," below.

data — Contents depend on record type. Case is preserved in names and data fields. All comparisons and lookups in the name server database are case insensitive.

The following characters have special meanings:

. — In the name field, a "." refers to the current domain.

@ — In the name field, an "@" sign denotes the current origin.

.. — In the name field, two dots represent the null domain name of the root.

\X — Where X is any character other than 0-9. A backslash tells the name server to ignore any special meaning X normally carries. For example, "\." can be used to place a dot character in a label.

\DDD — Where each D is a decimal digit. The resulting octet is assumed to be text and is not checked for special meaning.

( )	Used to group data specified on more than one line. In effect, line terminations are not recognized within parentheses.
;	Signals the start of a comment; the remainder of the line is ignored.
*	Signifies use of wildcards.

The current origin is 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 if you do not want this to happen. In general, if the name is not in the domain for which you are creating the data file, end the name with a ".".

Resource record types

This section describes and gives examples of each type of resource record. Files are listed in the order in which they normally appear in data files.

- **Start of authority (SOA)**—Designates the beginning of a domain. There should be only one SOA record per domain. Figure 51 shows the format and an example of an SOA record.

Figure 51 Start of authority (SOA) record

```
[name]   ttl     class    SOA origin/data          administrator

;
@                IN       SOA mars.acme.com. root.mars.acme.com(
                          1.1       ; serial
                          3600      ; refresh
                          300       ; retry
                          3600000   ; expire
                          3600 )    ; minimum
```

name	Domain name.
origin	Domain name of host on which this data file resides.
administrator	Address of person responsible for this name server.
serial	Data file version number. Increment this number whenever you make a change to the data.

refresh	Interval, in seconds, at which a secondary name server checks with the primary name server to see if it needs to update its data.
retry	Interval, in seconds, at which a secondary server retries after failing to get a response when checking for a refresh.
expire	Maximum duration in seconds that a secondary name server uses its data after it fails to get a refresh.
minimum	Default, in seconds, for the time-to-live interval.

- **Name server (NS)**—Specifies the name of the server that has authority over the domain. There should be one NS record for each primary master server for a domain. Figure 52 shows the format and an example of an NS record.

Figure 52 Name server (NS) record

```
[name]   ttl     class     NS        server name

;

                 IN        NS        mars.acme.com.
```

server name **Name of the primary master server for the domain.**

- **Address (A)**—Specifies the internet address(es) of a given host. There should be one A record for each network address assigned to the host. A host with multiple network interfaces will have multiple A records. Figure 53 shows the format and examples of the A record.

Figure 53 Address (A) record

```
[name]   ttl     class     A         address

;
mars             IN        A         128.32.0.4
                 IN        A         10.0.0.78
```

name	Host name. The first A record for a host must include the host name.
address	Internet address of the host.

- **Host information (HINFO)**—Specifies host-specific information. There should be one `HINFO` record for each host. Host name and address are specified on a preceding `A` record. Figure 54 shows the format and examples of HINFO records.

Figure 54 Host information (HINFO) record

```
[name]   ttl   class    HINFO   host-specific information

;
altair         IN       A       140.150.70.1
               IN       HINFO   VAX-11/780 UNIX
deneb          IN       A       140.150.70.2
               IN       HINFO   C240 ConvexOS
sirius         IN       A       140.150.70.4
               IN       HINFO   machine_room
```

The first two HINFO records in the above example specify the hardware and operating system of the listed hosts. Only a single space separates the hardware and the operating system information. If you want to include a space in the machine name, you must quote the name. The third HINFO record specifies the location of the host. You can use HINFO to specify any information you want to associate with the host.

- **Well-known services (WKS)**—Lists well-known services supported by a particular protocol at a specific address. The list of services and port numbers are defined in the /etc/services file. There should be only one WKS record per protocol per address. Figure 55 shows the format and examples of the WKS record.

Figure 55 Well-known services (WKS) record

```
[name] ttl  class WKS  address     prot       services

;
          IN   WKS  128.32.0.10UDP  who route timed
          IN   WKS  128.32.0.10TCP  ( echo telnet
                                    discard daytime
                                    netstat ftp auth
                                    time whois finger
                                    smtp hostname )
;
```

- **Canonical name (CNAME)**—Specifies an alias for a canonical name. A CNAME record should be the only record associated with the alias name. All other resource records that include a domain name as their value (for example, NS or MX) should be associated with the canonical name, not with the alias.

  Figure 56 shows the format and an example of a CNAME record.

Figure 56 Canonical name (CNAME) record

```
aliases   ttl   class   CNAME   canonical name

;
cave              IN      CNAME   saturn
```

- **Domain name pointer (PTR)**—Allows special names to point to some other location in the domain. PTR names should be unique within a domain.

  Figure 57 shows the format and an example of a PTR record.

Figure 57 Domain name pointer (PTR) record

```
name      ttl   class   PTR     domain name

;
70.1              IN      PTR     saturn.acme.com.
```

In the above example, a PTR record is used to set up reverse addresses for the special `IN-ADDR.ARPA` domain. The record in the example is from the named.rev file shown in Figure 57. Host saturn has an internet address of `140.50.70.1`; the PTR record points to subdomain `70.1`.

- **Mailbox (MB)**—Designates the host on which a user wishes to receive mail. Mailbox names should be unique within the domain.

  Figure 58 shows the format and an example of an MB record.

Figure 58 Mailbox (MB) record

```
name      ttl   class   MB      host

;
jones             IN      MB      mars.acme.com.
```

name User's login name.

host Denotes the machine to which the server is to deliver the user's mail.

- **Mail rename (MR)**—Associates aliases with a user name. Figure 59 shows the format and an example of an MR record.

Figure 59 Mail rename (MR) record

```
alias     ttl   class    MR      MB name

;
postman   IN    MR       jones
```

alias An alias for the user name listed in the fourth field.

MB name Name on the corresponding MB record.

- **Mailbox information (MINFO)**—Creates a mail group for a mailing list. This record is usually associated with the MG (mail group), but may be used with an MB record. Figure 60 shows the format and an example of a MINFO record.

Figure 60 Mailbox information (MINFO) record

```
name    ttl   class    MINFO   requests          maintainer

;
BIND          IN       MINFO   bind-requestjj.acme.com.
```

name — Name of the mailbox.

requests — Name to which mail such as requests to be added to a mail group should be sent.

maintainer — Mailbox to which the server should send error messages. This is particularly appropriate for mailing lists when errors in members' names should be reported to a person other than the sender.

- **Mail group member (MG)**—Specifies members of a mail group. Figure 61 shows the format of the MG record and a sample mailing list.

Figure 61 Mail group member (MG) record

```
[group]  ttl   class    MG      member name

;
Bind           IN       MINFO   bind-requestjj.acme.com.
               IN       MG      ralph.acme.com.
               IN       MG      barney.acme.com.
               IN       MG      uriah.acme.com.
               IN       MG      jones.acme.com.
               IN       MG      white.acme.com.
```

- **Mail exchanger (MX)**—Designates a host to serve as a gateway. Figure 62 shows the format and sample MX records.

Figure 62 Mail exchanger (MX) record

```
name          ttl    class   MX   preference   mail gateway

;
munnari.oz.au IN     MX    0          seismo.css.gov.
*.IL.         IN     MX    0          relay.cs.net.
```

preference — Order in which a mailer should attempt to forward mail when there is more then one route to a single machine. (Refer to RFCs for more detailed information.)

The first MX record in the example above designates seismo.css.gov. as a mail gateway to munnari.oz.au. This MX record allows hosts that are not directly connected to the same network as munnari.oz.au to forward mail through the gateway, seismo.css.gov. Hosts seismo and munnari may have a private connection or use a different transport medium.

The second record in the example above shows the use of wildcard names for mail routing with MX records. There are usually servers on the network that simply state that any mail to a domain is to be routed through a relay. In the example, all mail to hosts in the domain IL is routed through relay.cs.net. This is done by creating a wildcard record that states that *.IL has an MX of relay.cs.net.

/etc/hosts

When you configure your system to use the name server, the /etc/hosts file is only used for setting interface addresses and when the name server is not running. Therefore, /etc/hosts need only contain addresses for local hosts.

Set up the /etc/hosts file for the hosts on your LAN according to procedures outlined in Chapter 10, "Setting up the host name database" page 69.

named.hosts

The named.hosts file contains all authoritative data for hosts in a domain. You specify the name of this file in the boot file; its location is relative to the path name specified with the `directory` command in the boot file. Records in the file follow the standard resource record format.

Figure 63 shows an example of a named.hosts file.

Figure 63 Sample named.hosts file

```
; named.hosts

@           IN  SOA    pluto.acme.comroot.pluto.acme.com. (
                1.1    ; Serial
                10800  ; Refresh 3 hours
                3600   ; Retry 1 hour
                3600000; Expire 1000 hours
                86400 ); Minimum 24 hours
            IN  NS     pluto.acme.com.

localhost   IN  A      127.0.0.1
saturn      IN  A      140.50.70.1
            IN  HINFO  machine_room c1e22
cave        IN  CNAME  saturn
timehost    IN  CNAME  saturn
andrewhost  IN  CNAME  saturn
antares     IN  A      140.50.70.2
            IN  HINFO  machine_room c1e22
fontshost   IN  CNAME  antares
localserver IN  CNAME  antares
snoopy      IN  A      140.50.70.4
            IN  HINFO  cbrown c2w47-1
hercules    IN  A      140.50.70.101
            IN  HINFO  ralph c1e69-1
harpo       IN  A      140.50.70.102
            IN  HINFO  adolph c1e69-r
chocolate   IN  A      140.50.70.106
            IN  HINFO  grant c2w46-1
choc        IN  CNAME  chocolate
cumulus     IN  A      140.50.70.107
            IN  HINFO  doobey c2w44-1
daffy       IN  A      140.50.70.108
            IN  HINFO  binaca c2w44-r
holmes      IN  CNAME  daffy
shylock     IN  A      140.50.70.109
            IN  HINFO  bluto c1e70-1
veggy       IN  A      140.50.70.113
            IN  HINFO  katey c2w47-r
vege        IN  CNAME  veggy
inanna      IN  A      140.50.71.174
            IN  HINFO  jimjones c2w34-r
juno        IN  A      140.50.71.177
            IN  HINFO  klutz c2w41-r
eddie       IN  CNAME  juno
jambox      IN  A      140.50.73.1
            IN  HINFO  machine_room c1e22
            IN  MX     0     jambox
postmaster  IN  MR     root
;
```

named.local

The named.local file specifies the network address for the local loopback interface. By convention, the loopback is usually named `localhost` and has network address `127.0.0.1`. You specify the name of this file in the boot file; its location is relative to the path name specified with the `directory` command. Data records in the file follow the standard resource record format.

Figure 64 shows an example of a named.local file.

Figure 64 Sample named.local file

```
; named.local

@   IN  SOA   pluto.acme.com.root.pluto.acme.com. (
              1.1       ; Serial
              3600      ; Refresh
              300       ; Retry
              3600000   ; Expire
              14400 )   ; Minimum
    IN  NS    pluto.acme.com.

1   IN  PTR   localhost.
```

named.rev

The named.rev (reverse) file specifies host addresses in the IN-ADDR.ARPA domain. This is a special domain for allowing internet address-to-name mapping. Because internet host addresses do not fall within domain boundaries, this special domain was formed to allow inverse mapping. The IN-ADDR.ARPA domain has four labels preceding it that correspond to the four octets of an internet address. You must specify all four octets even if an octet is zero. Internet address 128.32.0.4 is located in the domain 32.128.IN-ADDR.ARPA. This reversal of the address is awkward to read but allows for natural grouping of hosts in a network.

You specify the name of this file in the boot file; its location is relative to the path name specified with the `directory` command. Data records in the file follow the standard resource record format.

Local hosts will be listed in the named.rev file as well as the named.hosts file.

Figure 65 shows an example of a named.rev file. Data records in the named.rev file correspond to entries in the named.hosts file shown in Figure 63, page 130.

Figure 65 Sample named.rev file

```
; named.rev
;
@       IN  SOA    pluto.acme.com.  root.pluto.acme.com.(
                   1.1              ; Serial
                   10800            ; Refresh 3 hours
                   3600             ; Retry 1 hour
                   3600000          ; Expire 1000 hours
                   86400 )          ; Minimum 24 hours
        IN  NS     pluto.acme.com.
;
1.70    IN  PTR    saturn.acme.com.
2.70    IN  PTR    antares.acme.com.
4.70    IN  PTR    snoopy.acme.com.
100.70  IN  PTR    solo.acme.com.
101.70  IN  PTR    hercules.acme.com.
102.70  IN  PTR    harpo.acme.com.
106.70  IN  PTR    chocolate.acme.com.
107.70  IN  PTR    cumulus.acme.com.
108.70  IN  PTR    daffy.acme.com.
109.70  IN  PTR    shylock.acme.com.
113.70  IN  PTR    veggy.acme.com.
174.71  IN  PTR    inanna.acme.com.
177.71  IN  PTR    juno.acme.com.
1.73    IN  PTR    jambox.acme.com.
;
```

root.cache

The root.cache file is used to prime the name server's cache with addresses of authoritative name servers for the root domain of the network. You specify the name of this file in the boot file; its location is relative to the path name specified with the `directory` command. Data records in the file follow the standard resource record format.

Figure 66 shows an example of a root.cache file.

Figure 66 Sample root.cache file

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

                99999999  IN  NS NS.NIC.DDN.MIL.
                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).
; Order does not matter
;
NS.NIC.DDN.MIL.   99999999  IN  A  192.67.67.53
SRI-NIC.ARPA.     99999999  IN  CNAME  NS.NIC.DDN.MIL.
A.ISI.EDU.        99999999  IN  A  26.3.0.103
                  99999999  IN  A  128.9.0.107
BRL-AOS.MIL.      99999999  IN  A  128.20.1.2
                  99999999  IN  A  192.5.25.82
C.NYSER.NET.      99999999  IN  A  192.33.4.12
GUNTER-ADAM.ARPA. 99999999  IN  A  26.1.0.13
NS.NASA.GOV.      99999999  IN  A  128.102.16.10
                  99999999  IN  A  192.52.195.10
TERP.UMD.EDU.     99999999  IN  A  128.8.10.90
;
```

Time-to-live values in the example are set very high to ensure that data is not discarded because of a time-out.

Building a system with a name server

BIND software consists of two parts. A group of user interface routines, collectively called the `resolver`, resides in the C library, /lib/libc.a. The resolver builds query packets and exchanges them with the name server. The actual name server, `named`, is a daemon that runs in the background and services queries on a given network port. The standard port for UDP and TCP is specified in the /etc/services file.

Resolver routines in libc

The C library uses either name server resolver routines or host table lookup routines to resolve host names and addresses.

If the file /etc/use_nameserver exists, networking software uses the name server to resolve host names. Otherwise, the host table lookup method of name resolution will be used.

Setting up your domain

When you set up a domain that is going to be on a public network, your 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.

Contacts are as follows:

- **DARPA Internet**—Sites that are already on the DARPA Internet and need information on setting up a domain should contact `HOSTMASTER@NIC.DDN.MIL`. You may also want to be placed on the BIND mailing list, a mail group for people on the Internet who run BIND. The group discusses future design decisions, operational problems, and other related topics. To be added to the BIND mailing list, send a request to the following network address:

  bind-request@ucbarpa.Berkeley.EDU.

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

  An organization that already has a registered domain name should keep the CIC informed about how it would like its mail routed. In general, the CSNET relay prefers to send mail via CSNET (as opposed to BITNET or the Internet). For an organization on multiple networks, this may not always be

the preferred method. The CIC can be reached via electronic mail at `cic@sh.cs.net`, or by telephone at (617) 873-2777.

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

Name server configuration

This section describes procedures for configuring each type of name server and gives examples of files.

The exact steps you perform in configuring the name server depend upon the type of server—primary master, secondary master, caching-only, or resolving-only—that will run on the host on which you are installing it.

To configure a primary master, secondary master, or caching-only server, perform the following steps:

- Set up the boot file for your particular configuration.
- Create named.local.
- Add hosts to named.hosts and named.rev.
- Create resolv.conf.
- Add commands to the rc.local file to start the name server.
- Activate the name server.

To configure a resolving-only server, you need only create resolv.conf and activate the name server. The sections that follow describe the outlined tasks in detail.

Configuring a primary master server

Follow steps outlined in this section to configure a primary master server. The directory, /usr/lib/conf/bind/setup, contains scripts and files to help you install a primary master server.

Step 1 Change directory into /usr/lib/conf/bind/setup.

Step 2 Read the comments at the beginning of the Makefile, then edit the necessary definitions to tailor the file to your system.

Figure 67 shows the first few lines of the Makefile installed with the software.

Figure 67 Sample primary master server makefile

```
#
HOSTSRC       = cat /etc/hosts
DESTDIR       = /usr/etc/domain
DOMAIN        = berkeley.edu
HOSTADDR      = 128.32.140.6
HOSTNAME      = monet
ROOT          = root
#
```

Customize the following definitions:

HOSTSRC—Enter the name of the file containing host data. Set HOSTSRC to `cat /etc/hosts`.

DESTDIR—Enter the path name of the directory to contain all configuration files except named.boot. This is the path name you also specify in the boot file with the directory command.

DOMAIN—Enter the name of the default domain, for example, acme.com.

HOSTADDR—Enter the internet address of the local host.

HOSTNAME—Enter the name of the local host.

ROOT—Enter the login name of the network administrator.

Step 3 Create customized name server configuration files by entering the following commands:

```
make clean
make
```

These commands use the definitions in the Makefile to create customized named.boot, named.hosts, named.local, named.rev, and resolv.conf files from the host database contained in the /etc/hosts file.

Note

The Makefile is only an aid to creating the necessary files. After running it, look closely at the files it generates to ensure that all information is correct for your site.

Step 4 Ensure that `syslog` is logging `LOG_DAEMON` messages at the `LOG_DEBUG` level. Check /etc/syslog.conf; `named` error messages are probably directed to /usr/adm/log/daemonlog. If /etc/syslog.conf does not contain an entry for these messages,

configure `syslog` to log them. *Managing ConvexOS: Configuration Guide* contains detailed information on configuring the /etc/syslog.conf file.

Step 5 At this point, you are ready to install the configuration files and start up the name server. Enter the following commands (you must be running as `root` to complete this step):

```
make install
/usr/etc/named
```

If your system is running the `share` scheduler, the last command will be

```
slxqt network -x /usr/etc/named
```

Step 6 Use the `ps` command to verify that `named` is running.

Step 7 Check the `syslog` file for an entry indicating that `named` was restarted.

Step 8 Run the script, /usr/etc/named.reload, to cause the server to reload its database. You should see a message in the `syslog` file indicating that the server has been reloaded.

Step 9 Send the name server a `SIGINT` signal to cause it to dump its cache and database to the file /usr/tmp/named_dump.db. Verify that this file contains data that resembles configuration data in the named.hosts file.

Step 10 Try using the `ftp` command to communicate with a host you know is up and running and exists in your /etc/hosts file. Because the name server is not active yet, you should have no problem using `ftp`. If when you execute `ftp` to a known host, you get the initial message, "Connected to <hostname>," you can be reasonably sure of the connection.

Step 11 Activate name server usage by creating the file, /etc/use_nameserver. Use the `touch` command as follows:

```
touch /etc/use_nameserver
```

Step 12 Try using the `ftp` command again. The initial message should now read similar to "Connected to <hostname.domain>."

Step 13 To further verify that `named` is working, run the utility `nslookup`, with no options. To do so, enter

```
/usr/etc/nslookup
```

Try a few nslookup commands, as follows:

- Enter a few names of known hosts, one at a time. Make sure to try a few hosts that are in the /etc/hosts file and a few that are not. In response, you should receive the fully qualified host name and its internet address.

- Try using the `finger` command. The output displayed should be the same as if you had used `rlogin` on the host, then executed `finger`.
- Enter a command similar to

  `ls -d` *acme.com* **>** *output_file*

  to dump the database to *output_file*. This file should resemble the database dumped in step 9, above.
- Exit `nslookup` by typing **CTRL-D**.

Step 14 Try using `/usr/etc/ping` to test the connections to several different hosts, both with the standard name and the fully-qualified name.

Step 15 Send the server a `SIGIOT` signal to cause it to dump statistics to the file, /usr/tmp/named.stats.

Step 16 Set up your rc.local file to start `named` whenever the system is initialized.

Note

You must start `named` before starting any utilities that depend on its services.

Figure 68 shows a partial /etc/rc.local file containing commands to start the name server.

Figure 68 Starting named from rc.local

```
# redirect all output to the console
exec > /dev/console 2>&1
PATH=/etc:/usr/etc:/bin:/usr/ucb:/usr/bin:/usr/acme
export PATH

/bin/rm -f /etc/use_nameserver

/bin/hostname pluto.acme.com

if [ -f /bin/domainname ]; then
      /bin/domainname ""only set if yellow pages is used
fi

if [ -f /etc/ifconfig ]; then
      /etc/ifconfig ex0 `/bin/hostname` up arp netmask 0xffffff00
      #
      # all physical interfaces MUST be configured
      # before lo0 is configured
      #
      /etc/ifconfig lo0 localhost up
fi

#syslogd must be started before any other daemons
if [ -f /usr/etc/syslogd ]; then
      rm -f /dev/log
      /usr/etc/syslogd & echo 'starting system logger'
fi

# start up named now
if [ -f /etc/named.boot ]; then
      touch /etc/use_nameserver
      /usr/etc/named & echo -n ' started named'
fi
.
.
.
(rest of the file)
```

Note

To deactivate the name server, enter

```
rm -f /etc/use_nameserver
```

Figure 69 shows the typical contents of a named.boot file for a primary master server.

Figure 69 Boot file for a primary master server

```
; boot file for authoritative master name server
; for Berkeley.EDU. There should be one primary
; entry for each SOA record.
;
sortlist 10.0.0.0

directory    /usr/local/domain

; type       domain                source host/file

primary      Berkeley.EDU          berkeley.zone
primary      32.128.IN-ADDR.ARPA   berkeley.rev
primary      0.0.127.IN-ADDR.ARPA  localhost.rev
cache        .                     root.cache
;
```

Figure 70 shows the contents of a short named.hosts file for a primary master server.

Figure 70 named.hosts file for a primary master server

```
; named.hosts

@           IN      SOA     pluto.acme.com. root.pluto.acme.com. (
                    1.1     ; Serial
                    10800   ; Refresh 3 hours
                    3600    ; Retry 1 hour
                    3600000 ; Expire 1000 hours
                    86400 ) ; Minimum 24 hours
            IN      A       140.50.73.4
            IN      NS      pluto.acme.com.

localhost   IN      A       127.0.0.1
saturn      IN      A       140.50.70.1
            IN      HINFO   machine_room c1e22
cave        IN      CNAME   saturn
antares     IN      A       140.50.70.2
            IN      HINFO   machine_room c1e22
snoopy      IN      A       140.50.70.4
            IN      HINFO   covue_vax_ultrix c2w47-1
solo        IN      A       140.50.70.100
            IN      HINFO   dpz c1e70-r
postmaster  IN      MR      root
```

Figure 71 shows the named.rev file that corresponds to the named.hosts file in Figure 70.

Figure 71 named.rev file for a primary master server

```
; named.rev

@       IN    SOA pluto.acme.com. root.pluto.acme.com. (
              1.1      ; Serial
              10800    ; Refresh 3 hours
              3600     ; Retry 1 hour
              3600000  ; Expire 1000 hours
              86400 )  ; Minimum 24 hours
            IN     NS      pluto.acme.com.

1.70        IN     PTR     saturn.acme.com.
2.70        IN     PTR     antares.acme.com.
4.70        IN     PTR     snoopy.acme.com.
100.70      IN     PTR     solo.acme.com.
```

Figure 72 shows the contents of the named.local file for `pluto`.

Figure 72 named.local file for a primary master server

```
; named.local

@   IN  SOA   pluto.acme.com.root.pluto.acme.com. (
          1.1      ; Serial
          10800    ; Refresh 3 hours
          3600     ; Retry 1 hour
          3600000  ; Expire 1000 hours
          86400 )  ; Minimum 24 hours
    IN  NS        pluto.acme.com.

1   IN  PTR       localhost.
```

Configuring a secondary master server

Configuring a secondary master server requires fewer steps than configuring a primary master. Basically, all you have to do is set up the boot and data files and activate the server. Follow the procedure outlined below.

The directory, /usr/lib/conf/bind contains example files you can customize for your system.

Step 1 Create the file named.boot on directory /etc. As installed, your system may already have a copy of /etc/named.boot that you can modify. If not, copy the file from /usr/lib/conf/bind to /etc, then customize it for your server configuration. Figure 73 shows a typical named.boot file for a secondary master server.

Figure 73 Boot file for a secondary master server

```
; named.boot
;
; boot file for secondary name server
; There should be one primary entry for each SOA record.
;
sortlist 10.0.0.0

directory /usr/local/domain

; type     domain              source host/file        backup file

secondary Berkeley.EDU         128.32.137.8 128.32.137.3ucbhosts.bak
secondary 32.128.IN-ADDR.ARPA 128.32.137.8 128.32.137.3ucbhosts.rev.bak
primary   0.0.127.IN-ADDR.ARPAlocalhost.rev
cache     .                    root.cache
```

Step 2 Create the directory you specified in the boot file if it does not already exist.

Step 3 Copy the sample data files, named.local and root.cache, from /usr/lib/conf/bind to the directory you specified in the boot file.

Step 4 Modify the named.local file you copied into the data directory, changing the host name to the name of the host on which you are configuring the server. For example, the named.local file for the local host, `mars`, is shown in Figure 74.

Figure 74 named.local file for a secondary master server

```
; named.local

@   IN  SOA   mars.acme.com.root.mars.acme.com. (
          1.1      ; Serial
          10800    ; Refresh 3 hours
          3600     ; Retry 1 hour
          3600000  ; Expire 1000 hours
          86400 )  ; Minimum 24 hours
    IN  NS        mars.acme.com.

1   IN  PTR       localhost.
```

Step 5 Activate name server usage by creating the file, /etc/use_nameserver. Use the `touch` command as follows:

```
touch /etc/use_nameserver
```

Step 6 Modify the rc.local file to start named when the system is booted, as shown in Figure 68.

Note

To deactivate the name server, enter

```
rm -f /etc/use_nameserver
```

Configuring a caching-only server

To configure a caching-only server, follow the procedure outlined below.

Step 1 The directory /usr/lib/conf/bind contains example files you can customize for your system.

Create the file named.boot in directory /etc. As installed, your system may already have a copy of /etc/named.boot that you can modify. If not, copy the file from /usr/lib/conf/bind to /etc, then customize it for your server configuration. Figure 75 shows a typical named.boot file for a caching-only server.

Figure 75 Boot file for a caching-only server

```
; named.boot
;
; boot file for caching-only name server
;
directory    /usr/local/domain

; type       domain                 source host/file          backup file

cache .                             root.cache
primary      0.0.127.IN-ADDR.ARPA   named.local
secondary    acme.com               128.32.137.8 128.32.137.3 hosts.bak
secondary    32.128.IN-ADDR.ARPA    128.32.137.8 128.32.137.3 hosts.rev.bak
forwarders                          128.32.137.8 128.32.137.3
```

Step 2 Complete configuration by following steps 2-6 in the "Configuring a secondary master server" page 141.

Note

To deactivate the name server, enter

```
rm -f /etc/use_nameserver
```

Configuring a resolving-only server

To configure a resolving-only server, follow the procedure outlined below.

Step 1 Create the file, resolv.conf, on directory /etc. As installed, your system may already have a copy of /etc/resolv.conf that you can modify. If not, create it and add lines like those shown in Figure 76.

Figure 76 resolv.conf file for a resolving-only server

```
# /etc/resolv.conf
#
# designate the alternate name servers
#
nameserver   128.109.178.1
nameserver   129.109.193.1
nameserver   128.109.130.3
```

Step 2 Activate name server usage by creating the file, /etc/use_nameserver. Use the `touch` command as follows:

```
touch /etc/use_nameserver
```

Step 3 Complete configuration by modifying rc.local as described in the "Configuring a primary master server" page 135.

Note

To deactivate the name server, enter

```
rm -f /etc/use_nameserver
```

Adding hosts to the name server database

To add hosts to the database for a primary master server, follow the steps outlined below:

Step 1 Edit the named.hosts file as follows:

For each host, add the host name, all of its network addresses, its host information record, and its common aliases. The `WKS` records list optional well-known services.

Figure 77 shows the entry for the host, `calder`.

Figure 77 resolv.conf file for a resolving-only server

```
calder     IN  A       128.32.140.1
           IN  A       128.32.129.3
           IN  WKS     128.32.0.12   tcp telnet ftp smtp echo
           IN  WKS     128.32.0.12   udp echo time tftp
           IN  HINFO   jones
ucbcalder  IN  CNAME   calder
```

For the machine you are adding,

- Replace `calder` with the new host name.
- Replace `128.32.140.1` with the internet address of the new machine.
- If there are multiple addresses for the host, add the other address lines.

Step 2 Edit the named.rev file as follows:

For each address assigned to a machine, enter the address in reverse notation. For example, the lines for `calder` are as follows:

```
12.0          IN      PTR     calder.acme.com.
3.129         IN      PTR     calder.acme.com.
```

Note

You must include the final period on the host name.

Step 3 Increment the serial numbers on both the above files. If you use `rcs`, arrange for this to happen automatically when changes are checked in.

Step 4 Reload name server data by running the script, /usr/etc/named.reload.

When you have completed the steps above, the name server should be configured and running properly.

Configuring Serial Line Internet Protocol (SLIP)

16

The *Serial Line Internet Protocol* (SLIP) provides point-to-point communication over asynchronous serial lines. SLIP is commonly used on dedicated serial lines and dial-up lines. SLIP provides moderate network performance without the need for high-speed network hardware. Most standard network applications use SLIP transparently (though slowly) as they would a LAN.

This chapter describes the steps necessary to configure SLIP. Perform the procedure described here only if you intend to run TCP/IP over serial lines.

Prerequisites

Before performing the procedure below, you must establish a point-to-point connection between two hosts. A connection may be established via an autodial modem, a dedicated line, or by manually dialing the destination host.

TCP/IP protocols view SLIP as simply a network interface. As with a network interface, you must configure SLIP before running network utilities such as `ftp` over it. Use the procedure described below to configure the local end of the point-to-point connection. The other end of the serial line must also be connected to a SLIP implementation.

Configuring SLIP

To configure SLIP, perform the following tasks:

Step 1 Update the /etc/knetdctl.conf file by invoking `/etc/io2knetcf` with the `-slip` option.

```
io2knetcf -slip > /etc/knetd.conf
```

Step 2 Stop and restart the network as described in "Stopping and starting the STREAMS stack", page 92.

Step 3 Identify source and destination network interfaces.

Step 4 Attach a tty line to the source network interface.

Identifying interfaces

Add an entry to the host database for the interface at each end of the connection. To do so, update the /etc/hosts file and, optionally, configure the name server to provide the address.

Figure 78 shows an example of an /etc/hosts file with hosts defined for SLIP.

Figure 78 Sample /etc/hosts file for use with SLIP

```
# Host Database
#
127.1    localhost
#
# Serial Line Interface
#
100.50.0.1  jupiter-slip #local end of connection
#
140.55.0.10 sirius-slip #remote end of connection
#
```

Attaching tty lines

After you have associated a host name with each end of the SLIP connection, assign a tty to the local end by using `slattach`. You must ensure that `getty` is not running for the tty you have chosen. Turn off `getty` for that tty by entering **`off ttyXX`**.

Only the superuser, root, can use `slattach`. The syntax of the command is

`/etc/slattach` *ttyname src-name dst-name* [*baudrate*]

where

ttyname A string in the form `ttyXX` or `/dev/ttyXX`.

src-name Source (local) host name as defined in the host database.

dst-name Destination (remote) host name as defined in the host database.

baudrate Speed of the connection. The default is 9600.

For host names defined in the /etc/hosts file in Figure 78, the `slattach` command on `jupiter` would be:

```
/etc/slattach /dev/tty02 jupiter-slip sirius-slip 19200
```

For a permanent SLIP connection, you can add the `slattach` command to /etc/rc.local. Refer to the slattach(8) man page for information about additional options.

When you run `slattach`, the system responds with messages similar to the following:

```
slattach: unit sl0 assigned to /dev/tty02
slattach: sl0 at 19200 baud, remote sirius-slip (140.55.0.10) local
jupiter-slip (100.50.0.1)
```

At this point, if the remote system has brought the connection up, you should be able to use the link the same as any other network connection. Use the `ping` command to verify that the connection is up. For example,

```
ping sirius-slip
```

To detach the tty from the interface when you are through, kill the two `slattach` processes.

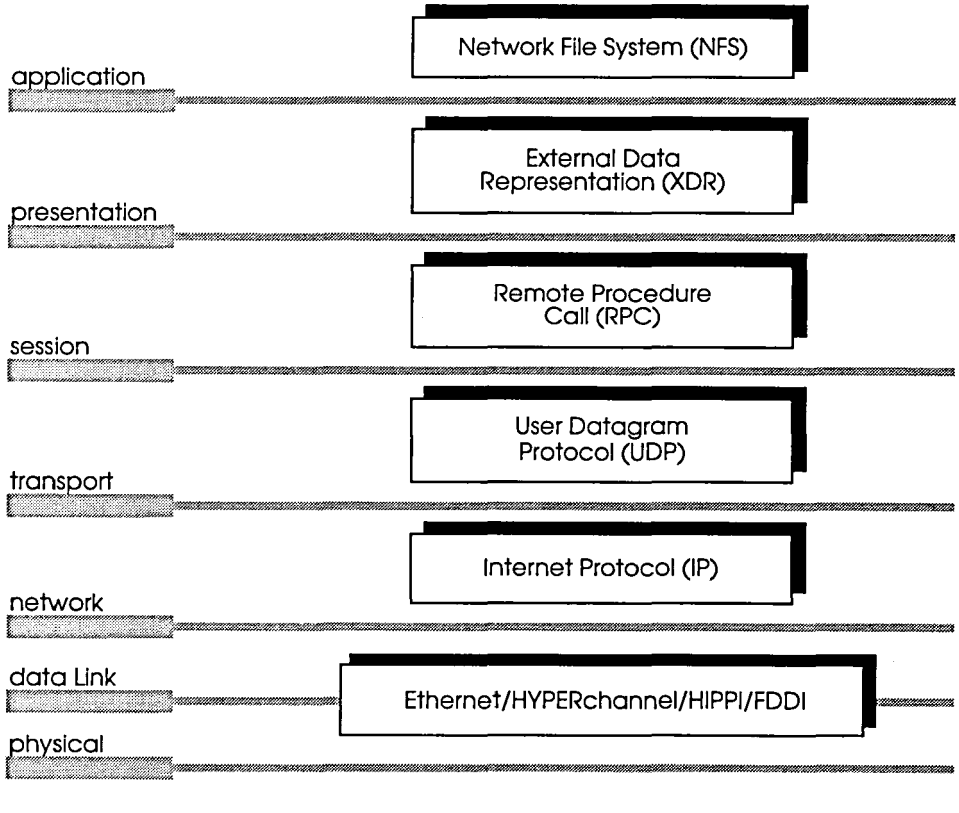
NFS mounts over SLIP

When using NFS over SLIP, the NFS read and write buffer sizes must be set to 2Kb or less to fit the SLIP window size. For example,

```
/etc/mount -o rw, rsize=2048 wsize=2048
sirius-slip: /mnt /mnt
```

You should set the options for permanent mounts in /etc/fstab.

Part 4 Managing NFS and NIS services



Introduction 17

The chapters in this part explain how to administer services provided by CONVEX Network File System (NFS) software.

Services provided by CONVEX NFS software

CONVEX NFS includes software to support the services listed below. You can configure hosts on your network to provide or to use some or all of these services, depending on the specific needs of your site.

- **Remote Procedure Call (RPC) facilities**—Enable client processes to have another process, often running on a different host, execute a procedure call as if the caller had executed the procedure call in its own address space.
- **`portmap`**–A daemon that maps port numbers to RPC program numbers, providing a standard way for clients to look up the port number of RPC-based programs supported by a server.
- **Network File System (NFS)**—A distributed file system that enables users to access remote file systems as if they were local.
- **Secure NFS**—An authentication system that enhances the security of network environments.
- **The automounter**—A utility that automatically mounts remote file systems when they are accessed and unmounts them when they have not been accessed for a period of time.
- **NIS**—A distributed lookup system that centralizes access to information in certain network configuration files otherwise maintained on each host.
- **NETdisk**—A collection of administrative programs that enables a CONVEX server machine to support booting diskless workstations using NFS.

- **Lock manager**—Facilities that enable cooperating processes to implement record locking.
- **REX**—Utility used to execute commands on a remote system.

Managing NFS and NIS services

Managing NFS requires the system manager to perform some or all of the following tasks:

- Set up a basic NFS configuration
- Set up NETdisk facilities
- Set up the automounter
- Set up NIS
- Set up the NFS authentication system
- Start the `portmap` daemon
- Set up the lock manager

Before you begin

This guide provides instructions for configuring and managing NFS and NIS services; it does not discuss installing the software.

Before attempting any procedure described in this part, you must successfully complete the following tasks, in the order in which they are listed:

- Install ConvexOS and Utilities
- Install CONVEX Internet Services
- Install CONVEX NFS
- Configure Internet Services

For information about installing the products listed above, refer to the installation procedures that accompanied the software distribution tape.

For information about configuring Internet Services, refer to Part 2, "Part 3 Configuring Internet Services," page 61.

Where to find more information

This guide focuses on the "how to" of NFS system management—the tasks that you as the system manager perform to set up, maintain, and troubleshoot NFS services. This book does not explain fundamental networking concepts and theories, or the design and implementation of the software. It also does not attempt to provide you with all the information you may need to help you decide whether or not to configure a particular service on a particular host.

Because NFS and TCP/IP networking are mature technologies, you can find information about the "real-world" uses and inner workings of NFS and other network services in the technical section of your local book store. Anyone responsible for managing NFS software should acquire and peruse at least one of these off-the-shelf books.

Managing NFS and NIS, by Hal Stern (O'Reilly & Associates, 1992. ISBN 0-937175-75-7).

You can find a concise introduction to basic networking concepts and terminology, an overview of CONVEX networking products, and a description of the CONVEX implementation of NFS services in *CONVEX Networking Concepts*.

Managing portmap

18

The portmapper (portmap) is a network service that maps port numbers to RPC program numbers. It provides a standard way for a client to look up the port number of RPC-based programs supported by a server. CONVEX Network File System (NFS) and the ConvexOS tape system use portmapper.

When an RPC server starts up, it tells portmap what port number it is listening to, and what RPC program numbers it is prepared to service. When a client wishes to make an RPC call to a given program, it first contacts portmap on the server to determine which port receives RPC packets.

Any system that provides RPC services must have portmap running before starting any other RPC daemons. Because the ConvexOS tape system relies on portmap, you must ensure that portmap is running, even if your system does not run NFS or NIS.

This chapter explains how to configure portmap and obtain status information about RPC services.

Configuring portmap

This section explains how to ensure that portmap is started at boot time.

Step 1 Log in as superuser.

Step 2 The /etc/rc.local file installed with ConvexOS should include lines to start portmap; verify this. Figure 79 shows a partial rc.local startup file. The lines in italics start portmap.

Figure 79 Starting portmap from rc.local

```
# build the networking streams stack
if [ "`/etc/knetdctl -q`" = "knetd not configured" ]; then
       /etc/knetdctl -c /etc/knetd.conf
       /etc/knetdctl -r
fi

# configure interfaces
/etc/ifconfig eth0 bozo arp up netmask 0xffffff0
/etc/ifconfig fddi0 bozo-f arp up netmask 0xffffff00

# start portmap
if [ -f /usr/etc/portmap ]; then
       portmap;     echo -n ' portmap'
fi

# start NFS daemons
if [ -f /etc/biod ]; then
       /etc/biod 4 & echo -n ' biod'
fi
if [ -f /etc/nfsd -a -f /etc/exports -a -f /usr/etc/exportfs ]; then
       /etc/nfsd 4 & echo -n ' nfsd'
       >/etc/xtab; /usr/etc/exportfs -a &
fi

/etc/umount -at nfs
/etc/mount -vat nfs
```

If the italicized lines are not in your rc.local file, use your favorite editor to add them, then reboot the system.

Step 3 After rebooting the system, verify that portmap is running by entering

```
ps ax | grep portmap
```

The system should respond with a line similar to

```
56 ? S 0:56 /etc/portmap
```

Obtaining status of `portmap` and RPC services

Use the `rpcinfo` utility to

- Query a server about the programs registered with its `portmap`.
- Verify that a remote machine is accepting and responding to RPC requests.

You can use `rpcinfo` to get status information about the local host or a remote host by entering

`/usr/etc/rpcinfo -p` *hostname*

Supply *hostname* for information about a remote host; omit this parameter for information about the local host. `rpcinfo` displays information about all registered RPC services.

Information reported on each service includes

- RPC number assigned to the service
- Version number of the service
- Protocol supported
- IP port used by the service
- RPC service name

Figure 80 shows sample `rpcinfo` output for the local host as requested by the **-p** option.

Figure 80 Checking RPC services with `rpcinfo`

```
% rpcinfo -p
 program   vers proto   port
    100000    2   tcp    111  portmapper
    100004    1   udp    670  ypserv
    100007    2   tcp   1031  ypbind
    100004    1   tcp    671  ypserv
    100007    1   udp   1025  ypbind
    100009    1   udp   1023  yppasswdd
    100003    2   udp   2049  nfs
    100024    1   udp    724  status
    100024    1   tcp    726  status
    100021    1   tcp    727  nlockmgr
    100020    1   tcp    732  llockmgr
    100021    2   tcp    735  nlockmgr
 536920065    1   tcp    873  tpdaemon
    100002    1   udp   1206  rusersd
 536920065    2   tcp    873  tpdaemon
    100002    2   udp   1206  rusersd
    100001    1   udp   1230  rstatd
    100001    3   udp   1230  rstatd
    100008    1   udp   1245  walld
    100011    1   udp   1246  rquotad
    100012    1   udp   1041  sprayed
    100017    1   tcp   1038  rexd
    100005    1   udp   1293  mountd
    100026    1   udp   1297  bootparam
    100005    1   tcp    938  mountd
```

When a remote `portmap` has died or is not accepting connections, `rpcinfo` times out while attempting to reach it and reports an error.

Managing NFS

19

The Network File System (NFS) enables users to access remote file systems as if they were local. As the administrator of a system running NFS, your goal is to set up a distributed file system that is completely transparent to users. Users should be able to directly access files without knowing where the data actually resides.

With the network services provided by NFS, network file administration is no more difficult than administration of a set of local files on a time-sharing system.

This chapter guides you through a set of step-by-step tasks for setting up and running NFS on a ConvexOS system. These tasks include

- Setting up a host to function as an NFS server
- Setting up a host to function as an NFS client
- Performance tuning
- Removing temporary files created by NFS

You can find information about troubleshooting NFS in Chapter 26, "Troubleshooting NFS and NIS," page 321.

Setting up an NFS server

An NFS server is a host that exports directories in its local file system, making them available to client hosts on the network. NFS servers control who may mount a directory by limiting named file systems to approved clients. Options can be used to further limit clients to particular types of access, such as read-only.

Any host that has a local file system can be configured as both an NFS server and an NFS client. An NFS server can be a client of another NFS server, but no server can act as an intermediary between a client and another server. That is, a server cannot export directories that it has remote mounted from another server.

This section contains instructions for performing the following tasks:

- Initially configuring an NFS server
- Identifying file systems to be exported at boot time
- Exporting and unexporting file systems on demand

Initially configuring an NFS server

This section describes daemons that provide NFS server functionality and gives instructions for verifying that all necessary daemons are started at boot time. Information in this section is of interest only if you are configuring an NFS server for the first time.

Before attempting to configure a host as an NFS server, you must successfully complete the following tasks:

- Install ConvexOS and Utilities
- Install CONVEX Internet Services
- Install CONVEX NFS
- Configure Internet Services

For information about installing the products listed above, refer to the installation procedures that accompanied the software distribution tape. For information about configuring Internet Services, refer to Part 2, "Part 3 Configuring Internet Services," page 61.

After completing the tasks listed above, your system should be ready to function as an NFS server—all you need to do is identify the file systems to be exported; however, we recommend that you verify that necessary daemons will be started at boot time. A brief description of each NFS server daemon follows.

Three daemons provide NFS server functionality.

- **nfsd**—Receives file access requests from clients, performs the actual file system operation, and sends a response back to the client. If the client request is for a read operation, requested data is also returned.

  An `nfsd` processes client requests one at a time. Completion of a request often involves delays, such as waiting for data to be retrieved from disk. While a server processes a request, it ignores all others, causing other clients to retransmit requests until the server becomes free.

  You can avoid this potential bottleneck by running multiple `nfsd` daemons. The number of `nfsd` daemons running on the server should correspond to the number of `biod` daemons running on its clients. Four and eight are popular numbers.

- **mountd**—Receives client mount requests and reads the server's xtab file to determine which clients are permitted to mount which file systems. If the client has permission to mount the requested file system, `mountd` returns a file handle to the client.

- **portmap**—The `portmap` daemon is the means by which clients discover the port number of a service. `portmap` maps the RPC program number of a service to a port number, and provides port numbers in response to queries by clients.This port number then becomes part of the IP address clients use to request a particular service.

  Because NFS servers communicate with clients through RPC, you must ensure that the `portmap` daemon is running before you start NFS server daemons. For more detailed information about the service provided by `portmap` and instructions for running it, refer to Chapter 18, "Managing portmap," page 157.

Note

In addition to `nfsd`, `mountd`, and `portmap`, an NFS server may need to run record locking and automounting daemons. Refer to the Lock Manager and Automounter chapters.

Follow these steps to verify that NFS server daemons will be started at boot time:

Step 1 Log in to the host you intend to use as an NFS server and become superuser.

Step 2 To ensure that server daemons are started automatically whenever the server is booted, examine the rc.local script for lines similar to those shown in Figure 81.

Figure 81 Starting NFS server daemons from rc.local

```
        .
        .
        .
if [ -f /usr/etc/portmap ]; then
        portmap;            echo -n ' portmap'
fi
        .
        .
        .
if [ -f /etc/exports ]; then
        > /etc/xtab
        exportfs -a
        nfsd 4 &            echo -n ' nfsd'
        rpc.mountd
fi
```

Step 3 If the server's rc.local script does not contain the lines shown in Figure 81, use your favorite editor to add them. Any changes you make to the rc.local script will take effect when the server is rebooted, causing rc.local to run.

Step 4 To start `portmap` and `nfsd` without rebooting, enter

```
/usr/etc/portmap
/usr/etc/nfsd 4
```

Step 5 The inetd.conf file installed with the software should contain an entry for `mountd`. This entry is shown in boldface in the partial inetd.conf file in Figure 82.

Figure 82 Example inetd.conf file

```
 ftp        stream  tcp    nowait   root    /usr/etc/in.ftpd             ftpd
telnet      stream  tcp    nowait   root    /usr/etc/in.telnetd          telnetd
shell       stream  tcp    nowait   root    /etc/in.rshd                 rshd
login       stream  tcp    nowait   root    /etc/in.rlogind              rlogind
exec        stream  tcp    nowait   root    /usr/etc/in.rexecd           rexecd
syslog      dgram   udp    wait     root    /usr/etc/in.syslog           syslog
comsat      dgram   udp    wait     root    /usr/etc/in.comsat           comsat
talk        dgram   udp    wait     root    /usr/etc/in.talkd            talkd
crashdump   dgram   udp    wait     root    /usr/etc/in.crashreceive     crashrcv
rusers      dgram   udp    wait     root    1-2 /usr/etc/rpc.rusersd     rusersd
rup         dgram   udp    wait     root    1-3 /usr/etc/rpc.rstatd      rstatd
rwall       dgram   udp    wait     root    1 /usr/etc/rpc.rwalld        rwalld
mount       dgram   udp    wait     root    1 /usr/etc/rpc.mountd        mountd
quota       dgram   udp    wait     root    1 /usr/etc/rpc.rquotad       rquotad
spray       dgram   udp    wait     root    1 /usr/etc/rpc.sprayd        sprayd
rex         stream  tcp    wait     root    1 /usr/etc/rpc.rexd          rexd
```

Examine your server's inetd.conf file. If it does not include an entry for `rpc.mountd`, use your favorite editor to add the boldfaced line shown in Figure 82.

Any changes you make to the inetd.conf file will take effect at boot time, when `inetd` restarts.

Once you have verified that the server's rc.local and inetd.conf files are set up correctly, you need to identify the file systems the server will export. The sections that follow present instructions for exporting file systems.

Exporting file systems

You can use either or both of the following methods to cause an NFS server to export file systems:

- Identify file systems to be exported at boot time by including entries in the /etc/exports file.
- Export and unexport file systems explicitly with the `exportfs` command.

Export options

Regardless of which exporting method you use, a set of *export options* controls the way in which NFS clients access exported directories. You can specify export options within entries in the exports file and on the `exportfs` command line. If you do not provide export options, exported directories are available for unrestricted use by any NFS client on the network. Export options are summarized as follows:

`ro`	Export directory read-only. (Default is read/write.)
`async`	Export directory asynchronously. (Default is synchronously.) Refer to "Operating NFS asynchronously," page 167, for more information.
`rw`[=*client*[:*client*]]...	Give read/write access to specified client(s) only. All other clients have read-only access. If no clients are specified, all clients are given read/write access. You can specify a maximum of 200 clients with this option.
`access`=*client*[:*client*]...	Limit access to named client(s).
`root`=*client*[:*client*]...	Grant root access to named client(s). You can specify a maximum of 200 clients with this option. Refer to "Allowing over-the-net root access," page 168. (Default is for no clients to be granted root access.)
`anon`=*uid*	If a request comes from an unknown user, use *uid* as the effective user ID. Refer to "Allowing over-the-net root access," page 168.
`secure`	Require clients to use Secure RPC when accessing this directory.

In the above syntax, *client* can be either a host name or a *netgroup*. The `exportfs` utility first checks for the named client in the hosts file, then in the /etc/netgroup file. To use netgroups, the server must run NIS. Refer to Chapter 23,"Managing NIS," page 247, for complete information about setting up the netgroup file.

Options that require a more complete explanation are discussed in the following sections.

Operating NFS asynchronously

This section describes NFS operation on file systems exported with the `async` option specified.

NFS servers normally operate synchronously—data is written to permanent storage before an NFS transaction is acknowledged. Synchronous operation enables the server to maintain file integrity despite crashes. In synchronous operation, clients do not distinguish between a server crash and slow server response; they simply retry until the transaction completes.

The benefit of synchronous operation is increased reliability; however, reliability is gained at a cost in performance. Each time a client makes a write request, the server writes not only the data buffer, but "in-core" copies of the inode and indirect blocks as well. When NFS operates asynchronously, making full use of the ConvexOS buffer cache, large files can be written several times faster.

Note

Before attempting asynchronous NFS operation, carefully consider the NFS environment and its ability to recover from inconsistent data if a server fails. Weigh carefully your need for higher performance against inconvenience and administrative effort required to recover when the server crashes. Be aware that clients may not detect the server's crash and may not be aware that data has been lost.

You enable asynchronous operation on a file system basis by specifying the `async` option in the exports file. For example, adding the following lines to the exports file enables asynchronous operation for /tmp and /usr/spool:

```
/tmp        -async
/usr/spool -access=genghis:attila:leona,async
```

File systems served by asynchronous servers should be soft mounted, so that NFS returns an error if the server malfunctions. By using `mount` command options, you can set the time-out and retry count values so that the operation times out and returns an error before the server can be rebooted; default values for the time-out and retry `mount` options ensure this.

Refer to the section, "Mounting remote file systems," page 177, and to the mount(8) and fstab(5) man pages for more information about the use of `mount` options.

Allowing over-the-net root access

Under NFS, a server exports file systems it owns so clients may remotely mount them. When a client becomes superuser, it is by default denied permission on remote-mounted file systems, as shown in the following example:

```
% cd
% touch test1 test2
% chmod 777 test1
% chmod 700 test2
% ls -l test*
-rwxrwxrwx  1 jsbach       0 Mar 24 16:12 test1
-rwx------  1 jsbach       0 Mar 24 16:12 test2
```

Now, retry it as root.

```
% su
Password:
# touch test1
# touch test2
# touch: test2: Permission denied
# ls -l test*
-rwxrwxrwx  1 jsbach       0 Mar 21 16:16 test1
-rwx------  1 jsbach       0 Mar 21 16:12 test2
```

The problem usually appears during the execution of a set-uid root program. Programs that run as root cannot access files or directories unless "others" have permission to do so.

Also, you cannot change ownership of remote-mounted files. Because users cannot do a chown command and root is treated as a normal user on remote access, only root on the server can change the ownership of remote files. For example, as yourself, you attempt to change ownership of a new program, a.out, that must be set-uid root. The chown command fails, as demonstrated in the following example:

```
% chmod 4777 a.out
% su
Password:
# chown root a.out
a.out: Not owner
```

To change ownership, you must log in to the server as root, then make the change. Or, you can move the file to a file system owned by your machine (for example /usr/tmp is always owned by the local machine) and make the change there.

If you are prepared to accept serious security risks, you may also solve this problem by enabling over-the-net root access. Enabling over-the-net root access allows client root accounts superuser privileges on remotely-mounted file systems. You can use export options to enable over-the-net root access on a file-system-by-file-system basis. Before you consider using this method, however, carefully consider the security risks.

Note

CONVEX does not recommend enabling over-the-net root access as discussed in the following paragraphs.

You can enable over-the-net root access for particular file systems by using the exports options, **-anon=0** or **-root=** (refer to "Export options," page 166). Export options can be specified either in the exports file or on the `exportfs` command line. Use export options as follows:

- To enable over-the-net root access to particular hosts on the access list, use the **-root=***hostname* option. In the following exports file entry, root access is given to convex2, but not to convex3:

```
/usr    -root=convex2,access=convex2:convex3
```

- To enable over-the-net root access for all hosts, specify the **-anon=0** option for each file system to be accessed by root. In the following example, the /usr file system has been modified to allow over-the-net root access by convex2 and convex3:

```
/usr          -anon=0,access=convex2:convex3
/fonts        -access=convex4
/usr/spool    -access=convex2:convex3:convex4
```

- To enable read-only access, specify the **-ro** option. In the following example, over-the-net root access is mapped to UID -2 for every file system listed (because the default value is -2). No user, including root, can write the files in /usr/src. Although users can mount /usr/src for reading or writing, attempts to write by any user, including root, fail with a EROFS (Read-Only File System) error.

```
/usr/src      -ro,access=convex2:convex3
```

- To enable read-write access only on selected machines, use the **-rw=***hostname* option. For example, the following lines specify that convex3 has read-write privileges, but not convex2:

```
/usr/src      -rw=convex3,access=convex2:convex3
```

Identifying file systems to be exported at boot time

The start-up script, rc.local, runs the `exportfs` utility at boot time. `exportfs` reads the exports file, makes each listed directory available for remote mounting by approved NFS clients, and stores a list of currently exported file systems in the xtab file.

To cause the server to export file systems at boot time, you must create or modify the exports file, which identifies file systems to be exported, clients that may access exported file systems, and any restrictions on that access.

Follow these steps to export file systems from an NFS server at boot time:

Step 1 Log in to the host you intend to use as an NFS server and become superuser.

Step 2 Using your favorite editor, create an entry in the exports file for each directory you want the server to export. (Because the exports file is not installed with NFS software, you must create it if you are setting up an NFS server for the first time on a given host.) The format of the exports file is

directory [*-export_option*, *export_option*,...]

where *directory* is the mount point path name of the directory you want to export. *directory* must be local to the NFS server. You cannot export either a parent directory or a subdirectory of an exported directory within the same file system. For example, it would be illegal to export /usr and /usr/local if both directories reside on the same disk partition.

Options specified in the exports file control the way in which NFS clients may access exported directories. If you do not provide options in the exports file, exported directories are available for unrestricted use by any NFS client on the network. If present, the list of options is preceded by a dash; options are separated by commas.

Figure 83 shows a few typical exports file entries.

Figure 83 Typical exports file entries

```
/usr/local -ro                          # export read-only to all clients
#
/usr2       -access=saki:sushi:boris# limit access to named hosts
#
/home       -root=daisey:odie           # allow read/write access by any client
                                        # grant root access to named hosts
#
/share/suns-rw=admin:ops,-secure        # export to all clients, but limit
                                        # write access to named hosts
                                        # and require use of Secure RPC
#
```

In the example in Figure 83

- Any NFS client can read /usr/local, but none can write to it.
- Only saki, sushi, and boris can access /usr2.
- Any NFS client can read and write /home, but only daisey and odie have root access.
- Any NFS client can read /share/suns, but only admin and ops can write to it. All clients must use Secure RPC when accessing this directory.

Refer to the exports(5) man page for more information about this file.

Note

Changes made to the exports file will not take effect until you either reboot the server or run `exportfs` from the command line, as described in the following section.

Exporting and unexporting files on demand

This section describes the `exportfs` command and provides examples for using `exportfs` to export or unexport file systems on demand.

Syntax of the `exportfs` command is

`exportfs` **`[-avuif]`** **`[-o`** *export-options*`]` [*dir*]

where

`a` Export all directories listed in the exports file. If you also specify **`-u`**, unexport all currently exported files.

v Run in verbose mode, printing the name of each directory as it is exported or unexported.

u Unexport the indicated directories.

i Ignore options found in the exports file. Used to override exports file options with options specified on the `exportfs` command line.

f Force the normally illegal export of a parent or child of a directory that is already exported.

o [*export-options*]
Identical to the exports file options (refer to "Export options," page 166). *export-options* can be used to override options in the exports file, or applied to the *dir* specified on the command line.

dir Directory you wish to make available to NFS clients.

You can run `exportfs` at any time to export or unexport file systems on demand. For example, say you have just added entries to the exports file and you want to give clients access to the additional file systems without waiting for the server to be rebooted. You can cause the server to immediately export all directories listed in the updated exports file by entering

```
exportfs -a
```

In addition to forcing a modified exports file to take effect, there are other situations in which you may want to run `exportfs` from the command line. For example,

- To make a directory not listed in the exports file available immediately and without rebooting, enter

  exportfs -o *options dir*

- To change the options of a currently exported directory, enter

  exportfs -i -o *overridden.options dir*

- To unexport a directory, enter

  exportfs -u *dir*

- To unexport all directories, enter

  exportfs -au

Because `exportfs` stores in the xtab file the list of directories available to NFS clients, after running **`exportfs -a`**, your xtab file is identical to your exports file. Later, if you run `exportfs` from the command line and specify a directory, the xtab file will reflect the changes you made. For example, say you reboot your system, causing `exportfs` to make available the directories listed in the example exports file shown in Figure 83. Your xtab file is now identical to that in Figure 83.

Later, a user on an NFS client requests read-only access to /usr/bin/games, so you run `exportfs` from the command line by entering

```
exportfs -o ro /usr/bin/games
```

Your xtab file now contains all entries found in the exports file, plus an entry for /usr/bin/games, as shown in Figure 84.

Figure 84 /etc/xtab file

```
/usr/local -ro                          # export read-only to all clients
#
/usr2       -access=saki:sushi:boris# limit access to named hosts
#
/home       -root=daisey:odie           # allow read/write access by any client
                                        # grant root access to named hosts
#
/share/suns-rw=admin:ops,-secure        # export to all clients, but limit
                                        # write access to named hosts
                                        # require use of Secure RPC
#
/usr/bin/games-ro
```

Note

Any time you run `exportfs -a`, the contents of the exports file replace the contents of the xtab file, overwriting any xtab file entries you created by running `exportfs` from the command line. Also, remember that any xtab entries you create by running `exportfs` from the command line are overwritten when the system is rebooted.

Refer to the exportfs(8), exports(5), and xtab(5) man pages for more information about the use of the `exportfs` command.

Setting up an NFS client

An NFS client is a host that accesses remote directories exported by one or more NFS servers.

Any host that has a local file system can be configured as both an NFS server and an NFS client. An NFS server can be a client of another NFS server, but no server can act as an intermediary between a client and another server. That is, a server cannot export directories that it has remote mounted from another server.

This section contains instructions for performing the following tasks:

- Initially configuring an NFS client
- Creating local mount points for remote directories
- Mounting remote directories

The sections below describe the steps needed to complete these tasks.

Initially configuring an NFS client

This section describes daemons that provide NFS client functionality, and gives instructions for verifying that all necessary daemons are started at boot time. Information in this section is of interest only if you are configuring a system as an NFS client for the first time.

Before attempting to configure a host as an NFS client, you must successfully complete the following tasks:

- Install ConvexOS and Utilities
- Install CONVEX Internet Services
- Install CONVEX NFS
- Configure Internet Services

For information about installing the products listed above, refer to the installation procedures that accompanied the software distribution tape. For information about configuring CONVEX Internet Services, refer to Part 2, "Part 3 Configuring Internet Services," page 61.

After you complete the tasks listed above, your system should be ready to function as an NFS client—all you need to do is identify the file systems to be remotely mounted; however, we recommend that you verify that necessary daemons will be started at boot time.

Two daemons provide NFS client functionality.

- **biod**—Processes client requests for block I/O operations. biods process one request at a time; while a client request is being processed, all others are ignored. Because completion of a request often involves delays, such as waiting for data to be retrieved from the remote system, it is recommended that you run multiple biods. Four and eight are popular numbers.
- **portmap**—The portmap daemon is the means by which clients discover the port number of a service. portmap maps the RPC program number of a service to a port number, and provides port numbers in response to queries by clients.This port number then becomes part of the IP address clients use to request a particular service.

  Because NFS clients communicate with servers through RPC, you must ensure that the portmap daemon is running before you start client daemons. For more detailed information about the service provided by portmap and instructions for running it, refer to Chapter 18, "Managing portmap," page 157.

Note

In addition to biod and portmap, an NFS client may need to run record locking and automounting daemons. Refer to the Lock Manager and Automounter chapters.

Follow these steps to verify that NFS client daemons will be started at boot time:

Step 1 Log in to the host you intend to use as an NFS client and become superuser.

Step 2 To ensure that client daemons are started automatically whenever the client is booted, examine the rc.local script for lines similar to those shown in Figure 81.

Figure 85 Starting NFS client daemons from rc.local

```
if [ -f /usr/etc/portmap ]; then
        portmap;            echo -n ' portmap'
fi

if [ -f /usr/etc/biod ]; then
        /etc/biod 4 & echo -n 'biod'
fi
```

If these lines are not in rc.local, use your favorite editor to add them. Any changes you make to the rc.local script will take effect when the client is rebooted, causing rc.local to run.

Step 3 To start `portmap` and `biod` without rebooting, enter

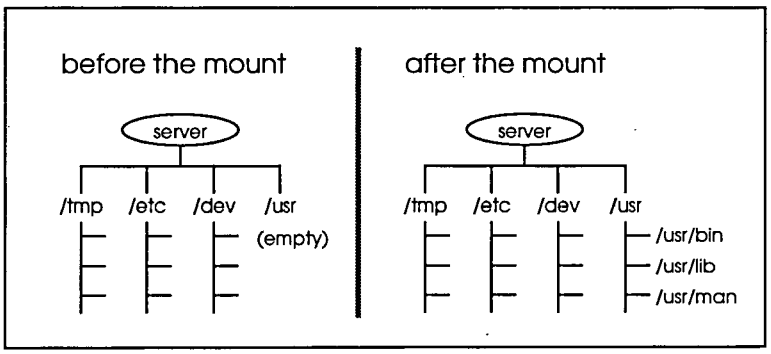
```
/usr/etc/biod 4
```

Once you have verified that the client's rc.local file is set up correctly, you need to identify the file systems the client will remotely mount. The sections that follow present instructions for mounting remote file systems.

Creating mount points

Before an NFS client can mount a remote directory, a *mount point* for that directory must exist in the local file system. A mount point is the location in the local directory structure at which the remote directory is accessed once it is mounted. For example, in Figure 86, /usr is the mount point at which remote directories /usr/bin, /usr/lib, and /usr/man are accessed.

Figure 86 Effect of remote mounts on the client file system



Follow these steps to create NFS mount points:

Step 1 Log in to the host you intend to use as an NFS client and become superuser.

Step 2 Use the `mkdir` command to create mount points for all directories to be remotely mounted from NFS servers. For example, to create the mount point for the remote directory structure shown in Figure 86, enter

```
cd /
mkdir usr
```

Step 3 Use the `chmod` command on the mount point to enable read and execute permissions for others (drwxrwxr-x). If you do not, attempts to determine the current working directory with the `pwd` command in the mounted file system may fail.

Mounting remote file systems

An NFS client can mount any exported file system if

- the client can communicate with its server over the network.
- the client is granted access to the file system in the server's /etc/exports file.

NFS clients access remote directories through any combination of the following three methods:

1. **Automatic mounting at boot time**—At boot time, a client mounts remote directories to which it needs access by including specific entries in its fstab file.
2. **Explicit mounting on demand**—A superuser on the client can explicitly mount and unmount remote directories at any time with the `mount` command.
3. **Automatic mounting when accessed**—When a user or a process on a client running the `automount` utility accesses a remote file or directory, `automount` mounts the hierarchy to which that file or directory belongs. The file or directory remains mounted for as long as it is needed; it is automatically unmounted if it is not accessed within a predefined time interval.

You can configure an NFS client to use any or all of these methods, depending on factors such as the frequency with which a directory is used by remote systems. In all likelihood, you will mount some remote directories at boot time, others with the `mount` command, and others via the `automount` utility. Because users' needs for access to remote directories change over time, you will need to periodically reassess your mounting strategy.

As a general rule,

- Use the fstab file to automatically mount directories to which networked hosts need more or less continuous access. For example, if hosts on your network share utilities located in /usr/local/bin, this would be a good candidate for automatic mounting at boot time.
- Directories used sporadically can be explicitly mounted and unmounted with the `mount` command.
- Because the `automount` utility eliminates the need for frequently updating the fstab files on your networked clients, automounting is the method of choice for handling large, complex networked file systems.

The following sections outline procedures for setting up an NFS client to mount remote directories with the `mount` command, whether automatically at boot time or explicitly on demand. Procedures for setting up the `automount` utility are explained in Chapter 22, "Using the automounter utility," page 217.

Setting up the fstab file

Follow these steps to mount remote file systems:

Step 1 Log in to the host you intend to use as an NFS client and become superuser.

Step 2 Using your favorite editor, create an entry in the fstab file for each remote directory you want the server to mount at boot time. The format of the fstab file is

fsname dir **`nfs`** [*mount_options*] *frequency passno*

where

fsname Name of remote file system to be mounted at mount point *dir*. For NFS file systems, *fsname* takes the form *server:path*. *server* is the name of the NFS server on which file system *path* resides.

dir Full path name of the local directory on which the remote file system will be mounted.

`nfs` Specifies NFS-type file system is to be mounted.

mount_options
Most fstab file options are identical to `mount` command *mount_options*. Refer to the list of mount options in Table 3 page 180, or to the mount(8) man page for information about *mount_options*.

frequency Specifies the dump interval for this directory, in days.

passno Indicates on which pass of the consistency checking program, `fsck`, this file system should be checked.

Figure 87 shows a few typical fstab entries for NFS-served file systems.

Figure 87 fstab entries for NFS file system mounts

```
fred:/usr2 /usr2 nfs rw,hard 0 0
barney:/usr/man /usr/man nfs rw,soft 0 0
bambam:/usr/docs /mnt nfs rw,intr,bg 0 0
```

Step 3 For all file systems listed in the fstab file to be mounted at boot time, the rc.local startup script needs to run the `mount -a` command. Because the `mount` command is also used to mount local file systems, the client's rc.local script should already contain commands to mount all file systems. Just to be sure, examine the client's rc.local script, searching for the following command sequence:

```
/etc/umount -at nfs
/etc/mount -vat nfs
```

The `umount` command ensures that all NFS file systems are unmounted prior to attempting to mount them. Add these commands to rc.local if necessary.

Step 4 If you wish to mount the file systems listed in the fstab file without rebooting the client; for example, after modifying the fstab file, enter the following commands:

```
/etc/umount -at nfs
/etc/mount -vat nfs
```

Step 5 To verify that you have successfully mounted a file system where you expected to, use either the `df` or `mount` commands without arguments. Both display information about currently mounted file systems, but in different formats.

Using the `mount` command

Use the `mount` command to mount a file system on demand. File systems mounted explicitly with the `mount` command remain mounted until you explicitly unmount them or reboot the client.

Step 1 Log in to the host you intend to use as an NFS client and become superuser.

Step 2 Mount remote file systems by entering `mount` commands according to the following syntax:

mount [**-afnvpF**] [**-o** *mount_options*] [*fsname*] [*dir*]

where

p	Print tabular list of mounted file systems.
a	Mount all directories listed in fstab. If you also specify **-u**, unexport all currently exported files.
v	Run in verbose mode, printing the name of each directory as it is exported or unexported.
u	Unexport the indicated directories.
i	Ignore options found in the fstab file. Used to override fstab file options with options specified on the `mount` command line.

f Force the normally illegal export of a parent or child of a directory that is already exported.

o [*mount_options*]
Identical to fstab file *mount_options*. Can be used to override options in the fstab file, or applied to the *dir* specified on the command line. Mount options are listed in Table 3 page 180.

fsname Name of the remote file system. If fsname is of the form, *host:path*, the file system type is assumed to be `nfs`.

dir Local mount point through which the client accesses the remote file system specified as *fsname*.

Table 3 lists a few of the more commonly used mount options.

Table 3 Mount options

Option	Function
`rw`	Allow read/write access to file system. This is the default.
`ro`	Mount file system read-only.
`bg`	If the mount attempt fails, retry in the background.
`hard`	Retry operations until the server responds; operations cannot time out or be interrupted. If the server fails to respond, the following message is displayed: `NFS Server Not Responding.` File systems are hard mounted by default.
`soft`	Return a time-out error if operation fails to complete within specified `timeo` interval `retrans` number of times.
`retrans=`*n*	Requests *n* attempts be made on an operation on a soft-mounted file system before returning a time-out error.
`timeo=`*n*	Specifies the time-out interval, in tenths of a second.

For more information about mount options, refer to the mount(8) and fstab(5) man pages.

Step 3 To make sure you have successfully mounted a file system (and that you have mounted it where you expected to) use either `df` or `mount`, without arguments. Both display information about currently mounted file systems, but in different formats.

Removing temporary files created by NFS

To create a temporary file in a local environment, many applications create or open a file then immediately unlink (remove) the file from the directory in which it was created. The file handle returned by the kernel is how the application process continues with reads and writes, while the file-open connection state is maintained by the kernel. When the application closes the file, the operating system (kernel) detects that no more references are being made to this temporary file, and so it deletes the temporary file.

In the case of NFS, servers do not maintain connection state information. Hence, temporary files created by client calls to a server are renamed in the format, `.nfs`*xxxx*, where *xxxx* is a number related to the time the file was renamed. It is the NFS client's responsibility to detect when a temporary file actually closes, i.e., when the file's reference count goes to zero. With the reference count zero, the client calls the server to remove the temporary `.nfs`*xxxx* file. Figure 88 shows the format of .nfs file entries in a directory.

Figure 88 NFS temporary files

```
-rw-r--r-- 1 jones 1024 Apr 22 13:46 .nfs0DAA
-rw-r--r-- 1 jones 3072 May 11 12:52 .nfsDCAA
-rw-r--r-- 1 jones 1024 May 18 10:16 .nfsE637
-rw-r--r-- 1 jones 3072 Jun 11 12:12 .nfsECAA
-rw-r--r-- 1 jones   0 Jun 18 10:16 .nfsF637
```

NFS temporary files may never get deleted if an NFS client (NFS code in kernel on the client system) does not call the server to remove them. Conditions such as a workstation reboot or program error conditions can occur before a client has a chance to request the removal of its temporary files.

You can manually remove `.nfs`*xxxx* files, or you can set `cron` to remove them for you. For example, to have `cron` change the working directory to a specified project directory, and remove all the `.nfs`*xxxx* files at 7 a.m. every Sunday, create a .crontab entry containing

```
0 7* *7 cd ~/project_dir ".nfs????" -atime +7 rm {}\;
```

Tuning NFS

The following topics are discussed in this section:

- How to check privileged ports
- How to patch Sun-3 clients for compatibility with file systems with large block sizes

Checking privileged ports

A BSD-based operating system includes some Internet domain source ports to which only privileged users can attach. NFS does not check to see if a client is bound to one of these ports; that is, an NFS server has no way of knowing whether a client's file request originated from the real client's kernel or from an intruder's user program. You may want to enforce privileged port checking for extra security. To turn on NFS server port checking, use the following procedure:

Step 1 Add the following line to bootcmd on the SPU:

```
tune cpu nfs_portmon = 1
```

Step 2 Be sure that the /etc/inetd.conf file includes the line, shown in Figure 89, that ensures that the `-n` option is not set in the `rpc.mountd` daemon. The `-n` option causes `mountd` to not check whether clients are root users.

Figure 89 `rpc.mountd` without **`-n`** option in /etc/inetd.conf

```
mount      dgram    udp     wait root 1        /usr/etc/rpc.mount          mountd
```

Step 3 Restart `inetd` according to the instructions in Chapter 25, "Managing NFS and NIS daemons," page 315 (send `inetd` a `SIGHUP` [kill -1] signal). Unless you restart `inetd`, privileged port checking does not start until after the system is rebooted.

Exporting file systems with large block sizes

If you are using a Sun-3 workstation as a client to a CONVEX NFS server, note the following. Because of an bug in NFS releases prior to and including V3.0, the Sun-3 kernel panics when accessing remote file systems with block sizes larger than 8 Kbytes. This bug is relevant to you because CONVEX NFS fully supports file system block sizes from 4 Kbytes through 64 Kbytes.

You can fix this bug by patching the Sun-3 kernel. To do this, use `adb` to change nfs_mount+2b6. The previous instruction was `bles`, hex 6f06 (halfword). Replace it with a `bra` instruction, hex value 6006 (halfword).This change is to be made on the Sun-3 machine. The `adb` session to make the change would look like

```
sun# adb -w /vmunix
nfs_mount+2b6?x(check for 6f06)
nfs_mount+2b6?w 6006(replace 6f06 with 6006)
$q
sun#
```

Versions of NFS ported by other vendors may not be designed to support file systems with block sizes larger than 8 Kbytes. You may have difficulty using other types of hardware as a client for these file systems.

Using the lock manager 20

CONVEX NFS supports advisory record locking. This capability ensures System V compatibility and provides a mechanism for locking records in a network environment, making the use of distributed databases using NFS smoother and more useful. Some of the utilities used to support record locking include

- **fcntl**— Executes file and record locking requests.
- **lockf**— User-friendly front end to `fcntl`.
- **lockd**—Processes lock requests that are either sent locally by the kernel or remotely by another lock daemon.
- **statd**—Interacts with `lockd` to provide crash and recovery functions for the NFS locking services.

Subsequent sections explain these programs.

Handling local and remote lock requests

The `lockd` daemon processes and arbitrates lock requests from local processes and remote NFS client processes. When a user issues a `lockf` or `fcntl` call, the kernel contacts the local `rpc.lockd` process. This happens whether the file is local or remotely mounted via NFS.

If the `lockd` process has not been registered with `portmap`, `fcntl` exits with the ENOLCK diagnostic. If `portmap` has not been started, `fcntl` waits either for `portmap` to be started or until a signal is delivered to the process; for example, the user presses the interrupt key.

After the local `lockd` process has been contacted, the kernel sends the particular locking request to the local `lockd` process, and the local `lockd` processes the request as follows:

- `lockd` determines whether the request is for a file on the local host or for a partition remotely mounted via NFS.

- If the request is for a lock on a remote file, `lockd` contacts the local `statd` process to register for monitor server crash-recovery services.
- If the request is for a remote file, `lockd` contacts the remote server's `lockd` process to register the request and return the result of the operation.

The local `lockd` process contacts the local `statd` process for remote files. Then the local `lockd` process either contacts the remote `lockd` (if the request is for a remote file) or processes the request (if the request is for a local file). The results are returned to the kernel, and the kernel delivers the results to the user.

Installing the lock manager system

Follow the steps in this section, modifying rc.local according to whether you plan to use record locking only on local files or in the network environment with NFS. Installng the lock manager system is merely a matter of editting your /etc/rc.local file to start two daemons: `statd` and `lockd`.

Step 1 Find the lines in rc.local that look similar to those shown in Figure 90. The rc.local file in Figure 90 is the current version; you may have to edit your version to make it current.

Figure 90 Updating lines in rc.local

```
echo -n 'checking quotas: '
          /usr/etc/quotacheck -p -a
          /usr/etc/quotaon -a
echo  'done.'
#
if [-f  /usr/etc/nfsd -a -f /etc/exports -a -f /usr/etc/exportfs ]; then
        /usr/etc/nfsd 4 & echo -n ' nfsd'
        >/etc/xtab; /usr/etc/exportfs -a &
fi
```

Step 2 Add the following lines to /etc/rc.local, within the section `echo -n 'local daemons:'` that follows immediately below the lines listed in Step 1.

Figure 91 Updating the `statd` and `lockd` lines in rc.local

```
if [ -f   /usr/etc/rpc.statd -a -f /usr/etc/rpc.lockd ]; then
          /usr/etc/rpc.statd & echo -n ' statd'
          /usr/etc/rpc.lockd & echo -n ' lockd'
fi
```

The /usr/etc/rpc.statd program is a status monitor. It is informed of the recovery of NFS servers after they crash and communicates with /usr/etc/rpc.lockd to provide crash-and-recovery functions.

rpc.lockd processes record-locking requests sent either locally by the kernel or remotely by another lock daemon. Lock requests are forwarded to the server's lock daemon using standard RPC and XDR routines. The lock daemon then requests statd for monitor services.

Capturing state information for recovery

The statd process maintains state information about which hosts have outstanding record lock requests of this server. State information is contained in the /etc/sm directory in the form of one directory entry for each host that has requested monitor services. When statd starts, it notifies the statd process of each host listed in the /etc/sm directory that it has recovered. When the remote statd receives this notification, it notifies its local lockd of the recovery. The local lockd tries to reclaim the lock, if possible. If it cannot, the process receives a SIGLOST signal to note the lost lock.

Incompatibilities with non-NFS file systems

A few things work differently, or do not work at all, on remote NFS file systems. Here are some incompatibilities with work arounds.

Unsupported file operations

The `flock` call is not defined across the network. Neither does it recognize locks placed by `lockf` or `fcntl`. This means that server processes using `flock` may mistakenly believe they have exclusive access to a file that has already been locked using `lockf`.

To prevent this problem, use `lockf` exclusively and avoid `flock`.

Using NETdisk for booting diskless workstations

21

Standard shipments of CONVEX NFS include NETdisk, which allows a CONVEX server to support booting diskless workstations using NFS. This chapter presents an overview of NETdisk operation, and explains how to

- Determine disk space requirements
- Use `INSTALL` to load client software
- Boot a diskless workstation
- Add and remove clients
- Install optional software after running `INSTALL`

For further information about NETdisk, see the INSTALL(8), setup_client(8), and sunboot(8s) man pages.

Overview

As an introduction to NETdisk, this section describes

- What is NETdisk
- What happens when a client workstation is booted
- Important files used by NETdisk

What is NETdisk?

NETdisk is a collection of administrative programs loaded into the /usr/etc/install directory when NFS is installed. NETdisk is used to boot diskless workstations (called clients) from a CONVEX NFS server.

With NETdisk, you can use relatively inexpensive diskless workstations without purchasing expensive boot or file servers, because your CONVEX NFS server provides the needed files.

NETdisk supports are Sun-3 and Sun-4 workstations, and workstations whose release tapes (boot procedure) model Sun workstation release tapes. The supported operating system releases are SunOS 4.0, 4.0.3, 4.1, and 4.1.1 with a caveat.

SunOS 4.1.1 requires an uncompress tool for its tar archives during install. Compress tools are in the public domain and available through many sources. Running `INSTALL`(8) with `uncompress` in the search path facilitates a successful install. This is discussed further in "What happens when a client is booted," below.

Workstation software for diskless workstations must be ordered from the workstation's vendor and loaded onto your CONVEX system.

What happens when a client is booted

This section describes, in general terms, the mechanics of booting a diskless workstation. The general booting procedure is further discussed in the sunboot(8s) man page.

When booting, a client knows only its Ethernet hardware address. This address is used in a broadcast Reverse ARP request to discover the client's internet address (IP address). The NETdisk server responds to this request by providing the client's IP address, found in the server's `arp` cache, a kernel table of internet/Ethernet address mappings.

Once the client knows its IP address, it uses `tftp` to send a `get` request to retrieve a file named the same as its IP address in uppercase hex characters. Some clients also append a dot and uppercase architecture name, such as .SUN4, to the IP address in this request; the `in.tftpd` daemon handles this peculiarity.

This initial `tftp` request is first directed to the server that answers the Reverse ARP query. If no response is received from that server, the request is broadcast on the local network.

The file to be downloaded via `tftp` is the actual boot program. So that there is no need for information passed between booting phases, the boot program also sends a Reverse ARP request as above. Boot then sends a `bootparams whoami` request, directed first to the responding server, then broadcast, as with `tftp`. The result of this remote procedure call tells the client its host name, domain name, and the IP address of an acceptable IP router. A second `bootparam` procedure, `getfile`, then determines the NFS directory of the operating system image (i.e., the client's root path name). This path name is used in an NFS mount request to get the directory file handle, which is then used in normal NFS lookup and read procedures to load the kernel (vmunix by default).

Once the kernel is loaded and running, it uses reverse ARP and `bootparam` requests to get path names for root, swap, and dump. Default keys used in the `getfile` requests are root, swap, and dump. To override these keys, use the **`-a`** option with the Sun PROM monitor boot command as follows:

```
>b le() -a
```

You will then be prompted for the name of the root device.

For an example of this procedure, refer to "Booting a diskless Sun" page 213.

Important NETdisk files and directories

Files and directories used by NETdisk are described in Table 4.

Table 4 Key NETdisk files and directories

File or directory	Function
/etc/exports	Specifies export options used by the client root directory, swap files, and executables. It is automatically updated by the `INSTALL` script.
/etc/bootparams	Used by `rpc.bootparamd` to supply information about a client's host name and location of its root, swap, and dump files. Automatically updated by `INSTALL` and `setup_client` scripts. An NIS map is also produced for this file.
/usr/etc/install	Contains the `INSTALL` and `setup_client` scripts. The current working directory must be /usr/etc/install and "." must be in your PATH environment variable when either script executes.
/tftpboot	Downloads the boot program using the `tftp` protocol from a boot server to a diskless client.
/export/exec	Contains architecture-specific programs and libraries (essentially, the /usr file system) shared among similar clients. For example, the programs for a Sun-3 host are stored in the /export/exec/sun3 directory.
/export/root	Contains the root file system for each client served from this server. For example, the client, jogger, finds its root file system in the /export/root/jogger directory.
/export/swap	Contains the files that clients use for swapping. For example, the swap file for the client, jogger, is /export/swap/jogger.
/export/dump	Holds crashdump image of a crashed client. Normally, the image is written to the swap file.

Determining disk space requirements

The amount of disk space consumed per workstation depends on the following factors:

- The number of software packages loaded per architecture. Each loaded architecture may be loaded with selected software packages; the more packages loaded, the larger the disk space needed.
- The number of clients supported. Each client workstation requires its own root directory, plus /tmp and spool space.

- The size of a client's swap files. Each client workstation requires one or more files that it uses exclusively as a swap device. Disk space must be statically allocated for this swap file on the CONVEX server to be certain that a swap `write` request does not fail because a disk partition filled up. The size of the swap file varies depending on the amount of memory and type of application the client workstation is intended to run.

Use the following approximate disk space values for SunOS 4.0 to decide how much disk space to allocate for NETdisk:

- Architecture-shared executables (/usr):58 Mbytes
- Root directory per client workstation:2 Mbytes
- Swap file per client workstation:12 Mbytes (minimum)

To support one Sun-3 workstation with all standard optional software loaded requires about 72 Mbytes of disk space. Each additional client, however, requires only about 14 Mbytes. Supporting 10 diskless Sun-3 workstations would therefore require 198 Mbytes of disk storage (58 Mbytes for /usr, 140 Mbytes for the root directories and swap files of the 10 diskless clients).

These disk space values are estimates. Your disk space requirements may differ depending on the size of swap files used and the amount of optional software installed.

CONVEX recommends that you create a separate file system called /export to hold NETdisk files. Within /exports, set up these subdirectories

/export/root/	root directory for client
/export/swap/	swap file for client
/export/exec/	architecture (/usr) directory containing an entry for each architecture

Prerequisites to loading software

The /etc/hosts and /etc/ethers files must be updated prior to invoking the INSTALL program. If they are not, INSTALL fails to find the needed information and prompts for another client host name.

Before loading the software and booting a diskless workstation, complete the following steps:

Step 1 Add the host name and internet protocol (IP) address of each new client workstation (the one you are going to boot) to the /etc/hosts file. For information on assigning IP addresses, Refer to Chapter 10, "Setting up the host name database" page 69.

Step 2 Identify the Ethernet address of the client workstation by powering up the monitor.

Step 3 Log in to the NETdisk server.

Step 4 Add the Ethernet address of each client workstation to the /etc/ethers file.

Step 5 If your installation uses NIS, remake the NIS maps. Refer to Chapter 23, "Managing NIS" page 247 for information about setting up NIS maps.

Step 6 Take the client's Ethernet address stored in the /etc/ethers file and make a permanent address entry in the kernel by entering this arp command

```
/etc/arp -e /etc/ethers
```

Step 7 Check that /etc/rc.local file contains these lines automatically inserted by the NFS installation procedure.

```
if [ -d /tftpboot ] ; then
     if [ -f /etc/ethers -a -f /etc/arp ] ; then
     /etc/arp -e /etc/ethers & echo -n 'rarp'
     fi
fi
```

In subsequent boots of the diskless client workstation, these lines in /etc/rc.local file will automatically map the client's Ethernet address to a permanent address entry in the kernel.

Step 8 Set up the exports file system. Refer to Chapter 19, "Managing NFS" page 161, and to the newfs(8) man page for details on setting up a new file system.

Using INSTALL to load software

This section guides the superuser through an example installation of the client architecture software. This example installation assumes the following conditions to be true:

- You are about to install a Sun-3 architecture from a local 1/2 inch magnetic tape drive.
- A diskless Sun-3 workstation named "jogger" has been configured and set up.
- A CONVEX host is running in multiuser mode, and the NFS system is fully functional.
- The SunOS 4.0 software distribution tape for the Sun-3 has been loaded onto the local CONVEX host's tape unit 0.
- The CONVEX server is running NIS.
- The /export file system has been set up and is large enough to hold
  - /export/root/
  - /export/swap/
  - /export/exec/
- The hosts database lists the client as a known host.
- The /etc/ethers file includes the client's Ethernet address.
- The current directory is in the PATH environment variable.
- You have superuser (root) privileges.

Preparing to load a sample architecture from tape

This procedure takes approximately two hours to complete. The program does not have to be constantly monitored after you choose the software to be loaded for a given architecture; however, you may have to change tapes. An optional software package called Games will be installed at the end of this chapter, after base architectures and clients are defined and installed.

Step 1 Log in as superuser.

Step 2 Change your working directory to /usr/etc/install by entering

```
cd /usr/etc/install
```

This directory contains administrative scripts that extract software distributions from tapes, as well as add and remove clients.

Step 3 Invoke the INSTALL program by entering

```
INSTALL
```

Step 4 INSTALL prompts you for the type of tape installation. Enter **local**, as shown in Figure 92.

Figure 92 Supplying the installation type

```
Enter tape drive type ? [local | remote]: local
```

Step 5 Next, INSTALL wants to know the tape drive to use. You must specify a nonrewind device. For a local installation on 1/2 inch magnetic tape, enter **/dev/rmt12**.

When using a local magnetic tape, you must specify the full path name to a nonrewind tape drive. Do not enter the shorthand mt8 specification because the INSTALL program interprets it to be a Sun cartridge tape drive, which is unavailable on CONVEX hosts.

If you had entered remote in Step 4, INSTALL would have prompted for the host name to which the remote tape drive is connected. (Remote installations require that the remote host's /.rhosts file contain the local CONVEX host name on a line by itself.) When doing a remote installation using a 1/4 inch QIC-24 cartridge tape from an already configured Sun system, the correct response is st08 to select the Sun QIC-24 cartridge tape drive.

Step 6 Next enter the architecture type to load. Because this example assumes a Sun-3 architecture, the appropriate response is **sun3**, as shown in Figure 93.

Figure 93 Entering the type of architecture to load

```
Enter next architecture type to load
[ sun3 | sun4 | ... | continue | done ]: sun3
```

Step 7 The INSTALL program asks where to load the executables. Enter the directory named /export/exec that you created in "Determining disk space requirements" page 192, as shown in Figure 94.

Figure 94 Directing the load of the sun3 executables

```
Enter pathname for sun3 executables ? /export/exec
```

INSTALL automatically creates the Sun-3 directory in /export/exec so the entry becomes /export/exec/sun3 to hold these executables.

Step 8 Be sure the tape is mounted and online before you follow the instruction shown in Figure 95.

Figure 95 Mounting the Sun-3 release tape

```
Reading the table of contents for sun3
architecture...
Mount a sun3 release tape and hit RETURN, or
enter "exit" :
```

The INSTALL program now tries to access the tape. Press **RETURN** when ready.

Loading architecture from tape

At this point, the INSTALL program reads a standard format-header file on the tape that describes the entire contents of all the tapes comprising the release. The INSTALL program then prompts for confirmation to load each file; although, no files are loaded at this time. Rather, an extraction list is created for automatic extraction at a later time.

Figure 96 is a sample that shows how you would select both base and optional software for extraction later. The Games file will be loaded later in this chapter during an exercise of how to rerun INSTALL to add optional software.

Figure 96 Specifying each software file to be extracted from tape

```
Select optional software for the sun3 architecture:
Install "User File System" [20971520 bytes; required] ? [y/n]: y
Install "Sys" [2721792 bytes; desirable] ? [y/n]: y
Install "Networking tools and programs" [953344 bytes; desirable] ? [y/n]:y
Install "Debugging tools" [3383296 bytes; desirable] ? [y/n]: y
Install "SunView_Users Programs" [1453056 bytes; common] ? [y/n]: y
Install "SunView_Programmers Programs" [2064384 bytes; optional] ? [y/n]: y
Install "SunView_Demo Program source" [564224 bytes; optional] ? [y/n]: y
Install "Text Processing tools" [690176 bytes; optional] ? [y/n]: y
Install "Install tools" [1004544 bytes; optional] ? [y/n]: y
Install "User Level Diagnostics tools" [1491968 bytes; optional] ? [y/n]: y
Install "SunCore Libraries" [2991104 bytes; optional] ? [y/n]: y
Install "uucp programs" [270336 bytes; optional] ? [y/n]: y
Install "System V programs and libraries" [4481024 bytes; optional] ? [y/n]: y
Install "Manual Pages" [6072320 bytes; optional] ? [y/n]: y
Install "Demonstration Programs" [2790400 bytes; optional] ? [y/n]: y
Install "Games" [2468864 bytes; optional] ? [y/n]: n
Install "Versatec" [6117376 bytes; optional] ? [y/n
Install "SunOs Security Features" [146432 bytes; optional] ? [y/n]: y
```

Support has been removed for the sharing of nonexecutable files, which includes man pages and nroff macro sets.

This completes the first phase of the procedure—loading the Sun-3 architecture.

Step 9 At this point, you have several choices for your next step. You can define another architecture, the same as the type just entered or another type, or you can move on to define the clients for an architecture already defined. The display that you see is shown in Figure 97.

Figure 97 Electing what to do next

```
Enter next architecture type to load
[ sun3 | sun4 | ... | continue | done ]: done
```

To proceed to supply client information for an architecture already defined, enter

```
done
```

Specifying client information

After defining one or more architectures and selecting their respective software for loading, the next phase is to define clients for each architecture type.

This phase of the INSTALL procedure is where you set up the root and swap files for the clients to be installed.

Step 1 In Figure 11, the client name — **jogger**—is entered.

Figure 98 Specifying a client host name

```
Enter a sun3 client name ? [ name | done ]: jogger
Verifying ip address...   192.18.44.130 jogger
Verifying ethernet address...   8:0:20:0:f:ad jogger
```

INSTALL checks the /etc/hosts file (or the hosts database if NIS is running) to verify that the host you specified, in this case jogger, is a known host. INSTALL also checks the /etc/ethers file (or the NIS ethers map) to determine the Ethernet address of the host named jogger.

Step 2 INSTALL asks for the NIS type of the host name you just entered. This example procedure assumes that NIS is running on the CONVEX server, so the appropriate response is shown in Figure 99 as **client**.

Figure 99 Telling NIS the host type

```
Enter NIS type of jogger ? [ master | slave | client | none ]: client
```

Step 3 Next, INSTALL requests information about the client host you named, in this example—jogger. You are prompted for the swap size, root directory, swap and dump file, and home directory. CONVEX suggests using the values listed below as initial default values. The swap size of 12 Mbytes can be easily increased later if necessary.

Figure 100 Defining path names for client software on server

```
Enter swap size of jogger ? 12m
Enter root pathname of jogger ? /export/root
Enter swap pathname of jogger ? /export/swap
Enter dump pathname of jogger (or "none") ? none
Enter home pathname of jogger (or "none") ? none
```

The dump pathname and home pathname queries are answered **none** because this example installation does not use the /home partition and the dump pathname defaults to the swap partition /export/swap. The path names /export/dump and /export/home may also be used.

Step 4 Figure 101 asks you to confirm that the information you specified is correct; to confirm enter

y

Figure 101 Confirming client host information

```
#Information for jogger ok ? [y/n] : y
```

If you answer **n**, the program prompts you again to specify the file swap size, root pathname, swap pathname, and so on.

Step 5 At this point, if you want to add another client, respond to the prompt displayed in Figure 102 by entering a new client name. More clients may be added at this time by repeating steps 1 through 5. For this example only one client is added, so **done** is entered in Figure 102.

Figure 102 Completion of adding clients

```
# Enter a sun3 client name ? [ name | done ]: done
```

Installing architectures and setting up clients

Each architecture and its clients are installed as related units. First INSTALL loads architecture software from the distribution tapes into /export/exec/*arch*, then it sets up each client of that architecture.

Figure 103 displays the screen on which you elect to start the install. Enter

y

as shown in Figure 103.

Figure 103 Starting the install

```
Are you ready to start the installation ? [y/n] : y
Beginning Installation for the sun3 architecture.
Installation of sun3 executable files begins :
[Loading version 4.0 of sun3 architecture.]
Loading prototype root tree...
Extracting "usr" files from "/dev/rmt12" release tape.
Extracting "Sys" files from "/dev/rmt12" release tape.
Extracting "Networking" files from "/dev/rmt12" release tape.
Extracting "Debugging" files from "/dev/rmt12" release tape.
Extracting "SunView_Users" files from "/dev/rmt12" release tape.
```

Time to install all software is approximately two hours—about one hour per tape. After about an hour, a prompt, like the one shown in Figure 104, tells you to load the second release tape; in this example, this tape is for Sun-3.

Figure 104 Prompting you to load the second tape

```
Tape loaded is #1
Load release tape #2 for architecture sun3 and hit RETURN:
```

Load the second tape, and press **RETURN**.

Figure 105 shows the display as files are installed.

Figure 105 Output during installation

```
Extracting "SunView_Programmers" files from "/dev/rmt12" release tape.
Extracting "SunView_Demo" files from "/dev/rmt12" release tape.
Extracting "Text" files from "/dev/rmt12" release tape.
Extracting "Install" files from "/dev/rmt12" release tape.
Extracting "User_Diag" files from "/dev/rmt12" release tape.
Extracting "SunCore" files from "/dev/rmt12" release tape.
Extracting "uucp" files from "/dev/rmt12" release tape.
Extracting "System_V" files from "/dev/rmt12" release tape.
Extracting "Manual" files from "/dev/rmt12" release tape.
Extracting "Demo" files from "/dev/rmt12" release tape.
Extracting "Versatec" files from "/dev/rmt12" release tape.
Extracting "Security" files from "/dev/rmt12" release tape.
Installation of sun3 executable files completed.
```

Following the installation of the executables, INSTALL sets up the clients and updates the associated administrative files—the root file system for each client. Figure 106 shows the output as each event occurs.

Figure 106 Output during the set up of clients and update of associated files

```
Starting installation of sun3 clients...
Start creating sun3 client "jogger" :
Updating bootparams ...
Creating root for client "jogger".
Creating 12m bytes of swap for client "jogger".
Setting up /tftpboot directory.
Completed creating sun3 client "jogger".
Updating bootparams NIS map...
updated bootparams
pushed bootparams
Diskless Client Installation Completed.
```

The system prompt returns when installation completes.

Adding and removing NETdisk clients

This section describes using the `setup_client` program to add and remove NETdisk clients. The `INSTALL` program may be used interactively to load client architecture support onto a CONVEX server. Once the client architecture support has been loaded, however, it may be more convenient to use the `setup_client` program to add new clients or remove existing ones.

The setup_client program, as `INSTALL`, is found in the /usr/etc/install directory. For more information on the setup_client program, refer to the `setup_client`(8) man page. To display its options, enter

```
setup_client
```

When you enter this command without arguments, the response is a list of available optional arguments (options).

Unlike `INSTALL`, `setup_client` is not interactive. Instead, the steps of adding and removing a client take place on the command line without confirmation.

Removing a client

To remove the client "jogger, "defined and set up by `INSTALL` earlier in this chapter, follow the steps outlined in this section.

Step 1 Log in as superuser.

Step 2 Change your current working directory to /usr/etc/install by entering

```
cd /usr/etc/install
```

Step 3 Enter

```
setup_client
```

and read about the available arguments, as shown in Figure 107.

Figure 107 Listing the options for `setup_client`

```
# setup_client
setup_client: incorrect number of arguments.
usage:
setup_client op name NIS size rootpath swappath dumppath
homepath execpath sharepath arch
where:
op          = "add" or "remove"
name        = name of the client machine
NIS         = "master" or "slave" or "client" or "none"
size        = size for swap
(e.g. 16M or 16m ==> 16 * 1048576 bytes
16K or 16k ==> 16 * 1024 bytes
16B or 16b ==> 16 * 512 bytes
16         ==> 16 bytes )
rootpath  = pathname of root (e.g. /export/root )
swappath   = pathname of swap (e.g. /export/swap )
dumppath    = pathname of dump (e.g. /export/dump ) or "none"
homepath   = pathname of home (e.g. /home or homeserver:/home)
or "none"
execpath   = full pathname of exec directory (e.g. /export/exec/sun3 )
sharepath   = full pathname of shared files (e.g. /export/exec/share or
"none"
arch        = "sun3" or "sun4" etc
```

To execute `setup_client`, include the appropriate arguments in the order specified in Figure 107.

Step 4 To remove the client jogger, enter the `setup_client` specification defined in Figure 108.

Figure 108 Command specification that removes a client

```
# setup_client remove jogger client 10m /export/root\
 /export/swap none none /export/exec/sun3 none sun3
```

As jogger is removed, output displays the progress of the removal. Figure 109 shows the removal of jogger.

Figure 109 Confirmation statements as a client is removed

```
Start removing sun3 client "jogger" :
Updating bootparams ...
updated bootparams
pushed bootparams
Removing root for client "jogger".
Removing swap for client "jogger".
Completed removing sun3 client "jogger".
```

Adding a client

To add the client, jogger, enter the `setup_client` specification defined in Figure 110.

Figure 110 Command specification to add a client

```
# setup_client add jogger client 10m /export/root\
 /export/swap none none /export/exec/sun3 none sun3
```

Figure 111 shows the output displayed as the client is added.

Figure 111 Confirmation statements as a client is added

```
Start creating sun3 client "jogger" :
Updating bootparams ... updated bootparams pushed bootparams
Creating root for client "jogger".
Creating 10m bytes of swap for client "jogger".
Setting up /tftpboot directory.
Updating /etc/exports to export "jogger" info.
Completed creating sun3 client "jogger".
```

Making mount points

Mount points are locations within a directory tree through which a computer accesses directories. These directories can be mounted locally from a disk or tape or remotely from another computer on the network. For example, when you use the `restore` command, you mount the files on the backup tape through /mnt, an already existing mount point in the root file system. Furthermore, when the installation script (`INSTALL`) creates a client's root, swap, and home directories in a server's /export file system, it also creates the mount points /, /swap, and /home for them in the client's directory tree.

Installation script-created mount points

When a server is first set up, the installation script creates default fstab files for each of the server's clients.The contents of /etc/fstab remain the same until you change them. A default client /etc/fstab file resembles these example lines, set up by server "dancer" for client "raks".

Entries in an /etc/fstab contain the fields shown in Figure 112.

Figure 112 Sample /etc/fstab entries

```
dancer:/export/root/raks / nfs rw 0 0
dancer:/export/swap/raks / nfs rw 0 0
dancer:/usr /usr nfs ro 0 0
dancer:/home/dancer /home/dancer nfs rw 0 0
```

User-created mount points

If the client needs to mount additional directories, then a mount point must be created for each additional directory. You need to create mount points for any nondefault directories mounted through the /etc/fstab file or through the `mount` command. Mount points are essentially empty directories that you create through the `mkdir` command. The syntax is

`mkdir` *`mount_point`*

where *mount_point* is the full path name of the directory you want created. To add a client, you specify the data shown in Figure 110.

Figure 113 Command specification to add a client

```
server:server_dir  client_mountpoint   type   options   freq   pass
```

`server`	Name of the server exporting the directory the client wants to mount. In the sample file, dancer is the only server listed.
`server_dir`	Name of the directory to be mounted from the indicated server. In the first entry, the directory is /export/root/raks, client raks' individual root directory. Note that in the default fstab file for the client, the only mounted directories are those that the client requires to operate its individual root, swap, home, and /usr, which, along with the client root, contains all essential programs.
`client_ mountpoint`	Mount point on the client through which it accesses directories mounted from the server. In the first entry in the example /etc/fstab file above, the mount point is the client's root directory.
`type`	Type of mount taking place. In the example above, this is an nfs mount. If the client has its own disk, you can also set up its /etc/fstab file so that it can perform BSD 4.3 mounts of file systems on its local disk.
`options`	Any of a number of options provided for nfs or BSD 4.3 mounts. Refer to the fstab(5) man page for a complete list. For example, if you are running secure, you may want to select the secure option for nfs mounts. The example file shows client "raks" performing nfs mounts with the common options `-rw` or `-ro`. This client can mount its own root swap and home directories with read-write access. However, it accesses the server's /usr directory with read-only permission; thus a user on the client cannot add or modify a file in /usr.

`freq`	Interval in days between dumps of the directory.
`pass`	fsck pass in which the listed file system is checked. If a zero is indicated, the file system is not checked during `fsck`.

Rerunning INSTALL to load optional software

At the beginning of this chapter base and optional software were specified in the extraction list then loaded in FIGURE xx—all except the optional package "Games." This section contains the instructions for how to use the INSTALL script to extract optional software *after* the basic architectural support has been loaded, in this case, "Games." More information is available on the INSTALL(8) man page.

All the assumptions stated in the "Prerequisites to loading software" page 194 apply to this installation session.

Step 1 Log in as superuser.

Step 2 Change your working directory to /usr/etc/install, by entering

```
cd /usr/etc/install
```

Step 3 Invoke the INSTALL script by entering

```
INSTALL
```

Step 4 INSTALL prompts for the type of installation. enter

```
local
```

Step 5 INSTALL then asks which tape to use. For a local installation on 1/2 inch magnetic tape, enter

```
/dev/rmt12
```

because it is a nonrewind tape device.

Step 6 Next, enter the architecture type to load. This exercise assumes a Sun-3 architecture; therefore enter

```
sun3
```

Step 7 INSTALL asks where to load the executables. Supply the same path name entered for the load of the basic architecture and software, which was

```
/export/exec
```

Step 8 INSTALL finds that the /export/exec directory already exists, and prompts you to confirm the next step. The objective now is to install optional software; therefore, enter

use

as the response to the program prompt, requested in the example display shown in Figure 114.

Figure 114 Electing to add optional software to an existing directory

```
INSTALL: exec tree /export/exec/sun3 appears to already exist.
You may select one of the following options:
ignore  continue as if this architecture had not been specified
remove  remove existing exec tree, then continue
use     continue, loading any new optional software specified
upgrade continue, loading all new files unconditionally
clients continue with sun3 clients, but use existing exec tree? use
```

Step 9 The INSTALL procedure tries to access the tape.

With the tape mounted and online, the message in Figure 114 displays.

Figure 115 Message from INSTALL requesting a tape mount

```
Reading the table of contents for sun3 architecture...
Mount a sun3 release tape and hit RETURN, or enter "exit" :
```

To proceed, respond by pressing **RETURN**.

Step 10 Figure 116 shows the responses that extract only the **Games** file.

To extract another file, apply this pattern of answers as appropriate.

Figure 116 Extracting one file only

```
Select optional software for the sun3 architecture:
Install "User File System" [20971520 bytes; required] ? [y/n]: n
Install "Sys" [2721792 bytes; desirable] ? [y/n]: n
Install "Networking tools and programs" [953344 bytes; desirable] ? [y/n]: n
Install "Debugging tools" [3383296 bytes; desirable] ? [y/n]: n
Install "SunView_Users Programs" [1453056 bytes; common] ? [y/n]: n
Install "SunView_Programmers Programs" [2064384 bytes; optional] ? [y/n]: n
Install "SunView_Demo Program source" [564224 bytes; optional] ? [y/n]: n
Install "Text Processing tools" [690176 bytes; optional] ? [y/n]: n
Install "Install tools" [1004544 bytes; optional] ? [y/n]: n
Install "User Level Diagnostics tools" [1491968 bytes; optional] ? [y/n]: n
Install "SunCore Libraries" [2991104 bytes; optional] ? [y/n]: n
Install "uucp programs" [270336 bytes; optional] ? [y/n]: n
Install "System V programs and libraries" [4481024 bytes; optional] ?
[y/n]: n
Install "Manual Pages" [6072320 bytes; optional] ? [y/n]: n
Install "Demonstration Programs" [2790400 bytes; optional]? [y/n]: n
Install "Games" [2468864 bytes; optional] ? [y/n]: y
Install "Versatec" [6117376 bytes; optional] ? [y/n]: n
Install "SunOs Security Features" [146432 bytes; optional] ? [y/n]: n
Load sharable objects in shared location ? [y/n] : n
```

Step 11 This procedure completes the loading of the **Games** file. Respond with **done**, as in Figure 114, to the next two prompts—stating that the no other architecture support is to be loaded and that no clients are to be loaded.

Figure 117 Completion of architecture and client loads

```
Enter next architecture type to load
[ sun3 | sun4 | ... | continue | done ]: done
Enter a sun3 client name ? [ name | done ]: done
```

Step 12 The INSTALL program asks you if you are ready to start the installation. If you are ready, enter

y

Step 13 Figure 118 shows INSTALL asking for confirmation once again before proceeding. In this case, choose **ignore** to load the optional software directly onto the existing directory tree.

Figure 118 Giving INSTALL load instructions

```
Beginning Installation for the sun3 architecture.
/usr/etc/install/setup_exec: /export/exec/sun3 already exists.
You may select one of the following options:
abort   exit with error status
ok      exit with ok status
remove  remove existing tree, then continue
ignore  continue (load on top of existing tree)? ignore
```

Step 14 INSTALL realizes in Figure 119 that the **Games** file is not on the first tape and prompts for the second tape to be loaded.

Figure 119 Loading second tape for INSTALL

```
Installation of sun3 executable files begins :
[Loading version 4.0 of sun3 architecture.]
Tape loaded is #1
Load release tape #2 for architecture sun3 and hit RETURN:
```

Load the second tape, then press **RETURN** for the install to continue. Figure 120 contains the confirmation of the requested extraction.

Figure 120 Confirmation of load from INSTALL

```
Extracting "Games" files from "/dev/rmt12" release tape.
Installation of sun3 executable files completed.
```

Because no new clients are to be loaded, the procedure to install **Games** completes with a confirmation message and the system prompt shown in Figure 121.

Figure 121 Confirmation of install completion

```
Starting installation of sun3 clients...
Diskless Client Installation Completed.
#
```

Booting a diskless Sun

This section describes the steps required to boot the Sun workstation (client host) `jogger` installed in the previous section. Overall, booting is simple and can be done by either cycling the power to the Sun workstation or by simply typing **b** at the Sun PROM monitor.

This sample boot procedure that begins at Step 1 assumes that the following tasks have been completed:

- The Sun-3 has been connected to the Ethernet and is functional.
- The Sun-3 is turned on and the Sun PROM monitor prompt (>) is displayed.
- The Sun-3 architectural support programs and root directory for the client host (jogger in this example) have been loaded as described in the previous section.
- The Sun-3 is on the same subnet as the CONVEX server where the Sun-3 architectural support programs are loaded.
- The name of the CONVEX server is "conserve" and it is in the NIS domain "convex."
- NFS on the CONVEX host is functional, and the rpc.bootparamd server is running or installed in the /etc/inetd.conf file.

Before booting

When initially powered on, all a diskless workstation knows about itself is its 48-bit Ethernet address. In order to boot, the IP address is needed, plus its host name, boot server name, and location of its root and swap files. Diskless clients use Reverse ARP to fine their IP addresses from Ethernet addresses. Servers run an rarpd daemon to accept and process RARP requests broadcast by diskless clients attempting to boot. RARP requests are received by all boot servers on a network because these requests are broadcast.

Boot servers use the ethers and hosts maps to match an Ethernet address to an IP address. Many servers may reply, but one shows the symbolic link from the client's IP address to the boot block for that client's architecture. The server's tftpboot directory contains boot blocks for each supported client architecture.

Instructing the client to boot

You give a diskless Sun the command to boot. Booting is automatic and takes less than five minutes. These two basic steps are all that you need to tend to.

Step 1 Have the Sun-3 powered on with the Sun PROM monitor prompt—the greater than symbol—on the screen

```
>
```

Step 2 Enter the boot command immediately following the Sun PROM monitor prompt

```
> b
```

From this diskless workstation, a client tftp request is received by the server's inetd daemon, which in turn starts an in.tftpd daemon that follows the symbolic link from the client's IP address to the related boot file.

In Figure 122, the tftp server address 192.18.44.122 is the IP address of the CONVEX NETdisk server, "conserve," responding to the tftp request to download the boot program. The address of 192.18.44.130 is the booting Sun's IP address determined from "conserve" answering the client's Reserve ARP broadcast.

Figure 122 Downloading the boot program

```
Using IP Address 192.18.44.130 = C0122C82
Boot: le(0,0,0)
Booting from tftp server at 192.18.44.122 = C0122C7A
Downloaded 126056 bytes from tftp server.
```

For example, the boot program for a Sun-3 host is named/tftpboot/boot.sun3. The diskless clients download the file that is their internet address converted into uppercase hex numbers. This file is typically a symbolic link to the boot.sun? file (where sun? is the name of the sun client architecture). In this case, the client named jogger has an internet address of 192.18.44.130. This number converted to hex is C0122C82 (i.e., 192=C0, 18=12, 44=2C, and 130=82). The client downloads the file /tftpboot/C0122C82, which is a symbolic link to tftpboot/boot.sun3, then executes it.

After the tftpd daemon downloads the boot block from the server's tftp directory to the client, the client moves from the PROM code and executes the boot code. Again the client (actually the boot script) determines the correct IP address to use, location of its root file system, its hostname, and boot server name; it then connects to the rpc.bootparamd server on conserve and uses the NFS protocols to download the vmunix kernel. The names and address returned by rpc.bootparamd must match those in the client's configuration files and those in the NIS maps. Figure 123 shows the output as downloading takes place.

Figure 123 Downloading the vmunix kernel

```
Using IP Address 192.18.44.130 = C0122C82
hostname: jogger
domainname: convex
server name 'conserve'
root pathname '/export/root/jogger'
root on conserve:/export/root/jogger fstype NFS
Boot: vmunix
Size: 632768+115120+16076 bytes
```

Now the vmunix (kernel) issues a boot parameter request for the location of the client's swap file system. The client Sun-3 configures root and swap devices, then begins single user startup, and sets its

- IP address
- hostname
- NIS domain

During singleuser boot, the client mounts its /usr file system from the server listed in /etc/fstab. Finally, the client acts as if it were a system booting from a local disk.

Figure 124 displays the output lines as the Sun-3 kernel executes and the system comes up in multiuser mode. Note that portmap must be started before any other RPC (Remote Procedure Call) daemons, including inetd. It registers and maps RPC program numbers to their current (TCP or UDP) prot numbers. If portmap dies, all RPC daemons must be killed and restarted.

Figure 124 Output as vmunix executes

```
SunOS Release 4.0 (GENERIC) #5: Thu Apr 14 19:24:56 PDT 1993
Copyright (c) 1988 by Sun Microsystems, Inc.
mem = 4096K (0x400000)
avail mem = 2949120
Ethernet address = 8:0:20:0:f:ad
si0 at obio 0x140000 pri 2
sd0 at si0 slave 0
sd1 at si0 slave 1
sd2 at si0 slave 8
sd3 at si0 slave 9
st0 at si0 slave 32
st1 at si0 slave 40
zs0 at obio 0x20000 pri 3
zs1 at obio 0x0 pri 3
le0 at obio 0x120000 pri 3
bwtwo0 at obmem 0x100000 pri 4
bwtwo0: resolution 1152 x 900
hostname: jogger
domainname: convex
root on conserve:/export/root/jogger fstype NFS
swap on conserve:/export/swap/jogger fstype NFS size 10240K
dump on conserve:/export/swap/jogger fstype NFS
checking filesystems
Automatic reboot in progress...
Thu Sep  1 07:40:26 PDT 1993
checking quotas: done
starting rpc and net services: portmap ypbind keyserv routed.
mount: conserve:/export/exec/sun3 already mounted
rdate conserve
starting additional services: biod
starting system logger
starting local daemons: auditd sendmail statd lockd.
link-editor directory cache
preserving editor files
clearing /tmp
standard daemons: update cron uucp.
starting network daemons: inetd printer.
Thu Sep  1 07:41:22 PDT 1993
```

When booting completes, the system displays the login prompt for jogger.

```
jogger login:
```

Using the automounter utility

22

The automounter utility enables users to mount and unmount remote directories on an as-needed basis. When a user on a client machine running the automounter invokes a command that accesses a remote file or directory, such as opening a file with an editor, the automounter mounts the hierarchy to which that file or directory belongs. The automounter leaves the file or directory mounted as long as it is needed, but automatically unmounts it if it is not accessed for a certain amount of time. No mounting occurs at boot-time, and users need not know the superuser password to mount a directory. It is all done automatically and transparently.

Mounting some file hierarchies with `automount` does not exclude the possibility of mounting others with mount. For example, a diskless machine must mount /export/root, /export/swap, /usr and /export/exec/kvm through the `mount` command and /etc/fstab file.

This chapter discusses the following topics:

- Invoking the automounter
- Creating automounter maps
- Using NIS to maintain automounter maps
- Solving problems

Automounting file systems

Unless suppressed, the automounter's start-up definition in rc.local directs the automounter daemon to read the master map from the NIS auto.master map, as directed by shell script lines shown if Figure 125.

Figure 125 Default rc.local automounter entry

```
if [ -f /usr/etc/automount -a -f /etc/auto.master ]; then
     /usr/etc/automount -f /etc/auto.master & echo -n '
     'automount'
fi
```

To suppress automount from reading the NIS auto.master map and instead direct it to read an automounter auto.master file on a local host, the script line shown in Figure 126 would replace the above script lines in rc.local:

Figure 126 Alternate rc.local automounter entry

```
automount -m -f /etc/auto.master
```

The -m option instructs automounter to not consult the master file distributed by NIS. If you do not run NIS, you do not have to specify the -m option. Automounter remains silent when it does not find a distributed master file.

Without NIS, a system manager must distribute copies of automounter maps and keep them consistent on every host. This is not much of an improvement over hard-coding /etc/fstab files. A network benefits by using NIS and automounter together.

A master map lists the mount points of automounter's direct, indirect maps and usually a built-in map. The behavior of the `automounter` is based on the type of map in which a remote file system's mount-point information resides— in a direct map, an indirect map, or a built-in map. The difference between a direct and indirect map is the key; for a direct map, the key is a full pathname; for an indirect map, it is a simple name. Built-in maps, explained further on, use existing files to give the appearance of being an extension of the local file system.

Referencing direct maps

When a mount-point of the remote file system name—*server:pathname*— is found in a direct map, automounter mounts the remote file system under the directory /tmp_mnt. It answers the client kernel by telling it this is a symbolic link. Next, the client kernel sends an NFS `readlink` request, and the `automounter` returns a symbolic link to the real mount point under /tmp_mnt.

Referencing an indirect map

Indirect maps are relative to some mount point, which is given either at the command line or in a master map. When a mount-point of the remote file system is in an indirect map, `automounter` describes itself as a directory; it emulates a directory of symbolic links.

In response to a `readlink`, it returns a path to the mount point in /tmp_mnt, and a `readdir` of the automounter's mount point returns a list of the entries that are currently mounted.

Whether the map is direct or indirect, if the file system is already mounted and the symbolic link has been read recently, the cached symbolic link is returned immediately. Because the automounter is on the same host, the response is much faster than a `READLINK` to a remote NFS server. On the other hand, if the file system is not mounted, a short delay occurs while the mounting takes place.

Referencing a built-in map

The `-hosts` map is a built-in map This map uses the hosts database, which consists of the /etc/hosts file and the hosts.byname NIS map, for a list of every hostname on the network. How this map works is built into the automounter. Its entry in the auto.master map would be

```
/net -hosts
```

which tells automounter when a user changes to a named host, to mount all accessible exported file systems of that host.

For example, this command line

```
cd /net/belmont
```

causes automounter to mount at /net all of system belmont's exportable file systems.

Using NIS to maintain automounter maps

If maps are maintained on each client machine, then the benefits of using automounter are really lost. All three maps—direct, indirect, and built-in—can be distributed by NIS.

To add an automounter map to the NIS database, edit the NIS master server's Makefile so these script lines are read.

Preparing automounter maps

A server neither knows nor cares whether its shared files are accessed through `mount` or `automount`. For servers, nothing needs to be done differently for `automount` than what is done for `mount`; however, `mount` is restricted in that it cannot handle replicated file systems.

A client, however, does need special files for the automounter. As mentioned previously, `automount` does not consult /etc/fstab. Rather, when it starts up, it is passed the path name of a master map that lists where to find the file systems that can be accessed from the master map's mount point. All automounter maps are located in /etc, identified by the prefix, auto.

Locating maps through the master map

A master map, usually found in /etc/auto.master, tells automounter where to find the other maps. These other maps are either direct or indirect maps, and their location definition within auto.master looks like this:

```
# mount-point map              mount-options
/net          -hosts
/-            /etc/auto.direct-soft, rw, intr, timeo=20
/home         /etc/auto.home  -soft, ro, intr, timeo=20
```

Each definition contains three fields to describe the mounting of each map. These fields are

mount-point — Full pathname of a directory.

map — Name of the map that `automount` should use to find the mount points and locations.

mount-options — An optional, comma-separated list of options that regulate the mounting of the entries contained in the map, unless the map entries list other options. Options listed in the map entries override options specified in the mount-options list. Refer to Table 3, "Mount options," page 180 for an description of these options.

A line whose first character is # is treated as a comment. Note that /etc/auto.home is the name of an indirect map, /etc/auto.direct is the name of a direct map, and `-hosts` is a built-in map.

Referencing a built-in map: mount point /net

In this example, the automounter mounts under the mount point, /net, all the entries under the special map `-hosts`. This is a built-in map that does not use any external files except the hosts database, /etc/hosts. When a user changes to

`/net/` *hostname*

automounter will mount the exportable file systems for the host named in *hostname*.

Referencing an indirect map: mount point /home

The mount point /home is the directory under which the entries listed in /etc/auto.home (an indirect map) are to be mounted. Those entries are mounted under /tmp_mnt/home and a symbolic link is provided between /home/directory and /tmp_mnt/home/directory.

An entry in an indirect map looks like

```
parsley     -ro,intr          veggies:/usr/greens
```

Referencing a direct map: mount point /-

The mount point /- is a filler that tells the automounter to reference /etc/auto.direct, but not to associate the entries in /etc/auto.direct with any directory. Automounter then uses the mount points listed in the map; for a direct map, the key is a full path name.

An entry in a direct map reads as

```
/usr/man    -ro, intr belmont:/usr/man
```

Looking inside a direct map

Direct maps contain direct mount points for file systems with nonuniform naming schemes. The key for a direct map is a full path name of some directory. A direct map entry follows this syntax:

```
key        mount-option[...]          location
```

where each element provides the following information:

`key`	Path name of the mount point.
`mount-option`	One or more options that you want to apply to this particular mount.
`location`	Server:path name of the resource.

Writing an entry

Of all the maps, direct map entries most closely resemble their corresponding entries in /etc/fstab. Compare these two entries, and note their similarities.

- /etc/fstab entry

```
dancer:/usr/local /usr/local/tmp nfs ro 0 0
```

- direct map entry

```
/usr/local/tmp -ro          dancer:/usr/local
```

Figure 127 shows the setup of a direct map.

Figure 127 Example auto.direct map

```
/usr/local\
                /bin            -ro,soft        ivy:/export/local/sun3  \
                /share          -ro,soft        ivy:/export/local/share  \
                /src            -ro,soft        ivy:/export/local/src  \
/usr/man                        '-ro,soft       oak:/usr/man  \
                                                rose:/usr/man  \
                                                willow:/usr/man
/usr/games                      -ro,soft        peach:/usr/games
/usr/spool/news                 -ro,soft        pine:/usr/spool/news
/usr/frame                      -ro,soft        redwood:/usr/frame1.3  \
                                                balsa:/export/frame
```

Describing multiple mounts in a map entry

A map entry can describe any number of mounts, where the mounts can be from different locations and with different mount options. Consider the first entry in Figure 127. It is, in fact, one long entry whose readability has been improved by splitting it into three lines by using the backslash and indenting the continuation lines. Entry mounts /usr/local/bin, /usr/local/share, and /usr/local/src from the server ivy with the options read-only and soft. Figure 128 shows the full path name of these mount points.

Figure 128 Entry points for server ivy

```
/usr/local/bin              -ro,soft        ivy:/export/local/sun3  \
/usr/local/share            -ro,secure      willow:/usr/local/share \
/usr/local/src              -ro,intr        oak:/home/jones/src \
```

Figure 129 shows another version of how this entry could read, but the options are different and more than one server is used.

Figure 129 Multiple mounts Entry points for server ivy

```
/usr/local\
                /bin            -ro,soft        ivy:/export/local/sun3  \
                /share          -ro,secure      willow:/usr/local/share \
                /src            -ro,intr        oak:/home/jones/src \
```

Multiple mounts can be hierarchical. When file systems are mounted hierarchically, each file system is mounted on a subdirectory within another file system. When the root of the hierarchy is referenced, the automounter mounts the whole hierarchy. The concept of root here is very important. The symbolic link returned by the automounter to the kernel request is a path to the mount root. This is the root of the hierarchy that is mounted under /tmp_mnt. This mount point should theoretically be specified:

```
parsley         /       -ro,intr veg:/usr/greens
```

In practice, it is not specified because in a case of a single mount as above, it is assumed that the location of the mount point is at the mount root or '.'. Therefore, instead of the above, it is preferable, to enter

```
parsley                 -ro,intr veg:/usr/greens
```

However, the mount point specification becomes important when mounting a hierarchy: here the automounter must have a mount point for each mount within the hierarchy. The example above is a good illustration of multiple, nonhierarchical mounts under /usr/local when the latter is already mounted. Figure 130 shows a true hierarchical mounting.

Figure 130 Hierarchical mounting

```
/usr/local      /               -rw intr        peach:/export/local/ \
                /bin            -ro,soft        ivy:/export/local/sun3 \
                /share          -rw,secure      willow:/usr/local/share \
                /src            -ro,intr        oak:/home/jones/src
```

A true hierarchical mount can be problematic if the server for the root of the hierarchy goes down. Any attempt to unmount the lower branches will fail, because the unmounting has to proceed through the mount root, which also cannot be unmounted while its server is down.

The mount points used here for the hierarchy are /, /bin, /share, and src. Note that these mount-point paths are relative to the mount root, not the host's file system root. The first entry in the example above has / as its mount point. It is mounted at the mount root. There is no requirement that the first mount of a hierarchy be at the mount root. The automounter issues `mkdir` to build a path to the first mount point if it is not at the mount root.

Describing multiple locations

In the example for a direct map, displayed in Figure 131,

the mount points /usr/man and /usr/frame list more than one location (three for the first, two for the second). This means that the mounting can be done from any of the replicated locations. This procedure makes sense only when you are mounting a hierarchy read-only, since you have some control over the locations of files you write or modify.

Figure 131 /Direct map: etc/auto.direct map

```
/usr/local      /bin          -ro,soft      ivy:/export/local/sun3   \
/usr/local      /share        -ro,soft      ivy:/export/local/share  \
/usr/local      /src          -ro,soft      ivy:/export/local/src    \
/usr/man                      -ro,soft      oak:/usr/man   \
                                            rose:/usr/man  \
                                            willow:/usr/man
/usr/games                    -ro,soft      peach:/usr/games
/usr/spool/news               -ro,soft      pine:/usr/spool/news
/usr/frame                    -ro,soft      redwood:/usr/frame1.3   \
                                            balsa:/export/frame
```

A good example is the man pages. In a large network, more than one server may export the current set of manual pages. It does not matter which server you mount them from, so long as the server is up and running and sharing its file systems. In the example above, multiple mount locations are expressed as a list of mount locations in the map entry:

```
/usr/man  -ro,soft  oak:/usr/man  rose:/usr/man\ willow:/usr/man
```

This could also be expressed as a list of servers, separated by commas and followed by a colon and the pathname (as long as the pathname is the same for all the replicated servers):

```
/usr/man  -ro,soft  oak,rose,willow:/usr/man
```

Here you can mount the man pages from the servers oak, rose, or willow. From this list the automounter selects servers on the local network and pings these servers. This launches a series of RPC requests to the null procedure of the mount service in each server. (Note that the list does not imply any ordering.) The first server to respond is selected, and an attempt is made to mount from it.

This redundancy, which is useful in an environment where individual servers may or may not be sharing their file systems, exists only at mount time. There is no status checking of the mounted-from server by the automounter once the mount occurs. If the server goes down while the mount is in effect, the file system becomes unavailable. An option here is to wait five minutes until the automount takes place and try again. The next time, the automounter will choose one of the available servers. Another option is to use the umount command, inform the automounter of the change in the mount table and retry the mount.

Looking inside an indirect map

Indirect maps show their association to regularly named file systems such as home, tools, and system utilities. Inconsistent naming within a directory can be written to appear consistent on a client machine. Additionally, this map allows you to edit the mount point for a single indirect map, rather than all the files within that map, you reorganize your mount tree.

The syntax for an indirect map (like that for a direct map) is

```
key        [mount-option...]        location
```

where key is the name of the directory to be used as the mount point and `location` is the server:path name pair that identifies the source of the file system. When the automounter obtains the key, it appends the key to its associated mount point. If the mount point is not provided on the command line, the automounter takes the mount point from the master map that invoked the indirect map.

Writing an entry

An entry in an indirect map reads as

```
parsley     -ro, intr       veggies:/usr/greens
```

The key here needs more information: where is the mount point parsley really located? That is why you must provide that information either at the command line or through another map. The solution here is: let us say that this entry above is part of a map called /etc/auto.veggies. The command line entry would be:

```
automount /veggies /etc/auto.veggies
```

or listed in in the master map as:

```
/veggies    /etc/auto.veggies -ro,soft,nosuid
```

In either case, you are associating a mount directory veggies with the entries (parsley in this case) mentioned in the indirect map /etc/auto.veggies. The end result is that the file system /usr/greens from the machine veggies is mounted on /veggies/parsley when needed.

Another sample entry in the master map reads as

```
/home          /etc/auto.home     rw,intr,secure
```

Here /etc/auto.home is the name of the indirect map that contains the entries to be mounted under /home.

Figure 132 shows what auto.home map might contain.

Figure 132 etc/auto.home indirect map

```
key              [mount-option...]      location

willow                                  willow:/home/willow
cypress                                 cypress:/home/cypress
poplar                                  poplar:/home/poplar
pine                                    pine:/export/pine
apple                                   apple:/export/home
ivy                                     ivy:/home/ivy
peach            -rw,nosuid             peach:/export/home
```

Assume that the map in Figure 132 is on host oak. If user laura has an entry in the password database specifying her home directory as /home/willow/laura, then, whenever she logs into machine oak, the automounter mounts (as /tmp_mnt/home/willow) the directory /home/willow residing in machine willow. If one of the subdirectories is indeed laura, user laura will be in her home directory, which is mounted read/write, interruptible, and secure.

Suppose, however, that user laura's home directory is specified as /home/peach/laura. When she logs into oak, the automounter mounts the directory /export/home from peach under /tmp_mnt/home/peach. Her home directory will be mounted read/write, nosuid. Any option in the file entry overrides all options in the master map or the command line.

Now, assume the following conditions occur:

- User laura's home directory is listed in the password database as
  /home/willow/laura
- Machine willow shares its home hierarchy with the machines mentioned in
  auto.home
- All those machines have a copy of the same auto.home and the same password database.

Under these conditions, user laura can run `login` or `rlogin` on any of these machines and have her home directory mounted in place for her.

Furthermore, now laura can also enter the command

```
cd ~brent
```

and the automounter will mount brent's home directory for her (if permissions allow it).

On a network without YP, you must change all relevant databases (such as /etc/passwd) on all systems on the network in order to accomplish this. On a network running YP, propagate all the relevant databases throughout the network to ensure this.

Adding a subdirectory field within an indirect map

In the indirect file auto.home shown in Figure 133, access to a home directory in /home/willow requires that all the directories under it be mounted. Frequently used mount points such as home can become large and have multiple references to the same parent directory.

Because automounter treats each indirect map entry as a separate entity, it has to request multiple mounts from the same parent directory—one mount request for each of its subdirectories listed in the indirect map. Using a subdirectory field specifies only to mount the parent directory if needed. If it is already mounted, then automounter establishes a link to the new subdirectory of the common parent.

When using subdirectory fields, an indirect map has a slightly different appearance. Compare the entries in Figure 131 with those shown in Figure 133.

Figure 133 Indirect map entries with a subdirectory field

```
#key            [mount-option...]          location
johnwillow:/home/willow:john
marywillow:/home/willow:mary
joewillow:/home/willow:joe
```

The above example assumes that home directories are of the form /home/user rather than /home/server/user. In general, it is a good idea to provide a subdirectory entry in the location when multiple map entries refer to the same mounted file system from the same server. This approach means less mounts and better response time.

In response to this request

```
ls ~john ~mary
```

automounter performs the equivalent of the following actions:

```
mkdir /tmp_mnt/home/john
mount willow:/home/willow/john /tmp_mnt/home/john
ln -s /tmp_mnt/home/john /home/john

mkdir /tmp_mnt/home/mary
mount willow:/home/willow/mary /tmp_mnt/home/mary
ln -s /tmp_mnt/home/mary /home/mary
```

The complete syntax of a line in a direct or indirect map is actually

```
key[mount-option] server:pathname[:subdirectory]
```

Now, when a user refers to john's home directory, the automounter mounts willow:/home/willow. It then places a symbolic link between /tmp_mnt/home/willow/john and /home/john.

If the user then requests access to mary's home directory, the automounter sees that willow:/home/willow is already mounted, so it only returns the link between /tmp_mnt/home/willow/mary and /home/mary. In other words, the automounter now only does

```
mkdir /tmp_mnt/home/john
mount willow:/home/willow /tmp_mnt/home
ln -s /tmp_mnt/home/john /home/john
ln -s /tmp_mnt/home/mary /home/mary
```

Simplifying map entries

Two conventions reduce the size of map entries. One is the ampersand stating that the key should be substituted into the map. Figure 134 is an indirect map with subdirectores specified, where Figure 135 shows the same entries but substitues the key —the ampersand— for the subdirectory field.

Figure 134 Indirect map with subdirectory field

```
#key                [mount-option...]        location

john                                         willow:/home/willow:john
mary                                         willow:/home/willow:mary
joe                                          willow:/home/willow:joe
able                                         pine:/export/home:able
baker                                        peach:/export/home:baker
```

Wherever the ampersand (&) appears, it translates into the key value.

Figure 135 Indirect map with key substitution

```
#key                [mount-option...]        location

john                                         willow:/home/willow:&
mary                                         willow:/home/willow:&
joe                                          willow:/home/willow:&
able                                         pine:/export/home:&
baker                                        peach:/export/home:&
```

Another example of using the & is when the key and server have the same name. Compare the entries in Figure 137 to those in Figure 137.

Figure 136 Names of key and server are the same

```
#key            [mount-option...]        location

willow                                   willow:/home/willow
peach                                    peach:/home/peach
pine                                     pine:/home/pine
oak                                      oak:/home/oak
poplar                                   poplar:/home/poplar
```

Inserting the ampersand says substitute the key value here. Figure 137 shows how an ampersand is used.

Figure 137 Ampersand signals the insertion of the ke value

```
#key            [mount-option...]        location

willow                                   &:/home/&
peach                                    &:/home/&
pine                                     &:/home/&
oak                                      &:/home/&
poplar                                   &:/home/&
```

You could also use key substitutions in a direct map, in situations like the following:

```
/usr/man        willow,cedar,poplar:/usr/man
```

which is a good candidate to be written as

```
/usr/man        willow,cedar,poplar:&
```

Notice that the ampersand substitution uses the whole key string, so if the key in a direct map starts with a / (as it should), that slash is carried over, and you could not do something like

```
/progs  &1,&2,&3:/export/src/progs
```

because the automounter would interpret it as

```
/progs /progs1,/progs2,/progs3:/export/src/progs
```

The second convention is the asterisk—the wildcard—that matches all keys and provides a value for the & specified in the remainder of the map information. It should be the last or only entry in a map because automounter does not read any further once it reads a wild card entry.

All the entries in Figure 138 have the same format, a good case for the use of the wildcard. The asterisk reduces one line entry to:

```
*&:/home/&
```

where each ampersand is replaced by the value of any given key.

Once the automounter reads the wildcard, it does not continue reading the map, so the Figure 138 map is viable.

Figure 138 Specifying the wildcard

```
#key                [mount-option...]          location

oak                                            &:/export/&
poplar                                         &:/export/&
*                                              &:/home/&
```

whereas, in the map in Figure 139, the last two entries are ignored.

Figure 139 Incorrect placement of the wildcard

```
#key                [mount-option...]          location

*                                              &:/home/&
oak                                            &:/export/&
poplar                                         &:/export/&
```

Using the -hosts built-in map

The -hosts map provides a convenient way for users to access directories in many difference hosts without have to use `rlogin` or `rsh`. Both of these remote commands have to establish communication through the network every time they are invoked.

Automounter has built into it the sematics for the -hosts map. This map only references the hosts databases and the mount is done on /net, indicating that it contains exported filesystems from all over the network. Its entry in a master map would read

```
/net -hosts
```

Calling it from the command line would read

```
automount /net -hosts
```

Automounter can also do a mount of all exported filesystems from a named server (host), in response to an entry

`/net` *servername*

giving users a way to force mounts of all file systems by giving the server's name as a subdirectory of /net

Modifying NIS managed maps

You can modify the automounter maps at any time, but these changes only appear after automounter is restarted. If you want a map modification to take effect immediately, refer to "Invoking and tuning automounter" page 237. Automounter has each direct mount point as an entry in a mount table, so changes to this table cannot be done without automounter. Attempts by NIS to reference a mount point removed from this table returns an error message, even though the mount point is still listed in the mtab file.

Changes to an indirect map must be done at a master server so the map is rebuilt and pushed to the slave servers. Hence, the next time automounter reads the map for a mount request— and automounter has to read indirect maps for each mount request it handles—it will read an updated map.

To change parameters for a currently mounted file system, you must manually unmount it in /tmp_mnt and send the automounter daemon a `SIGHUP` (`kill -1`). When automounter receives this signal, it parses the mtab file and identifies mounted file systems that were unmounted by someone else.

Modifying maps maintained locally on each host

This really isn't needed is it?

Changes to automounter maps not maintained through NIS require the system network manager to update each host's local automounter maps. This task of keeping all maps consistent and current is not much of an improvement over hard-coding /etc/fstab files.

Invoking and tuning automounter

The automount(8) man page contains a complete description of all options. The suboptions are the same as those for a standard NFS mount, except for `bg` (background) and `fg` (foreground) that do not apply.

It is preferable to invoke automounter from rc.local, but it can be invoked from the command line. The syntax to invoke the automounter is:

```
automount [-m -n -T -v] [-Dname=vvalue]\
[-fmaster-file] [-Mmount-directory] \
[-tsub-options] [directory map] [-mount-options]
```

Command specifications that invoke automount

Given the maps defined in Figure 140, Figure 141, and Figure 142 automounter can be invokedfrom the command line, or preferably from /etc/rc.local in one of the following ways:

Figure 140 Sample auto.master map

```
#Mount-point        Map                    Mount options

/net                -hosts
/home               /etc/auto.home         -rw,intr,secure
/-                  /etc/auto.home         -ro,intr
```

- auto.home

Figure 141 Sample auto.home map

```
#key                [mount-option...]      location

willow                                     willow:/home/willow
cypress                                    cypress:/home/cypress
poplar                                     poplar:/home/poplar
pine                                       pine:/export/pine
apple                                      apple:/export/home
ivy                                        ivy:/home/ivy
peach               -rw,nosuid             peach:/export/home
```

- auto.direct \

Figure 142 Sample auto.direct \ map

```
#key                [mount-option...]        location

/usr/local/bin      -ro,soft                 ivy:/export/local/sun3
/usr/local/share    -ro,soft                 ivy:/export/local/share
/usr/local/src      -ro,soft                 ivy:/export/local/src
/usr/man            -ro,soft                 oak:/usr/man
/usr/man            -ro,soft                 rose:/usr/man
/usr/man            -ro,soft                 willow:/usr/man
/usr/games          -ro,soft                 peach:/usr/games
/usr/spool/news     -ro,soft                 pine:/usr/spool/news
/usr/frame          -ro,soft                 redwood:/usr/frame3.0
/ur/frame           -ro,soft                 balsa:/export/frame
```

- Specify all arguments to the automounter without reference to the master map, as in

  ```
  automount /net -hosts /home /etc/auto.home\
  -rw,intr,secure /- /etc/auto.direct -ro,intr
  ```

- Specify the same in the *auto.master* file and instruct the automounter to look there for instructions

  ```
  automount -f /etc/auto.master
  ```

- Specify mount points and maps in addition to those mentioned in the master map as follows

  ```
  automount -f /etc/auto.master /src\ /etc/auto.src
  -ro,soft
  ```

- Nullify one of the entries in the master map. (This is particularly useful if you use a map you cannot modify that does not meet the needs of your machine)

  ```
  automount -f /usr/lib/auto.master /home -null
  ```

- Replace one of the entries with your own

  ```
  automount -f /usr/lib/auto.master /home
  /myown/auto.home -rw,intr
  ```

In the example above, the automounter first mounts all items in the map /myown/auto.home under the directory /home. Then, when it consults the master file /usr/lib/auto.master it ignores the line corresponding to /home, it ignores it because it has already mounted on /home.

Given the auto.master file of the previous example, the first two commands are equivalent as long as your network does not have a distributed auto.master file. This file is available only on networks running NIS. If your network includes a distributed auto.master file, the second example must be modified in the following way to be equivalent to the first example:

```
automount -m -f /etc/auto.master
```

The -m option instructs the automounter not to consult the master file distributed by the NIS. If you do not run NIS, you do not have to specify the -m option. The automounter is completely silent when it does not find a distributed master file.

You can log in as superuser and type any of the above commands at shell level to start the automounter. Ideally, edit rc.local and include your preferred command there.

Using environment variables

You can use the value of an environment variable by prefixing a dollar sign ($) to its name. You can also use braces to delimit the name of the variable from appended letters or digits.

Environmental variables can be inherited from the environment or can be defined explicitly with the command line option -D. For example, if you want each client to mount client-specific files in the network in a replicated format, create a specific map for each client according to its name. The relevant line for host oak would be

```
/mystuff    cypress,ivy,balsa:/export/hostfiles/oak
```

and in willow it would be

```
/mystuff    cypress,ivy,balsa:/export/hostfiles/willow
```

This scheme is viable within a small network, but maintaining this kind of host-specific map across a large network soon becomes unmanageable. The solution for large networks is to invoke the automounter with a command line similar to

```
automount -D HOST='hostname' ......
```

and have the entry in the direct map read

```
/mystuff    cypress,ivy,balsa:/export/hostfiles/$HOST
```

Now each host finds its own files in the mystuff directory, and the task of centrally administering and distributing the maps becomes easier.

After you complete the maps, make sure that there are no equivalent entries in /etc/fstab and that all entries in the maps refer to NFS shared files.

Logging automount activities

The trace option -T tells automount to log its actions to the standard error stream. This option enables you to see the expansion of NFS calls handled by automounter. To use this option as a debugging tool, as part of the automounter start-up command in /etc/rc.local, redirect the output of standard output and standard error to a file. Something like this:

```
automounter -T /-auto.direct >& /tmp/log.nfscalls
```

Naming another directory as mount point

The default name for all mounts is /tmp_mnt. As with other names, this one is arbitrary. It can be changed at invocation by using the `-M` *tmpdir* option. For example:

```
automount -M /auto ...
```

causes all mounts to occur under the directory /auto, which the automounter creates if it does not exist. Do not designate a directory in a read-only file system, as this would prohibit automounter from modifing anything.

A working example

Here is an example that outlines what automounter does when passing a command. With the `automount` daemon in place, issuing

```
cd /net/gumbo
```

signals automounter to start executing these steps:

Step 1 Changes directory to the top of the hierarchy of files (i.e., the root file system) of the machine gumbo, provided the machine is in the hosts database.

Not all files and directories may display under /net/gumbo, because automounter can only mount the shared file systems of host gumbo in accordance with the restrictions placed on the sharing.

Step 2 Pings the null procedure of the server's mount service to see if it's alive.

Step 3 Requests the list of shared hierarchies from the server

Step 4 Sorts the shared list according to length of path name (to ensure proper mounting order):

/usr/src
/export/home
/usr/src/sccs
/export/root/blah

Step 5 Proceeds down the list, mounting all the file systems at mount points in /tmp_mnt (creating the mount points as needed).

Step 6 Returns a symbolic link that points to the top of the recently mounted hierarchy.

For the -hosts map, automounter must mount all the file systems that the server in question advertises for sharing. Even if the request is as follows:

```
ls /net/gumbo/usr/include
```

the automounter mounts all gumbo shared systems, not just /usr.

In addition, unmounting occurs after a certain amount of time has passed, working from the bottom up. This means if one of the directories at the top is busy, the automounter must remount the hierarchy and try again later.

Nevertheless, the -hosts special map provides a convenient way for users to access directories in many different hosts without having to use `rlogin` or `rsh`. (These remote commands have to establish communication through the network every time they are invoked.)

Notice that both /net and /home are arbitrary names dictated by convention. The automounter creates them if they do not already exist.

Mount table

Every time the automounter mounts or unmounts a file hierarchy, it modifies /etc/mtab to reflect the current situation. The automounter keeps an image of /etc/mtab in memory and refreshes this image every time it performs a mount or an automatic unmount. If you use `umount` to unmount one of the automounted hierarchies (a directory under /tmp_mnt), force the automounter to reread the /etc/mtab file. To do so, enter

```
ps -aux | grep automount | egrep -v grep
```

This gives you the process ID of the automounter. Automounter is designed to reread /etc/mtab when it receives a SIGHUP signal. To send it that signal, enter

```
% kill -1 PID
```

where *PID* stands for the process ID you obtained from the `ps` command.

Error messages

Table 5 lists error messages displayed if the automounter fails, and explains what each message indicates.

There are conditions and actions that need to be defined.

Table 5 Error messages related to automount

Error message	Condition cause with recommended action
no mount maps specified	Automounter was invoked with no maps to serve, and it cannot find the NIS auto.master map. This message is produced only when the *-v* option is given. Recheck the command, or restart NIS if that was the intention.
mapname: Not found	The required map cannot be located. This message is produced only when the *-v* option is given. Check the spelling and pathname of the map name.
dir mountpoint must start with '/'	Automounter mount point must be given as full pathname. Check the spelling and pathname of the mount point.
mountpoint: Not a directory	The mount point exists but it is not a directory. Check the spelling and pathname of the mount point.
hierarchical mountpoint: mountpoint	The automounter does not allow itself to be mounted within an automounted directory. You will have to use another strategy.
WARNING: mountpoint not empty!	The mount point is not an empty directory. This message is produced only when the -v option is given and it is only a warning. It means that the previous contents of the mount point will not be accessible.
Can't mount mountpoint: reason	Automounter cannot mount itself at mount point. The reason given with the message should be self-explanatory.
hostname:file system already mounted on mountpoint	Automounter has been mounted on an already mounted-on mount point and is attempting to mount the same file system there. This will happen if an entry in /etc/fstab also appears in an automounter map (either by accident or because the output of mount -p was redirected to fstab). Delete one of the redundant entries.
WARNING: hostname:file system already mounted on mountpoint	Automounter is mounting itself on top of an existing mount point.

Table 5 Error messages related to automount (continued)

Error message	Condition cause with recommended action
couldn't create directory: reason	Could not create a directory. The reason should be self-explanatory.
bad entry in map mapname "map entry"	
map mapname, key map key: bad	The map entry is malformed, and the automounter cannot interpret it. Recheck the entry; perhaps there are characters in it that need escaping.
mapname: yp_err	Error looking up an entry in a NIS map.
hostname: exports: rpc_err	Error getting share list from hostname. This indicates a server or network problem.
host hostname not responding	
hostname:filesystem server not responding	
Mount of hostname:filesystem on mountpoint: reason	You will see these error messages after the automounter attempt ed to mount from hostname but either got no response or failed. This may indicate a server or network problem.
mountpoint - pathname from hostname: absolute symbolic link	When mounting a hierarchy, the automounter has detected that mountpoint is an absolute symbolic link (begins with "/"). The content of the link is pathname. This may have undesired consequences on the client.
Cannot create socket for broadcast rpc: rpc_err	
Many_cast select problem: rpc_err	
Cannot send broadcast packet: rpc_err	
Cannot receive reply to many_cast: rpc_err	All these error messages indicate problems attempting to ping servers for a replicated file system. This may indicate a network problem.
trymany: servers not responding: reason	No server in a replicated list is responding. This may indicate a network problem.

Table 5 Error messages related to automount (continued)

Error message	Condition cause with recommended action
Remount hostname:filesystem on mountpoint: server not responding	Attempted remount after unmount failed. Indicates a server problem.
NFS server (pidn@mountpoint) not responding still trying	An NFS request made to the automount daemon with PID *n* serving mountpoint has timed out. The automounter may be temporarily overloaded or dead. Wait a few minutes; if the condition persists, the easiest solution (for a diskless client) is to reboot the client. If this is not possible, exit all processes that use automounted directories (or, change to a nonautomounted directory in the case of a shell), kill the current automount process and restart it again from the command line. If this does not work, you must reboot.
path: <pathname> does not exist on host: <hostname>	Where <pathname> is the pathname that was mounted from the remote system (including subdirectories from indirect maps using ':') and <hostname> is the name of the NFS server. To corect the problem, create the specified directory on the NFS server system.
Domain name not set. Cannot use YP.	
YP bind failed: can't communicate with ypbind.	
mount: serverhost:/exported/fs server no responding: RPC: Authentication error; why = Invalid client credential mount: giving up on: /exports/fs	

Managing NIS

23

The Network Information System (NIS), formerly known as Yellow Pages (YP), is a distributed lookup service that maintains a centralized set of configuration files that would otherwise be maintained as ASCII files on each local system. Networked hosts query a central server for this information as they would a database.

NIS greatly simplifies management of such configuration files as /etc/hosts, /etc/passwd, /etc/group, /etc/aliases, and /etc/ethers because it creates a single point of administration and gives all hosts access to a recent version of the data. NIS clients and servers retrieve needed information from maps—database tables detrived from these centralized configuration files.

This chapter explains procedures for setting up and troubleshooting NIS. These procedures include

- Setting up an NIS server
- Setting up an NIS client
- Modifying individual NIS maps
- Propagating an NIS map
- Making new NIS maps after installing NIS
- Adding an additional NIS server
- Changing the master server to a different host
- Accessing and modifying NIS maps
- Disabling NIS
- Troubleshooting servers, clients, and daemons
- Managing access: NIS security

NIS uses network services provided by CONVEX Internet Services. Therefore, before beginning any task described in this chapter, you must successfully complete the following tasks:

- Install ConvexOS and Utilities
- Install CONVEX Internet Services
- Install CONVEX NFS
- Configure Internet Services

For information about installing the products listed above, refer to the installation procedures that accompanied the software distribution tape. For information about configuring Internet Services, refer to Part 2, "Part 2 Configuring network interfaces," page 7.

Setting up an NIS server

This section explains how to

- Prepare a host to become a master NIS server
- Set up a master NIS server
- Set up a slave NIS server

Preparing to set up a master NIS server

Complete the following procedure before you begin setting up a master NIS server:

Step 1 Become root.

Step 2 Select an NIS domain name, if it is to differ from the name selected for your network domain during installation. Make sure your rc.local script contains a `domainname` command that defines your NIS domain. The domain name is defined in rc.local as follows:

```
/bin/domainname your-domain-name
```

Step 3 Make sure the host name is defined in rc.local. If you have configured CONVEX Internet Services, you have already defined the host name. The host name is defined in rc.local as follows:

```
/bin/hostname local-host-name
```

Step 4 Ensure that the following files in /etc are complete and up to date:

- passwd
- hosts

- ethers
- group
- networks
- protocols
- services
- pwrestrict

Step 5 If you know how /etc/netgroup is going to be set up, set it up now. If you do not set up /etc/netgroup at this time, `ypinit` makes an empty netgroup map.

Step 6 Edit /usr/lib/aliases if it needs to be updated.

Step 7 For security reasons, you may restrict access to the master NIS server to a smaller set of users than that defined by the complete /etc/passwd file. To do this, copy the complete /etc/passwd file to some place other than /etc/passwd. Edit out undesired users from the remaining /etc/passwd using `vipw`. (Refer to the vipw(8) man page for details.) For a security-conscious site, this smaller file should not include the NIS escape entry discussed in the next section.

After performing these steps, you are ready to create a new master server.

Setting up a master NIS server

Step 1 Log in as superuser and change directory to /usr/etc/yp.

Step 2 Run `ypinit`. Figure 143 shows an example run of `ypinit` on a master server.

`ypinit` asks you whether you want the procedure to exit at the first nonfatal error. (If `ypinit` exits, you can fix the problem and restart `ypinit`. This option is recommended if you have never done this procedure.) You may elect to continue despite nonfatal errors. In this case, you can try to fix all the problems by hand, or fix some, then restart `ypinit`. `ypinit` prompts you for a list of other hosts that will also be NIS servers. (Initially, this is the set of NIS slave servers, but in the future, any of them might become the NIS master server.) You need not add any other hosts at this time, but if you know that you will be setting up more NIS servers, add them now. You will save yourself some work later, and there is little runtime penalty for doing it.

Figure 143 Running `ypinit` to build master server databases

```
host1# cd /usr/etc/yp
host1# ypinit
usage:
                ypinit -m
                ypinit -s master_server
where -m is used to build a master NIS server data base, and -s is used for
slave data base.  master_server must be an existing reachable NIS server.
host1# ypinit -m
The local host's domain name hasn't been set.  Please set it.
host1# domainname host1
host1# ypinit -m
Installing the NIS data base will require that you answer a few questions.
Questions will all be asked at the beginning of the procedure.
Do you want this procedure to quit on non-fatal errors? [y/n: n]  y
Can we destroy the existing /usr/etc/yp/host1 and its contents? [y/n: n]  y
At this point, we have to construct a list of the hosts which will run NIS
servers.  host1 is in the list of NIS server hosts.  Please continue to add
the names for the other hosts, one per line.  When you are done with the
list, type a <ctl D>.
                next host to add:  host2
                next host to add:  CONTROL-D
The current list of NIS servers looks like this:
host1
host2
Is this correct?  [y/n: y]  y
There will be no further questions. The remainder of the procedure should
take 5 to 10 minutes.
Building /etc/yp/host1/ypservers...
Running /etc/yp/Makefile...
updated ypservers
updated passwd
updated group
updated hosts
updated ethers
updated networks
updated rpc
updated services
updated protocols
updated pwrestrict
updated netgroup
updated bootparams
10 aliases, longest 51 bytes, 310 bytes total
updated aliases
updated publickey
updated netid
host1 has been set up as a NIS master server without any errors.
If there are running slave NIS servers, run yppush now for any data bases
that have been changed.  If there are no running slaves, run ypinit on hosts
that are to be slave servers.
```

Step 3 Run ypserv and ypbind.

Step 4 Check rc.local for lines shown in Figure 144

Figure 144 Starting a master NIS server from rc.local

```
if [ -f /bin/domainname ] ; then
    if [ "`/bin/domainname`" != "" ]; then
        if [ -f /usr/etc/ypserv -a -d /etc/yp/`/bin/domainname`] ; then
            /usr/etc/ypserv & echo -n ' ypserv'
        fi
        if [ -f /usr/etc/ypbind ]; then
            /usr/etc/ypbind & echo -n ' ypbind'
        fi
        if [ -f /usr/etc/rpc.yppasswdd ]; then
            /usr/etc/rpc.yppasswdd /etc/passwd /etc/pwrestrict \
            -m passwd pwrestrict DIR=/etc & echo -n 'yppasswdd'
        fi
    fi
fi
```

If these lines do not already exist in your rc.local file, add them. ypserv will then start up automatically from rc.local every time the server boots.

Using a non-CONVEX master server

If you use a non-CONVEX host as the master NIS server, you must make a change to the NIS configuration on that host. CONVEX NIS expects the NIS services map and RPC map to be built so that they may be searched quickly with a single yp_match access. Sun Microsystems and some other vendors do not build these maps, necessitating a change to the master NIS makefile so the maps are built to allow the CONVEX system to find the information it needs. This change does not interfere with NIS accesses by machines from other vendors,because it only supplies additional information for the CONVEX system.

To make this change, copy lines from the /usr/etc/yp/Makefile file on the CONVEX system. Find the string "services.time"; copy all lines down to the string "protocols.time" to a temporary file, and move the temporary file onto the non-CONVEX machine.

On a Sun running SunOS version 4.0 or greater, look for the Makefile in /var/yp; on older Suns, look for it in /usr/etc/yp. Go to that directory and replace the appropriate lines in the Makefile with those from the temporary file.

The lines copied from the CONVEX Makefile contain several occurrences of `DOMDIR`. Edit these lines, replacing `DOMDIR` with the macro name used by the non-CONVEX vendor. On most Suns, this macro is $(YPDBDIR/$(DOM).

Once the Makefile is ready, remove the files services.time and rpc.time, and build the new maps with `make`. Now the CONVEX system can access the services and RPC maps correctly.

Setting up a slave NIS server

The network must be operational before you can set up a slave NIS server. In particular, you must be able to copy files from the master NIS server to NIS slaves. To create a new slave server, follow this procedure:

Step 1 Change directory to /usr/etc/yp.

Step 2 Become superuser.

Step 3 Select a host already set up as a master NIS server.

Ideally, the host you select is really the master server, but it can be any host that has its NIS database set up and is reachable over the network.To determine whether the host is reachable, enter

`rpcinfo -b ypserv 2 | grep` *hostname*

The server in question is reachable if it appears in the output list.

Step 4 Check that an entry for "daemon" exists in the /etc/passwd files of both slave and master, preceding any other entries that have the same user ID.

Step 5 Set the default domain name on the host intended to be the slave NIS server to be the same as the default domain name on the host named as the master.

Step 6 Run `ypinit` with the **`-s`** option.

An sample run of ypinit on a slave server is shown in Figure 145.

Figure 145 Sample run of ypinit for slave l

```
goofy# cd /usr/etc/yp
goofy# ypinit -s mickey
Installing the yp data base will require that you answer a few questions.
Questions will all be asked at the beginning of the procedure.
Do you want this procedure to quit on non-fatal errors? [y/n: n]  y
There will be no further questions. The remainder of the procedure should
take a few minutes, to copy the data bases from mickey.
Transferring ethers.byaddr...
Transferring ethers.byname...
Transferring group.bygid...
Transferring group.byname...
Transferring hosts.byaddr...
Transferring hosts.byname...
Transferring mail.aliases...
Transferring netgroup...
Transferring netgroup.byuser...
Transferring netgroup.byhost...
Transferring networks.byaddr...
Transferring networks.byname...
Transferring passwd.byname...
Transferring passwd.byuid...
Transferring protocols.byname...
Transferring protocols.bynumber...
Transferring services.byname...
Transferring ypservers...
Transferring services...
Transferring rpc.byname...
Transferring rpc.bynumber...
Transferring pwrestrict.byname...
Transferring pwrestrict.byuid...
Transferring bootparams...
Transferring publickey.byname...
Transferring netid.byname
goofy's NIS data base has been set up without any errors.
```

At this point, make sure that /etc/passwd, /etc/hosts, /etc/networks, /etc/group, /etc/protocols, /etc/services, /etc/rpc, /etc/pwrestrict, and /etc/netgroup have been edited so that when NIS is activated, the data bases you have just created will be used, instead of the /etc ASCII files.

As stated in "Setting up a master NIS server" page 249, you are not prompted for a list of other servers, but you do have the opportunity to choose whether or not the procedure gives up at the first nonfatal error.

Step 7 After running `ypinit`, make copies of /etc/passwd, /etc/hosts, /etc/group, /etc/networks, /etc/protocols, /etc/netgroup, and /etc/services. For example, to make a copy of /etc/passwd, enter

```
cp /etc/passwd /etc/passwd-
```

Step 8 Edit the original files according to the instructions in "Altering NIS client files to use NIS services" page 255, to ensure that processes on the slave NIS server actually use the NIS services, rather than local ASCII files. (That is, make sure the NIS slave server is also a NIS client.)

Step 9 Make backup copies of the edited files. For example,

```
cp /etc/passwd /etc/passwd+
```

Step 10 To give the slave access to domain maps, run `ypbind` from the slave by entering

```
/usr/etc/ypbind
```

Step 11 After the NIS database is set up by `ypinit`, to begin providing NIS services, enter

```
/usr/etc/ypserv
```

When the system is booted, `ypserv` will be started automatically from /etc/rc.local.

Setting up NIS clients

This section explains how to

- Set up NIS clients
- Alter an NIS client's database to use NIS services

Setting up an NIS client

To set up an NIS client, perform these steps:

Step 1 Edit the local files to use NIS services as described in "Altering NIS client files to use NIS services" page 255.

Step 2 If `ypbind` is not already running, start it.

Step 3 Make sure that /etc/rc.local contains the proper line to set the domainname. This line is of the format

`/bin/domainname` *your-domain-name*

Step 4 With the ASCII databases of /etc abbreviated and `ypbind` running, processes on the host will be clients of the NIS services.

At this point, there must be an NIS server available. Processes will hang if no NIS server is available while `ypbind` is running.

Note the possible alterations to the client's /etc database as discussed in "Altering NIS client files to use NIS services" page 255. Because some files may not be there, or some may be specially altered, it is not always obvious how the ASCII databases are being used. The escape conventions used within those files to force data to be included or excluded from the NIS databases are found in the following man pages: passwd(1), hosts(5), netgroup(5), hosts.equiv(5), and group(5). In particular, notice that changing passwords in /etc/passwd by editing the file, or by running the `passwd` command affects only the local client's environment. Change the client's NIS password database by running `yppasswd`

Step 5 To add a new client diskless workstation to a network already running NIS, use the `setup_client` program described in Chapter 21, "Using NETdisk for booting diskless workstations," page 189. Use the same syntax described in this chapter. Be sure to specify client for `yp_type`.

Altering NIS client files to use NIS services

Once you decide to run NIS at your site, you should have all hosts on the network access the NIS maps, instead of potentially out-of-date information in their local ASCII configuration files. To enforce the use of NIS maps, run a `ypbind` process on each client, including clients that may also be NIS servers, and abbreviate or eliminate the files traditionally implemented the NIS maps, namely

/etc/passwd
/etc/hosts
/etc/ethers
/etc/group
/etc/networks
/etc/protocols
/etc/services
/etc/netgroup
/etc/aliases
/etc/netmasks
/.rhosts

The treatment of each file is discussed in this section.

- /etc/networks, /etc/protocols, /etc/ethers, /etc/services, and /etc/netgroup need not exist on any NIS clients. These files are never consulted.

- /etc/hosts.equiv is never served by NIS. However, you can add escape sequences to reference NIS. This reduces problems with `rlogin` that are sometimes caused by different /etc/hosts.equiv files on the two hosts.

  To let anyone log in to a host, edit /etc/hosts.equiv to contain a single line, with only the character, + (plus) on it. A line containing only a + means that all further entries are retrieved from NIS rather than the local file.

  Alternatively, you can exercise more control over logins by using lines of the form:

  +@trusted_group1
  +@trusted_group2
  -@distrusted_group

  where the name to the right of an at sign (@) is interpreted as a netgroup name defined in the global netgroup database. The netgroup database is served by NIS.

  If neither the + or - is used, only entries in /etc/hosts.equiv are used; NIS is not used.

- /.rhosts is never served by NIS. Its format is identical to that of /etc/hosts.equiv; however, because this file controls remote root access to the local host, allowing unrestricted access to it is not recommended. Make the list of trusted hosts explicit, or use netgroup names for the same purpose. You can not use secondary hostnames in your .rhosts hosts.equiv or netgroup files. You can, however, use secondary hostnames in /etc/hosts. All of the above files enable local hosts to access remote hosts in some fashion.

- /etc/hosts must contain entries for the local host's name, and the local loopback name. These are accessed at boot time when the NIS service is not yet available. After the system is running, and after the `ypbind` process is up, the /etc/hosts file is not accessed at all.

- /etc/passwd should contain entries for the user name, `root`, and the primary users of the host, and the + escape entry to force the use of the NIS service. A few additional entries are recommended: `daemon`, to allow file-transfer utilities to work; and `operator`, to let a dump operator log in. An NIS client's /etc/passwd file might look like

```
root:9wxntql2tHT.k:0:1:Operator:/:/bin/csh
nobody:*:-2:-2::/:
daemon:*:1:1::/:
sys:*:2:2::/:/bin/csh
bin:*:3:3::/bin:
uucp:*:4:4::/var/spool/uucppublic:
news:*:6:6::/var/spool/news:/bin/csh
sync::1:1::/:/bin/sync
raks:7kjDXZD/Hug2s:624:20:Stefania:/home\
/dancer/raks:/bin/csh
+::0:0:::
```

  The last line informs the library routines to use the NIS service. If you remove the last line in the passwd file, you will disable NIS password access.

  A program that uses `getpwent` (refer to the getpwnet(3) man page) to access /etc/passwd first looks in the password file on your host; it then looks in the NIS password file (only if your host's password file contains + (plus sign) entries), as shown in the above example. Earlier entries in the file take precedence over, or mask, later ones with the same user name, or the same user ID. Therefore, please note the order of the entries for daemon and for sync (which have the same user ID) and duplicate this order in your own file.

- /etc/group may be reduced to a single line

```
+:
```

  which forces all translations of group names and group IDs to be made via the NIS service. This is the recommended procedure.

Administering NIS changes

Administering NIS takes place after the setup procedures have been completed. Data accessibility requirements over networks never remain stagnant; changes will always be needed. Creating and updating maps, adding new servers, changing the master server are the type of tasks you will need to do. This section explains how to make these changes. It includes instructions for

- Modifying existing maps
- Propagating NIS maps
- Creating new NIS maps after installing NIS
- Adding an additional NIS server
- Changing the master server to a different host

Modifying existing maps

Always make changes to the databases served by NIS on the master server. To modify the databases you expect to change most frequently, such as /etc/passwd, follow these steps:

Step 1 Become superuser.

Step 2 Edit the ASCII database file.

Step 3 Change directory to /usr/etc/yp.

Step 4 Run `make` (`makedbm`).

Nonstandard maps may be modified manually. These maps include

- maps specific to applications of a particular vendor or site
- maps that change infrequently
- maps for which no ASCII form exists; for example, maps created when you installed NIS.

Use `makedbm` with the `-u` option to disassemble them into a form that can be modified using standard tools, such as `awk`, `sed`, or `vi`.

Step 5 Build a new version using `makedbm`, which can be done manually using either of two methods.

- Redirect the output of `makedbm` to a temporary file that can be modified, then fed back into `makedbm`.
- Operate on the output of `makedbm` within a pipeline that feeds into `makedbm` again directly. This is appropriate if the disassembled map can be updated by modifying it with `awk`, `sed`, or a `cat` append, for instance.

Suppose you want to create a nonstandard NIS map, called mymap. You want it to consist of key-value pairs in which the keys are strings like al, bl, cl, etc. (l is for left), and the values are ar, br, cr (r is for right). There are two possible procedures to follow when creating new maps: one uses an existing ASCII file as input; the other uses standard input.

Creating new maps from existing ASCII files

For example, existing ASCII file named /usr/etc/yp/mymap.asc was created with an editor or a shell script. The map is located in the subdirectory, home_domain. Create the NIS map for this file by entering:

```
cd /usr/etc/yp
makedbm mymap.asc home_domain/mymap
```

At this point, you notice the map really should have included another 2-tuple: (dl, dr). You can make the modification simply. In all situations like this, remember to modify the map by first modifying the ASCII file. Modifications made only to the dbm files, and not also in the ASCII file, will be lost. Make the modification as follows:

```
cd /usr/etc/yp
makedbm mymap.asc home_domain/mymap
```

Creating new maps from standard input

When there is no original ASCII file, create the NIS map from the keyboard by entering input as follows:

```
cd /usr/etc/yp
makedbm - home_domain/mymap
al ar
bl br
cl cr
CTRL-D
```

When you need to modify that map, use `makedbm` to create a temporary ASCII intermediate file that you can edit using standard tools. For instance:

```
cd /usr/etc/yp
makedbm -u home_domain/mymap > mymap.temp
```

At this point, you can edit mymap.temp to contain the correct information. Create a new version of the database using the commands

```
makedbm mymap.temp home_domain/mymap
rm mymap.temp
```

The preceding paragraphs explained how to use a few tools. In reality, almost everything you must do can be done by `ypinit` and /usr/etc/yp/Makefile. If you add nonstandard maps to the database, or change the set of NIS servers after the system is already up and running, other procedures such as those explained above are required.

Whether you use the Makefile in /usr/etc/yp or some other procedure, the goal is the same: a new pair of well-formed dbm files must end up in the domain directory on the master NIS server.

Propagating map changes from master to slaves

Maps must be changed on the master, then propagated from the master NIS server to all slave NIS servers. Initially, `ypinit` moves changes, as described in "Setting up a slave NIS server" page 252. After a slave NIS server has been initialized, updated maps are transferred from the master server by `ypxfr`. You can force the propagation of changes by running `ypxfr` using any one of three different methods: via `cron`, via `ypserv`, or interactively. An explanation of each method follows.

Scheduling map updates using `cron`

Maps have differing rates of change; for instance, protocols.byname may not change for months at a time, but passwd.byname may change several times a day in a large organization. You can set up entries in a crontab file to periodically run `ypxfr` at a rate appropriate for any map in your NIS database. `ypxfr` contacts the master server and transfers the map only if the master's copy is more recent than the local copy.

To avoid needing a crontab entry for each map, several maps with approximately the same change characteristics can be grouped in a shell script, and the shell script can be run from a crontab file. Suggested groupings, mnemonically named, can be found in /usr/etc/yp, ypxfr_1perhour, ypxfr_1perday, and ypxfr_2perday. If the rates of change used by these scripts are inappropriate for your environment, you can easily modify or replace these shell scripts. For your site, look to see which maps are to be updated by each of these timed shell scrips.

These same shell scripts should be run at each NIS slave server in the domain. Alter the exact time of execution from one server to another, so as to prevent the checking from bogging down the master. If you want the map transferred from some particular server, not the master, you can specify that server within the

shell script by using the `ypxfr -h` option. You can use the **`-s`** option to transfer maps from another domain. Finally, maps having unique change characteristics can be checked and transferred by explicit invocations of `ypxfr` within the system crontab file, /.crontab.

Starting update cycle from the master server with `ypserve`

You may also send files out from the master server, rather than asking for them on the slave server. `yppush` sends a transfer Map RPC request to `ypserv`, which invokes `ypxfr` to send the map. `yppush` enumerates the NIS map ypservers, generating a list of NIS servers in your domain, and sends a Transfer Map request to each of the listed NIS servers. `ypserv` forks off a copy of `ypxfr`, invokes it with the `-C` flag, and passes it the information it needs to identify the map and call back the initiating `yppush` process with a summary status.

Entering `ypxfr` interactively

You can run `ypxfr` as a command. Typically, you do this only in exceptional situations —such as, when setting up a temporary NIS server to create a test environment, or when trying quickly to get a NIS server that has been out of service consistent with the other servers.

Logging `ypxfr` activities

In most situations, except when started interactively, `ypxfr` transfer attempts and results can be captured in a log file. If /usr/etc/yp/ypxfr.log exists, results are appended to it. No attempt to limit the log file is made. To turn off logging, remove the log file.

Adding new maps

Adding a new NIS map entails getting copies of the map's dbm files into the domain directory on each of the NIS servers in a domain. This section describes how to get the proper files in place so the propagation works correctly. Both master and slaves must be set up correctly.

After deciding which NIS server is to be the master of the map, modify /usr/etc/yp/Makefile on the master server so that the map can be conveniently rebuilt. Filter an ASCII file through `awk`, `sed`, and/or `grep` to make it suitable for input to `makedbm`.

Consult the existing Makefile as a source for programming examples. Use the mechanisms already in place in /usr/etc/yp/Makefile when deciding how to create dependencies that `make` will recognize; specifically, the use of .time files allows you to see when the Makefile was last run for the map.

To get an initial copy of the map, run `yppush` on the NIS master server. The map must be globally available before clients begin to access it. If the map is available from some NIS servers, but not all, you will see unpredictable behavior from client programs.

Adding a new NIS server

To add a host that has never been an NIS slave server, make the following modifications on the master NIS server:

Step 1 Add the host's name to the ypservers map in the default domain. If a host name is not in ypservers, it will not be notified of updates to the NIS map files.

To illustrate, the command sequence for adding a server to domain_name is

```
vi /etc/ypservers
cd /usr/etc/yp
make
```

Step 2 Set up the new slave NIS server's databases by copying the databases from NIS master server, ypmaster. To do this, remote log in to the new NIS slave, and run the `ypinit` command. To continue on with the example, the commands are

```
cd /usr/etc/yp
ypinit -s ypmaster
```

Step 3 Perform the steps described in "Setting up a slave NIS server" page 252.

Changing to a new master server

Step 1 Build the map at the new master. Because the old NIS master's name occurs as a key-value pair in the existing map, it is not sufficient to use an existing copy at the new master, or to send a copy there with `ypxfr`. The key must be reassociated with the new master's name. If the map has an ASCII source file, it should be present in its current version at the new master.

/etc/yp/Makefile must be set up correctly for this to work. If it is not, do it now. Remake the NIS map (called belmont.mass below) locally with the sequence

```
cd /usr/etc/yp
make belmont.mass
```

Step 2 Go back to the old master (if it is to remain an NIS server) and edit /usr/etc/yp/Makefile so that belmont.mass is no longer made there; that is, comment out the section of oldmaster:/etc/yp/Makefile that made belmont.mass.

Step 3 If the map exists only as a dbm database, you can remake it on the new master by disassembling an existing copy (one from any NIS server will do) and running the disassembled version back through makedbm. For example:

```
cd /usr/etc/yp
ypcat -k belmont.mass|\
makedbm - mydomain/belmont.mass
```

Step 4 After making the map on the new master, send a new copy of the map to the other (slave) NIS servers. However, do not use yppush; the other slaves will try to get new copies from the old master instead of from the new one. Rather, become superuser on the old master server and enter

```
/usr/etc/yp/ypxfr -h newmaster\ belmont.mass
```

Step 5 Now that you have a new copy on the old master, run yppush. The remaining slave servers still believe that the old master is the current master, and will attempt to get the current version of the map from the old master. When they do so, they will get the new map, which names the new master as the current master.

If the method above fails, there is a cumbersome but always effective option. On each NIS server machine, while superuser, execute the command shown just above. This will certainly work, but should be considered the worst case solution.

Troubleshooting an NIS client

This section provides suggestions to help you solve network problems encountered on NIS clients.

Hanging commands

Commands may be in a hang state even though the system as a whole seems okay and you are able to run new commands. The most common problem at an NIS client node is for a command to hang and generate the console message:

```
NIS: server not responding for domain <wigwam>.  Still trying
```

The message above indicates that `ypbind` on the local host is unable to communicate with `ypserv` in the domain wigwam. This often happens when hosts running `ypserv` crash. It may also display if the network or the NIS server is so overloaded that `ypserv` cannot get a response back to your `ypbind` within the time-out period. Under these circumstances, all the other NIS client nodes on your net show the same or similar problems. This condition is temporary in most cases. The message usually goes away when the NIS server reboots and `ypserv` gets back in business, or when the load on the NIS server nodes or the Ethernet decreases.

On the other hand, in the circumstances described below, this condition does not resolve itself.

- The NIS client has not set, or has incorrectly set, domain name on the host. Clients must use a domain name that the NIS servers know. Use `domainname` to see the client domain name. Compare that with the domain name set on the NIS servers. The domain name should be set in /etc/rc.local. When /etc/rc.local fails to set or incorrectly sets domainname, do the following: become superuser on the host in question, edit /etc/rc.local to fix the domainname line with a proper domain name (this assures domain name will be correct every time the host boots), and set domainname manually so it is fixed immediately. To do this, enter:

  domainname *good_domain_name*

- If your domain name is correct, make sure your local network has at least one NIS server. You can automatically bind only to a `ypserv` process on your local network, not on another accessible network. There must be at least one NIS server for your host's domain running on your local network. Two or more NIS servers will improve availability and response characteristics for NIS services.

- If your local network has an NIS server, make sure it is up and running. Check other hosts on your local net. If several clients have problems simultaneously, suspect a server problem. Find a client behaving normally, and try the `ypwhich` command. If `ypwhich` never returns an answer, kill it and go to a terminal on the NIS server by entering

  ```
  ps ax | grep yp
  ```

  and look for `ypserv` and `ypbind` processes. If the server's `ypbind` daemon is not running, start it up by entering

  ```
  /usr/etc/ypbind
  ```

  If there is a `ypserv` process running, do a `ypwhich` on the NIS server. If `ypwhich` returns no answer, `ypserv` has probably hung and should be restarted. Kill the existing `ypserv` process (you must be logged on as root), and start `/usr/etc/ypserv`, as follows:

  ```
  kill -9 pid
  /usr/etc/ypserv
  ```

  If `ps` shows no `ypserv` process running, start one.

NIS service unavailable

When other hosts on the network appear to be okay, but NIS service becomes unavailable on your host, different types of symptoms may show up. Symptons such as:

- Some commands appear to operate correctly while others terminate.
- An error message prints about the unavailability of NIS.
- Some commands limp along in a backup-strategy mode particular to the program involved.
- Some commands or daemons crash with obscure messages or no message at all.

Here are some examples of what you might see when one of these conditions occurs.

- my_machine% **ypcat myfile**
  ypcat: can't bind to NIS server for domain <wigwam>.

  Reason: cannot communicate with ypbind.

- my_machine% **/usr/etc/yp/yppoll myfile**
  Sorry, I can't make use of the yellow pages. I give up.

When symptoms like those shown in items 1 and 2 occur, try entering

```
ls -l
```

on a directory containing files owned by many users, including users not in the local host's /etc/passwd file; for example /usr. When the `ls -l` reports file owners, not in the local host's /etc/passwd file, as numbers rather than names, this is one more symptom that NIS service is not working.

These symptoms usually indicate that your ypbind process is not running. Run **ps ax** to check for it. If it is not running, start it by entering

```
/usr/etc/ypbind
```

NIS problems should disappear.

ypbind crashes

If ypbind crashes almost immediately each time it is started, you should look for a problem in some other part of the system. Check for the presence of the portmap daemon by entering:

```
ps ax | grep portmap
```

If you do not find it running, take the CONVEX host down to single-user mode, then return it to multiuser mode. Because so many other daemons must be started after portmap, this is the least disruptive means of correcting the problem.

If portmap itself will not stay up or behaves strangely, look for more fundamental problems. Check the network software.

You may be able to talk to the portmap on your host from a host operating normally. From such a host, enter

```
rpcinfo -p client
```

Where *client* is the name of your host. If your `portmap` is okay, the output should look like that shown in Figure 146.

Figure 146 `portmap` output

```
program    vers     proto     port
100007     2        tcp       1024     ypbind
100007     2        udp       1028     ypbind
100007     1        tcp       1024     ypbind
100007     1        udp       1028     ypbind
100021     1        tcp       1026     nlockmgr
100024     1        udp       1052     status
100021     1        udp       1054     nlockmgr
100024     1        tcp       1027     status
100020     1        udp       1058     llockmgr
100020     1        tcp       1028     llockmgr
100021     2        tcp       1029     nlockmgr
100012     1        udp       1083     sprayd
100011     1        udp       1085     rquotad
100005     1        udp       1087     mountd
100008     1        udp       1089     walld
100002     1        udp       1091     rusersd
100002     2        udp       1091     rusersd
100001     1        udp       109      rstatd
100001     2        udp       1094     rstatd
100001     3        udp       1094     rstatd
```

On your host, the port numbers will be different. The four entries that represent the ypbind process are shown in Figure 146.

Figure 147 Entries representing `ypbind`

```
100007     2        tcp       1024     ypbind
100007     2        udp       1028     ypbind
100007     1        tcp       1024     ypbind
100007     1        udp       1028     ypbind
```

If they are not there, `ypbind` has been unable to register its services. Reboot the host. If they are there and they change each time you try to restart `/usr/etc/ypbind`, reboot the system, even if the `portmap` is up. If the situation persists after reboot, call the CONVEX Technical Assistance Center.

`ypwhich` inconsistent

When you use `ypwhich` several times at the same client node, the answer you get back varies, whereas, the NIS server changes. This is normal. The binding of NIS client to NIS server changes over time on a busy net and when the NIS servers are busy. Whenever possible, the system stabilizes at a point where all clients get acceptable response time from the NIS servers. As long as your client gets NIS service, it doesn't matter where the service comes from. Often an NIS server gets its own NIS services from another NIS server on the net.

Troubleshooting an NIS server

This section provides suggestions to help you solve network problems encountered on NIS servers.

Multiple versions of an NIS map

Because NIS works by propagating maps among servers, you will sometimes find different versions of a map at servers on the network. This version skew is normal if transient, and abnormal otherwise.

Normal update is prevented when some NIS server or some router between NIS servers is down during a map transfer attempt. Normal updating of maps is described in the "Propagating map changes from master to slaves" page 260. When all the NIS servers, and all the routers between them, are up and running, `ypxfr` should succeed.

If a particular slave server has problems updating, log in to that server and run `ypxfr` interactively. If `ypxfr` fails, it will tell you why it failed, then you can fix the problem. If `ypxfr` succeeds, but you think it has been failing sometimes, create a log file to enable logging of messages by entering

```
cd /usr/etc/yp
touch ypxfr.log
```

This saves all output from `ypxfr`. The output looks much like what `ypxfr` creates when run interactively, except each line in the log file is time-stamped. You may see unusual orderings in the time stamps, which is okay. The time stamp tells you when `ypxfr` began its work. If copies of `ypxfr` ran simultaneously, but their work took differing amounts of time, they may actually write their summary status line to the log files in an order different from the order in which they were invoked.) Any pattern of intermittent failure shows up in the log. When you have fixed the problem, turn off logging by removing the log file. If you forget to remove it, it will grow without limit.

While still logged in to the problem NIS slave server, inspect the system crontab file, `/usr/spool/cron/crontabs/root` and the `ypxfr*` shell scripts it invokes. Typos in these files cause propagation problems, as do failures to refer to a map within any shell script.

Also, make sure that the NIS slave server is in the map ypservers within the domain. If omitted, it will still work as a server, but `yppush` will not tell it when a new copy of a map exists.

If the problem is not obvious, you can work around it while you debug using `rcp`, `ftp`, or `tftp` to copy a recent version from any healthy NIS server. You may not be able to do this as root, but you can probably do it as daemon. For instance, to transfer map "busted," do the following:

```
chmod go+w /usr/etc/yp/mydomain
su daemon
rcp ypmaster:/etc/yp/mydomain/busted.* \
/usr/etc/yp/mydomain
CTRL-D
chown root /usr/etc/yp/mydomain/busted.*
chmod go-w /usr/etc/yp/mydomain
```

Notice that the * character has been escaped in the command line, so it will be expanded on ypmaster, instead of locally on ypslave. Also, notice that the map files should be owned by root, so you must change ownership of them after the transfer. Obviously, if you can do the `rcp` as root, it makes the whole thing easier.

ypserv crashes

When the `ypserv` process crashes almost immediately, and won't stay up even with repeated activations, the debug process is virtually identical to that described above in "ypbind crashes" page 266. Check for the `portmap` daemon:

```
ps ax | grep portmap
```

Reboot the server if you do not find the daemon. If it is there, enter

```
rpcinfo -p speed
```

and look for screen output similar to that shown in Figure 149.

Figure 148 `portmap` output

```
program     vers   proto     port
100004      2      udp       1027        ypserv
100004      2      tcp       1024        ypserv
100004      1      udp       1027        ypserv
100004      1      tcp       1024        ypserv
100007      2      tcp       1025        ypbind
100007      2      udp       1035        ypbind
100007      1      tcp       1025        ypbind
100007      1      udp       1035        ypbind
100009      1      udp       1023        yppasswdd
100003      2      udp       2049        nfs
10002       1      udp       1074        status
100024      1      tcp       1031        status
100021      1      tcp       1032        nlockmgr
100021      1      udp       1079        nlockmgr
100020      1      udp       1082        llockmgr
100020      1      tcp       1033        llockmgr
100021      2      tcp       1034        nlockmgr
100012      1      udp       1104        sprayd
100011      1      udp       1106        rquotad
100005      1      udp       1108        mountd
100008      1      udp       1110        walld
100002      1      udp       1112        rusersd
100002      2      udp       1112        rusersd
100001      1      udp       1115        rstatd
100001      2      udp       1115        rstatd
100001      3      udp       1115        rstatd
```

Look for output to the screen similar to that shown in Figure 148. On your host, the port numbers will be different. The four entries that represent the `ypserv` process are shown in Figure 149.

Figure 149 Entries representing `ypser`

```
100004      2      udp       1027        ypserv
100004      2      tcp       1024        ypserv
100004      1      udp       1027        ypserv
100004      1      tcp       1024        ypserv
```

If they are not there, `ypserv` has been unable to register its services. Reboot the machine. If they are there, and they change each time you try to restart `ypserv` reboot the machine. If the situation persists after reboot, call the CONVEX Technical Assistance Center.

Other NIS error conditions

The following items show how various NIS utilities respond when one or more daemons are killed or aborted. Network problems, host server network problems, or a missing host server `portmap` can also cause these responses.

ypserv killed or aborted

- ypserver# **ypcat passwd**
  Reason: cannot bind to a server that serves domain.
- ypserver# **ypmatch <username> passwd**
  ypmatch: Can't match key <username> in map passwd.byname.
  Reason: cannot bind to a server that serves domain.
- ypserver# **ypwhich**
  ypwhich: Domain <mydomain> not bound on <mydomain>.
- ypserver# **ypwhich**
  ypwhich: can't call ypbind on <ypclient>: RPC: Timed out
  Reason: Two ypbind processes have spawned during the `ypwhich` command, causing a time out.
- ypserver# **/usr/etc/yp/yppoll passwd**
  ** never returns **
- ypserver# **/usr/etc/yp/ypset <mydomain>**
  ** never returns **

ypbind killed or aborted

- ypclient# **ypcat passwd**
  ypcat: can't bind to yp server for domain <mydomain>.
  Reason: cannot communicate with ypbind.
- ypclient# **ypmatch <username> passwd**
  ypmatch: Can't match key username in map passwd.byname.
  Reason: cannot communicate with ypbind.
- ypclient# **ypwhich**
  ypwhich: can't call ypbind on <ypclient>: RPC: Timed out
- ypclient# **/usr/etc/yp/yppoll passwd**
  RPC: Timed out
- ypclient# **/usr/etc/yp/ypset <ypserver>**
  Sorry, I can't make use of the yellow pages. I give up.

ypbind and ypserv not running

```
ypclient# /usr/etc/yp/ypset -d <mydomain> ypserver
Sorry, I couldn't send my rpc message to ypbind on host
<ypclient>.
```

Security with NIS

Security on a system running NIS is dependent upon how NIS consults the administrative files on which its maps are based. Local files on the host are consulted first, then the system consults maps in the NIS domain. For example, the system checks the NIS/etc/aliases file in the NIS domain for mail aliases, then checks the mail.aliases NIS map.

The remaining files on which NIS maps are based are global files: /etc/hosts, /etc/networks, /etc/ethers, /etc/services, /etc/netmasks, /etc/protocols, and/etc/netgroup. The information in these files is network-wide data and accessed only from NIS. However, when booting, each host needs an entry in /etc/hosts for itself. In summary, if NIS is running, global files are only checked in the NIS maps; a file on your local host is not consulted.

Special NIS password change

When you change your password with the `passwd` command, you change the entry explicitly given in your host's local /etc/passwd file. If your password is not given explicitly, but rather pulled in from NIS with a + entry, then the `passwd` command will print the error message

```
Not in passwd file.
```

To change your password in the NIS password file, you must use the `yppasswd` command. To enable this service, you must start up the daemon `yppasswd` server on the machine serving as the master for the NIS password file.

/etc/publickey

To enable users to use the secure option to mount a directory, the NIS database publickey must exist. publickey is the public key database used for secure networking. Each entry in the database consists of a network user name (which may either refer to a user or a hostname), followed by the user's public key (in hexadecimal notation), a colon, and then the user's secret key encrypted with its login password (also in hexadecimal notation).

This file is altered either by a user through the `chkey` command or by using the `newkey` command. The file /etc/publickey should contain only data on the NIS master, where it is converted into the NIS database publickey.byname. For details, refer to the ypupdated(8c), newkey(8), chkey(1), and publickey(3r) man pages.

Two NIS databases enhance system security:

- netid— created from the passwd, host, and group files— that requires no administering by you.
- publickey— contains every user that has a public key—that requies you to use the newkey program to enter new entries.

Do not use a text editor to alter the /etc/publickey file because the file contains encryption keys. To alter the /etc/publickey file, use the newkey command. Its two options are

 # newkey -u *username*
for a regular user on a host machine.

 # newkey -h *hostname*
for a root user on a host machine.

To create the NIS database from the /etc/publickey file, enter

```
# cd /usr/etc/yp
# make publickey
```

Network- wide groups: hosts and users

NIS uses the /etc/netgroup file on the master NIS server for permission checking during remote mount, login, remote login, and remote shell. It uses /etc/netgroup to generate three NIS maps in the /usr/etc/yp/domainname directory: netgroup, netgroup.byuser and netgroup.byhost. The NIS map netgroup contains the basic information in /etc/netgroup. The two other NIS maps contain a more specific form of the information to speed up the lookup of netgroups given the host or user.

The programs that consult these NIS maps are `login`, `mountd`, `rlogin`, and `rsh`. `login` consults them for user classifications if it encounters netgroup names in /etc/passwd. `mountd` consults them for machine classifications if it encounters netgroup names in /etc/exports. `rlogin` and `rsh` consult the netgroup map for both machine and user classifications if they encounter netgroup names in the /etc/hosts.equiv or /.rhosts file.

Disabling NIS

If `ypserv` on the master is disabled, you can no longer update NIS maps.

If you choose not to use NIS, you can disable it by simply renaming the /usr/etc/ypbind to /usr/etc/ypbind.orig. (This is what the install script does automatically if you tell it you do not want to run NIS.)

```
ypclient# ps ax | grep yp
ypclient# kill pid#

ypclient# cd /etc
ypclient# mv /usr/etc/ypbind/usr/etc/ypbind.orig
```

where *pid#* is the `yp pid` found using the previous command.

To disable NIS on a particular NIS slave or master, enter

```
ypclient# mv /usr/etc/ypser/usr/etc/ypserv.orig
```

Accessing and modifying NIS maps

This section describes how to use four interface programs designed to help you access or modify data stored in NIS maps. Actually, these maps are stored as dbm databases— indexed files with fast access provided through a has table. Specifically, these programs—`ypcat`, `ypmatch`, `yppasswd`, and `ypwhich`—enable you to access selected data from various mpas, modify your NIS password file, and determine which host is acting as a database server. None of these programs demand more than a casual knowledge of NIS.

`ypclnt`, also covered in this section, is a library of functions that enables you to create your own interface to NIS. Although `ypclnt` is more complicated to use than the other programs, individuals familiar with C language and the ConvexOS operating system should have no problems with it.

Printing values stored in NIS maps: `ypcat`

`ypcat` enables you to view on standard output the values from the NIS maps stored on your host or on other hosts across the network. `ypcat`, the cat equivalent for NIS, is typically used with a map name as an argument. The command sequence

```
% ypcat hosts.byaddr
```

writes the contents of the map hosts.byaddr to standard output, which is a listing of the names and addresses of network hosts. You can also invoke a map by using a "nickname." For example:

```
% ypcat hosts
```

produces the same output as hosts.byaddr. To list the nicknames for maps in your domain, enter upcat with its `-x` argument as shown here

```
% ypcat -x
Use "passwd" for map "passwd.byname"
Use "group" for map "group.byname"
Use "networks" for map "networks.byaddr
Use "hosts" for map "hosts.byaddr"
Use "protocols" for map "protocols.bynumber"
Use "services" for map "services.byname
Use "aliases" for map "mail.aliases"
Use "ethers" for map "ethers.byname"
Use "rpc" for map "rpc.bynumber"
Use "pwrestrict" for map "pwrestrict.byname"
```

If you invoke `ypcat -x` on your host, you get a good look at the maps that are available to you.

To display maps from different domains, use the -d option. If, for example, you wanted to list networks.byname from the venus domain, you would enter

```
% ypcat -d venus networks
```

```
sun-ether          192.9.200       sunether ethernet
sun-oldether       125             sunoldether
mktg-net           105
dragon-net         102
convex-net         100
ucb-ether          46              ucbether
arpanet            10              arpa
dragon-net         102
hyper-net          101             hyper
convex-net         100             convex
arpanet            10              # the arpanet
```

Printing selected data within NIS maps: ypmatch

ypmatch, like ypcat, enables you to view data stored in databases not only on your host but on other hosts connected to the network. Unlike ypcat, ypmatch enables you to get at *selected* parts of the database by printing information that matches *keys* that you provide. You enter a key, NIS searches an NIS map for a match, then writes the data associated with the matching key on standard output.

Suppose, for example, you want to check the Internet address of the host *venus*. You could log in to *venus* remotely, or you could use ypcat to print the contents of the *hosts* database on your host. A simpler method is to use ypmatch to search the *hosts* database for values associated with the *venus* key, as in

```
% ypmatch venus hosts
100.0.5.1 venus
```

The key you supply ypmatch must exactly match the key in the map. If you supply a key for which no values can be associated, you receive an error message:

```
% ypmatch loopback networks
ypmatch: Can't match loopback.
Reason: no such key in map.
```

As with ypcat, you can use the *-x* option to display the map nickname table

```
% ypmatch -x

Use "passwd" for map "passwd.byname"
Use "group" for map "group.byname"
Use "networks" for map "networks.byaddr"
Use "hosts" for map "hosts.byname"
Use "protocols" for map "protocols.bynumber"
Use "services" for map "services.byname"
Use "aliases" for map "mail.aliases"
Use "ethers" for map "ethers.byname"
Use "rpc" for map "rpc.bynumber"
Use "pwrestrict" for map "pwrestrict.byname"
You can also use the -d flag to access data in a remote
database, just as you did with ypcat:

% ypmatch -d eclipse docstaff aliases
moe, curly, larry
```

When you specify the -k option, the key is printed (with a colon) before the value of the key. This option is useful primarily for improving the clarity of the output you receive when you specify more than one key. Here's an example:

```
% ypmatch -k moe curly larry passwd

moe:
moe:VPF75JjbUCrvw:107:51:Moe,88/487,295,3415009:/doc/
moe:/bin/csh

curly:
curly:WTqWgH2mHeFRc:172:51:Curly,89/488,296,5307495:/
doc/curly:/bin/csh

larry:
larry:VL.fEiCaWwb/w:133:51:Larry,90/49,297,5273452:/d
oc/larry:/bin/csh
```

Specifying your NIS password: yppasswd

You can use yppasswd to change or install your NIS password. As you remember from the previous discussion of maps, user passwords are usually one of the first things networked when NIS is installed. The NIS password is much like a typical host password, but it can be used to sign onto many hosts across the network. Your NIS password may be different from the one on your own host.

Adding (or changing) a NIS password is just like adding or changing your password on a CONVEX supercomputer. You enter `yppasswd` at a command prompt to invoke the program; then you enter your old password, followed by the new one. As with `passwd` on a CONVEX supercomputer, you must enter the new password twice. Here's an example of a typical `yppasswd` terminal session. (You may receive some messages not listed here, depending on how you use the command.)

```
% yppasswd

Old yp passwd: enter_old_password_here
New yp passwd: enter_new_password_here
Retype new passwd: enter_new_password_here
```

Your new password must be at least six characters long. If you are not the password "owner," you must be a superuser to change a password. In either case, you must prove you know the old password. The update protocol passes all the information to the server in one RPC call, without ever looking at it. Thus if you enter your old password incorrectly, you are not notified until after you have entered your new password.

If you try to change your password on a host that is not running the `yppasswd` daemon (`/usr/etc/rpc.yppasswdd`), you receive a message similar to the following:

```
% yppasswd

convex1 is not running yppasswd daemon
```

If you receive this message, log in to the `passwd` master server and try again.

Querying about maps and NIS services: ypwhich

`ypwhich` provides a general information about which server is supplying NIS services and which host is acting as master for a particular map.

In the simplest case, you can use `ypwhich` just as you did `ypcat` and `ypmatch` to display a list of maps and their nicknames. You do this by using the `-x` option, as follows:

```
% ypwhich -x

Use "passwd" for map "passwd.byname"
Use "group" for map "group.byname"
Use "networks" for map "networks.byaddr"
Use "hosts" for map "hosts.byaddr"
Use "protocols" for map "protocols.bynumber"
Use "services" for map "services.byname"
Use "aliases" for map "mail.aliases"
Use "ethers" for map "ethers.byname"
Use "rpc" for map "rpc.bynumber"
Use "pwrestrict" for map "pwrestrict.byname"
```

The `-m` flag enables you to find out which host is the master server for a particular map. Suppose, for example, that you want to find out which host is the master server for the RPC map you're using. Simply invoke `ypwhich` with the `-m` option and the name of the map:

```
% ypwhich -m rpc

convex1
```

In this example, *convex*1 is the master server for the RPC map.

To locate master servers for maps in other domains, use the `-d` switch in conjunction with the `-m` switch:

```
% ypwhich -d eclipse -m networks

convex2
```

The -V1 and -V2 flags allow you to determine, respectively, which server is supplying Version 1.0 and Version 2.0 of the Yellow Pages. (Version 1.0 is provided for backward compatibility with older systems from other vendors. Version 2.0 is used as the default CONVEX version of NIS.) For example:

```
% ypwhich -V1
convex1

% ypwhich -V2
convex1
```

You can print a list of every map in a domain, with its associated master server, by invoking the -m option without arguments. For example:

```
% ypwhich -m
bootparams convex1
mail.aliases convex1
pwrestrict.byuid convex1
pwrestrict.byname convex1
netgroup.byhost convex1
netgroup.byuser convex1
netgroup convex1
protocols.byname convex1
services convex1
services.byname convex1
rpc.byname convex1
rpc.bynumber convex1
networks.byaddr convex1
networks.byname convex1
ethers.byname convex1
ethers.byaddr convex1
hosts.byaddr convex1
hosts.byname convex1
group.bygid convex1
group.byname convex1
passwd.byuid convex1
protocols.bynumber convex1
ypservers convex1
passwd.byname convex1
```

To list maps in other domains, use the -d option in conjunction with the -m switch, as follows:

```
% ypwhich -d eclipse -m
```

Building interfaces to NIS: ypclnt

ypclnt is a package of library functions that you can use to build your own interfaces to the NIS. These functions are designed to be used in *programs*; they are not stand-alone programs, and do not do you much good if you are not a C programmer. This section describes the available functions and provides examples. Refer to the ypclnt(3N) man page for more information and for a look at the functions themselves.

The functions available to you include the following:

yp_bind	Binds client program to a given yp server (domain)
yp_bind	Unbinds client programs from given yp domain

`yp_bind`	Returns default system domain name
`yp_bind`	Matches a key in a given yp map
`yp_bind`	Returns first entry in a yp map
`yp_bind`	Returns next entry in a yp map
`yp_bind`	Transfers an entire map in one TCP request
`yp_bind`	Returns map order number (a date stamp in seconds)
`yp_bind`	Returns host name of master serve for a given map
`yp_bind`	Returns string describing a given error
`yp_bind`	Converts NIS protocol error to `ypclnt` error code

Figure 150 through Figure 153 llustrate the use of `ypclnt` functions.

Figure 150 Using `ypclnt`, example 1

```
1       #include <stdio.h>
2       #include <rpcsvc/ypclnt.h>
3
4       #define MAP "passwd.byname"
5
6       main()
7       {
8       char *domain;
9       int err;
10      char *key, *old_key, *val;
11      int keylen, old_keylen, vallen;
12
13      if ((err = yp_get_default_domain(&domain)) != 0) {
14      fprintf(stderr, "demo: %s\n", yperr_string(err));
15      exit(1);
16      }
17      if ((err = yp_bind(domain)) != 0) {
18      fprintf(stderr, "demo: %s\n", yperr_string(err));
19      exit(1);
20      }
21      if ((err = yp_first(domain, MAP, &old_key, &old_keylen,
22          &val, &vallen)) != 0) {
23      fprintf(stderr, "demo: %s\n", yperr_string(err));
24      exit(1);
25      }
26      printf("%s", val);
27      while (yp_next(domain, MAP, old_key, old_keylen, &key, &keylen,
28          &val, &vallen) == 0) {
29      printf("%s", val);
30      old_key = key;
31      old_keylen = keylen;
32      }
33      exit(0);
34      }
```

Figure 150 shows the use of `ypclnt` functions `yp_get_default_domain` (at line 13); `yperr_string` (at line 14); `yp_bind` (at line 17); `yp_first` (at line 21); and `yp_next` (at line 27). The program finds the given domain, binds to it, and then reads and prints the entries in the file MAP. In function, this program is essentially a smaller version of `ypcat`.

Figure 151 Using `ypclnt`, example 2

```
1       #include <stdio.h>
2       #include <rpc/rpc.h>
3       #include <rpcsvc/ypclnt.h>
4       #include <rpcsvc/yp_prot.h>
5
6       #define MAP "passwd.byname"
8       extern int map_foreach();
9
10      main()
11      {
12      char *domain;
13      int err;
14      char *key, *old_key, *val;
15      int keylen, old_keylen, vallen;
16      struct ypall_callback callback;
17
18      if ((err = yp_get_default_domain(&domain)) != 0) {
19      fprintf(stderr, "demo: %s\n", yperr_string(err));
20      exit(1);
21      }
22      if ((err = yp_bind(domain)) != 0) {
23      fprintf(stderr, "demo: %s\n", yperr_string(err));
24      exit(1);
25      }
26      callback.foreach = map_foreach;
27      if ((err = yp_all(domain, MAP, &callback)) != 0) {
28      fprintf(stderr, "demo: %s\n", yperr_string(err));
29      exit(1);
30      }
31      exit(0);
32      }
34      map_foreach(instatus, inkey, inkeylen, inval, invallen, indata)
35      int instatus;
36      char *inkey;
37      int inkeylen;
38      char *inval;
39      int invallen;
40      char *indata;
41      {
42      if (instatus != YP_TRUE) {
43      return(1);
44      }
45      inkey[inkeylen] = inval[invallen] = '\0';
46      printf("%s\n", inval);
47      return(0);
48      }
```

Figure 151 is a rewritten version of Figure 150, in which `yp_all`, rather than `yp_first` and `yp_next`, is used. `yp_all` typically works much faster than the `yp_first/yp_next` combination, especially when it is likely that you need to read an entire map. When the code in Figure 150 and Figure 151 was run across a sample *passwd* file, the code in Figure 151 ran 500% faster. (Specifics: passwd file contained 182 entries and 14135 characters; time to read/print with code in Figure 150 = 10 seconds, while code in Figure 151 read and printed the file in 2 seconds.)

In Figure 151, the `inkey` and `inval` entries are not null-terminated. You are given the lengths of each string as `inkeylen` and `invallen`—you should null-terminate these strings, as shown at line 45.

Figure 152 Using `ypclnt`, example 3

```
1       #include <sys/types.h>
2       #include <sys/time.h>
3       #include <stdio.h>
4       #include <rpcsvc/ypclnt.h>
5
6       #define MAP "passwd.byname"
7       #define KEY "daemon"
8
9       main()
10      {
11      char *domain, *master;
12      int err;
13      char *key, *val;
14      int keylen, vallen;
15      int order;
16
17      if ((err = yp_get_default_domain(&domain)) != 0) {
18      fprintf(stderr, "demo: %s\n", yperr_string(err));
19      exit(1);
20      }
21      if ((err = yp_bind(domain)) != 0) {
22      fprintf(stderr, "demo: %s\n", yperr_string(err));
23      exit(1);
24      }
25      if ((err = yp_match(domain, MAP, KEY, strlen(KEY), &val, \ &vallen))
!= 0) {
26      fprintf(stderr, "demo: %s\n", yperr_string(err));
27      exit(1);
28      }
29      printf("%s", val);
30      if ((err = yp_order(domain, MAP, &order)) != 0) {
31      fprintf(stderr, "demo: %s\n", yperr_string(err));
32      exit(1);
33      }
34      printf("Map %s has order number %d -- %s", MAP, order, \
ctime(&order));
35      if ((err = yp_master(domain, MAP, &master)) != 0) {
36      fprintf(stderr, "demo: %s\n", yperr_string(err));
37      exit(1);
38      }
39      printf("The master server for map %s is %s\n", MAP, master);
40      exit(0);
41      }
42
43
44
```

Figure 152 shows the use of `yp_match` (at line 25); `yp_order` (at line 30); and `yp_master` (at line 35). The program binds to a specified domain, and then lists values in MAP that match the KEY argument. `yp_match` is used, of course, to search MAP for a particular KEY. `yp_order` and `yp_master` are typically not used for application programs. They return the time that a map was built and the name of the host that is the master server for MAP, respectively.

Figure 153 shows the output for Figure 152.

Figure 153 Output from `ypclnt`

```
daemon:6c2Lz36Q10Mc.:1:1:Our friend, the daemon:/:/bin/csh
Map passwd.byname has order number 532161701 -- Wed Nov 12 00:41:47 1986
The master server for map passwd.byname is convex1
```

Securing access to networked machines

24

The authentication system used by CONVEX NFS enhances the security of network environments, while being general enough to be used by a variety of operating systems. Because the underlying mechanism upon which secure NFS operates is Secure RPC, this chapter first focuses on RPC services before it explains secure NFS.

Note

Secure NFS relies on facilities provided by the Yellow Pages (YP). Therefore, you must run YP if you wish to use Secure NFS.

Secure NFS relies on facilities provided by the Network Information System(NIS). Therefore, NIS must be running if you want to use secure NFS.

The topics explained in this chapter are

- Introducing the RPC network security services
- Using Unix-like authentication
- Using DES encryption
- Using public key encryption
- Using secure NFS

Tightening NFS security with RPC

Two major factors in NFS give opening for unauthorized access to data:

- An NFS server authenticates a file request by authenticating the machine from which the request originates, but not the user making the request. No verifier exists.
- Anyone running `su` can impersonate the rightful owner of a file. Usually, a user logged in as root on a client has the same access right to the remote file system as the user nobody, which is most likely the same as the general public.

A common solution to network security problems is to leave the solution to each application. Secure NFS puts authentication at the RPC level. The result is a standard authentication system that covers all RPC-based applications, namely NFS and the NIS. This system allows authentication of users as well as machines, and makes a network environment act like a time-sharing environment.

Authentication is the cycle during which authorization information is generated and tested. The major elements in authorization are defined in the next section.

Introducing authorization

Each request sent from the client contains information that identifies the user that originated the request. This information is called the *authorization*. Authorization information consists of two pieces of data: the *credential* and the *verifier*. The credential is something that uniquely identifies a user like a user name or Unix user ID. The verifier is something that the server can use to ensure that the real user (person) originated the request containing that credential.

To summarize, the two major elements in authorization are

credential consists of a *network name,* which is composed of the Unix user ID and the *domainname* of the client machine. User IDs are unique within a domain.

verifier consists of the current time stamp encrypted using a DES key that only the server and the client machines know.

Neither the RPC protocol nor NFS verify that the user sending a UID is who they say they are—no validation of the credential. The system that does have credential verification is Secure RPC. By combining Secure RPC with NFS, secure NFS emerged with the capability to perform credential checking with validation.

Introducing RPC services

RPC is at the core of the NFS network security system. To understand secure NFS, first you need to understand how RPC works authentication. Explained in this chapter are the RPC-based authentication services, starting at the lowest level of security supported up through secure NFS. These services are:

- Unix-like authentication— security built into common Unix and internet services.
- Secure RPC— combination of two types of crytography:
  - DES encryption—encrypts the current time
  - Public key encryption—client and server calculate the conversation key, which is used to encrypt and decrypt the time stamp on the client and the server.
- Secure NFS—export and mount file systems with the secure option to utilize the predefined keys for encryption and decryption calculations.

Unix-like authentication

Before the introduction of secure NFS, most CONVEX network services used a Unix-style authentication. This type of authentication system has two problems:

- No verifier— potential for falsified use of credential, using hostname.
- Superuser (su) access— potential for misuse of root access.

The verification of credentials occurs only at mount time when the client gets its key—the file handle—for all further requests from the server. Someone could break security by obtaining a file handle without contacting the server, perhaps by tapping into the net or by guessing. After an NFS file system has been mounted, there is no checking of credentials during file requests; no verifier exists.

If a file system has been mounted from a server that serves multiple clients (as is typically the case), there is no protection against someone who has root permission on their machine using su (or some other means of changing UID) gaining unauthorized access to other people's files. Another part of this problem is the severe method NFS uses to circumvent the problem of not being able to authenticate remote client superusers: denying them superuser access altogether. DES authentication, explained in the next section, addresses both shortcomings.

Every NFS request contains the client's machine-name, user's user UID and GID, and list of groups to which that user belongs. The name of a network entity is basically the UID, one assigned per NIS naming domain that typically spans several machines. When domains are linked together, there is the possibility of UID clashes. Also, the superuser (UID 0) should not be assigned on a per-domain basis, but rather on a per-machine basis. By default, NFS refuses UID 0 root access across the network.The NFS server takes on the credential of the requesting user when it attempts to access the requested file or file system. Clients of non-Unix machines somehow must generate a Unix credential, often by adopting the default user ID nobody. Still, nothing in RPC protocol verifies that the user is really who the UID claims.

NFS combats deficiencies in this authentication by checking the source Internet address of `mount` requests as a verifier of the hostname field and accepting requests only from privileged Internet ports. Still, it is not difficult to circumvent these measures, and NFS really has no way to verify the user ID. Secure NFS uses the secure RPC mechanism to pass the real user ID of the requester, not the effective user ID. It also, through authentication, assures the server that the user ID being passed is coming from its true owner.

It is unrealistic to assume that all machines on a network are Unix-based machines. NFS works with MS-DOS and VMS machines, but Unix authentication breaks down when applied to them.

Understanding DES authentication

DES requires additional overhead for the encryption of the time stamp—a 64-bit quantity, which also happens to be the DES block size. Four encryption operations take place in an average RPC transaction:

- Client encrypts the requested time stamp;
- Server decrypts it;
- Server encrypts the reply time stamp; then
- Client decrypts it.

Encrypting the current time

The security of DES authentication is based on a sender's ability to encrypt the current time, which the receiver can then decrypt and check against its own clock. The time stamp is encrypted with DES. Two facts must be in agreement for this scheme to work:

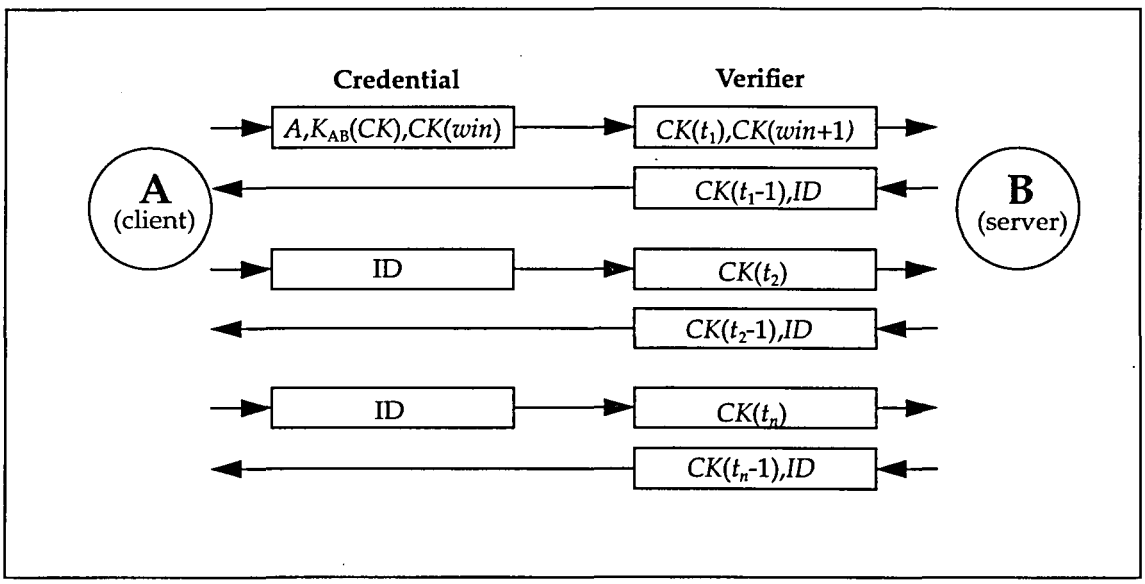
- Agreement on the current time by both client and server.
- Use of same encryption key by client and server.

If a network has time synchronization (`timed`, for example), then client/server time synchronization is performed automatically. However, if this is not available, time stamps can be computed using the server's time instead of network time. To do this, the client asks the server what time it is before starting the RPC session, then computes the time difference between its own clock and the server's. This difference is used to offset the client's clock when computing time stamps. If the client and server clocks get out of sync to the point where the server begins rejecting the client's requests, the DES authentication system just resynchronizes with the server.

Generating the encryption key

Here is how the client and server arrive at the same encryption key. When a client wishes to talk to a server, it generates a random key referred to as the *conversation key* (CK), to be used for encrypting the time stamps (among other things). The client encrypts the conversation key using a public key scheme and sends it to the server in its first transaction. This key is the only thing that is ever encrypted with public key cryptography. The particular scheme used is described further in the "Understanding public key encryption" page 302. For any two agents A and B, there is a DES key K_{AB} that only A and B can deduce. This key is known as the common key, K_{AB}.Figure 154 illustrates the authentication protocol in more detail, describing client A talking to server B.

Figure 154 DES authentication protocol



A term of the form $K(x)$ means x encrypted with the DES key K. Looking at Figure 154, you can see that, for its first request, the client's credential contains three items: its name A, the conversation key CK encrypted with the common key K_{AB}, and a thing called *win* (window) encrypted with CK.

What the window says to the server, in effect, is

> I will be sending you many credentials in the future, but there may be imposters sending them too, trying to impersonate me with bogus time stamps. When you receive a time stamp, check to see if your current time is somewhere

between the time stamp and the time stamp plus the window. If it is not, reject the credential.

For secure NFS file systems, the window currently defaults to 30 minutes. The client's verifier in the first request contains the encrypted time stamp and an encrypted verifier of the specified window, *win + 1*.

Someone could attempt to impersonate A by writing a program that, instead of filling in the encrypted fields of the credential and verifier, stuffing in random bits. The server decrypts CK into some random DES key and uses it to decrypt the window and the time stamp. These just end up as random numbers. After a few thousand trials, there is a good chance that the random window/time stamp pair would pass the authentication system. The window verifier makes guessing the right credential much more difficult.

After authenticating the client, the server stores four items in a credential table:

- Client's name (A)
- Conversation key *(CK)*
- Window
- Time stamp

The server stores the first three items for future use. It stores the time stamp to protect against replays. The server only accepts time stamps that are chronologically greater than the last one seen; replayed transactions are guaranteed to be rejected.

The server sends its verifier back to the client; the verifier contains an index ID into the server's credential table, plus the client's time stamp minus one, encrypted by the conversation key, *CK*. The client knows that only the server could have sent such a verifier, because only the server knows the time stamp the client sent and the conversation key being used. The reason for subtracting one from it is to ensure that it is invalid and cannot be reused as a client verifier.

The first transaction is rather complicated; transactions thereafter are much simpler. The client just sends its ID and an encrypted time stamp to the server, and the server sends back the client's time stamp minus one, encrypted by *CK*.

Why the DES key database files

DES authentication revolves around the key database. To successfully access files on an NFS file system mounted with the `-secure` option, each user must have a pair of keys in the key database of the server machine. Keys are 192-bit binary numbers.

This database consists of three files that should be backed up along with /etc/passwd and other critical files; these files are

- /etc/publickey
  contains keys for each user and machine on the network (also available as the NIS publickey.bynameNIS). The file is a set of ASCII records with these 48-byte fields:
  netname publickey:secretkey

  The *secretkey* stored here is encrypted with the user's password to ensure it stays secret. The superuser can place entries in this file and update the NIS map by using `newkey`, while users may change their existing keys with `chkey`.

- /etc/keystore
  contains a list of the users on the local machine who are trusted to use encrypted RPC. Managed exclusively by the keyserver daemon, its binary records are in this form:
  uid (2 bytes) secretkey (24 bytes == 192 bits)

  Each secret key stored here is encrypted with the secretkey for the local superuser to keep them secret. Each user needing to use Encrypted RPC must have an entry in this file; the entry is made automatically whenever a user logs in and types a password. Any user who logged in without typing a password does not have an entry and cannot use secure NFS. A way around this is to run `keylogin` and type the correct password.

- /etc/.rootkey
  contains the secretkey for root, unencrypted. It is also managed exclusively by the `keyserver` daemon. Permissions on this file must disallow access to any user other than root. It is simply a 48-byte ASCII file terminated with a newline, similar to this line:
  `54918c47567b569644517cad61033675898519f4b1d7c575`

  The root key is written out in this way so a machine reboots after a power failure without a superuser having to be present to type a password. If this was not done, secure NFS mounts would be inaccessible to everyone until the superuser logged in. This file is not written by the

keyserver until root logs in properly. Secure NFS also fails for all users and systems if root has not logged in properly.

Entering keys into the key database

Getting keys in the key database can be a logistical problem. As system manager you can run newkey for each user account for which you know the password, but this is not practical for existing installations. Root keys for other machines can also be a problem, as root passwords should be very well-guarded. Fortunately, there are ways to work around this.

Making key entries within a domain

The newkey and chkey programs communicate with the NIS update daemon to make sure that both /etc/publickey on the master server and the NIS publickey.byname map are updated correctly. In a given NIS domain, both the superuser and users can enter new keys and update existing keys.

```
client% chkey
Generating new key for unix.1955@belmont.east.sun.com
```

or as superuser, you can add a new user key

```
nis_sysmaster# newkey -u holden
Adding new key for unix.1492@belmont.east.sun.com
```

Only superuser can create new keys without the nobody key entry in the NIS publickey map. When the nobody entry exists in publickey, users can create their own keys using chkey, as in

```
client% chkey

Generating new key for unix.1492@belmont.east.sum.com
```

Superusers of other hosts in the domain may use newkey to set their machine up for secure NFS at any time, while users may be denied the ability to make their first keys, they may change a set of existing keys at any time.

```
# newkey -h gilbert
```

Adding new key for unix.gilbert@belmont.east.sun.com

Changes made to maps at the master server propagate normally by yppush, yppoll, and ypxfr.

To avoid running newkey or passwd for every user on an existing system, you can permit users to enter their own new keys with chkey by starting up the NIS update daemon, rpc.ypupdated, with a -i flag. chkey normally tries to use the Encrypted RPC protocol itself, which does not work without a proper set of keys in place; it will try to use a normal RPC if that

does not work. The `-i` flag on rpc.ypupdated allows it to accept normal RPC, which means that users without keys can get a new pair. This mode of operation is less secure, so only use the `-i` option for a week or so and add something to /etc/motd asking your users to run `chkey` once at some time within that week.

Making key entries across domains

Communicating with hosts in another NIS domains is less automatic; you must be using the same key pair as the other domain. The simplest way to accomplish this is outlined in the following steps for both domains:

Step 1 Have the other machine's system manager cut out the /etc/publickey entries for root and all users who need to use your machine and send these entries to you via electronic mail. Use an editor to add these entries to the /etc/publickey file on your machine.

Step 2 To define a key valid in the current NIS domain, for each user enter

users from the other machine, add this line of information to /etc/inetd:

unix.uid@NIS domain uid: gid, gid

where

os_type	Type of operating system issuing request.
.uid	Identifier of user
@domain	Access to this system.
UID:	Access to files owned by this user ID.
GID	Access to files owned by this group ID.

This example entry

```
unix.3900@snoopy 7100:10,33
```

allows the person with user ID3900 on another system access to files owned by user ID 7100 and to files owned by groups 10 and 33 on your machine in domain snoopy. You may give the user any user ID and group list you like, as long as you list at least one group after the colon.

Step 3 After you finish editing /etc/publickey and /etc/netid, to update the NIS database, run `make` in /usr/etc/yp.

You must perform this procedure any time a key used in both domains changes.

Employing DES authentication in applications

A generalized NIS update service is one application of DES authentication. This service allows users to update private fields in NIS databases. The NIS maps; hosts, ethers, bootparams, and publickey, employ the DES-based update service.

Guessing file handles is no longer a problem, because, in order to gain unauthorized access, would-be trespassers must guess the correct encrypted time stamp to place in the credential, which is virtually impossible. The problem of authenticating root users is addressed by authenticating machines.

The level of security associated with each file system may be altered by a system manager. The file /etc/exports contains a list of file systems and the machines that may mount them. By default, file systems are exported with Unix authentication, but the manager can have them exported with DES authentication by specifying `-secure` on any line in the /etc/exports file. Associated with DES authentication is a parameter: the maximum window size that the server is willing to accept.

Remaining security issues in DES

There are ways to break DES authentication, but most of them are very difficult or can be prevented by a few precautionary measures. Imposters could break the DES key itself, or compute the logarithm of the public key; however, either of these options would require months of compute time on a supercomputer.

Probably the easiest way to circumvent security is to guess someone's password; unfortunately, many people choose easily guessed passwords. There is no way for software to protect against guessing; it's up to each user to choose a secure password.

The next easiest attack is to attempt replays. For example, suppose someone records all your NFS transactions with a particular server. As long as the server remains up, there is no point in replaying those transactions since the server always demands time stamps that are greater than the previous ones seen. But suppose that same someone now pulls the plug on your server, causing it to crash. As it reboots, its credential table will be clean, so it will have no track of previously seen time stamps and will accept replayed transactions.

For starters, keep your servers in a secure place so that no one can pull the plug on them. However, all servers occasionally crash without any help. To protect against replayed transactions in this case, specify a window size that is smaller than the time it takes a server to reboot (5 to 10 minutes). The server will reject any replayed transactions because they will have expired.

One security issue DES authentication does not address is tapping of the net. Even with DES authentication in place, there is no protection against someone watching net traffic. (Though trying to make sense of all the bits flying over the net is not a trivial task.) You do not want someone reading your password from the net, so logins pose a bit of a problem. Key exchange is a side effect of the authentication system, but the network tapping problem can be tackled on a per-application basis.

Anyone attempting to break DES authentication cannot use `su` to do so. In order to be authenticated, your secret key must be stored by your workstation. This usually occurs when you login; the login program decrypts your secret key with your login password and stores it away for you.

Understanding public key encryption

Public key encryption addresses the problem of getting a secret key on both the client and server hosts by letting each side choose its own secret key, then applying a mechanism by which combinations of secret keys can be validated.

The particular scheme of public key encryption used is the Diffie-Hellman method. This method is based on the fact that the public and private keys are the mathematical inverse of each other. This algorithm generates a secret key SK_A at random and computes a public key PK_A using the following formula:

$$PK_A = \alpha^{\{SK_A\}}$$

where *PK* and *SK* are 192 bit numbers and α is a well-known constant. Public key PK_A is stored in a public directory, but secret key SK_A is kept private. Next, PK_B is generated from SK_B in the same manner as above. Now common key K_{AB} can be derived as follows:

$$K_{AB} = PK_B^{\{SK_A\}} =$$
$$(\alpha^{\{SK_B\}})^{\{SK_A\}} =$$
$$\alpha^{\{SK_A SK_B\}}$$

Without knowing the client's secret key, the server can calculate the same common key K_{AB} in a different way, as follows:

$$K_{AB} = PK_A^{\{SK_B\}} =$$
$$(\alpha^{\{SK_A\}})^{SK_B} =$$
$$\alpha^{\{SK_A SK_B\}}$$

No one but the server and client can calculate K_{AB}, because doing so requires knowing either of the secret keys. This arithmetic is actually computed modulo *M*, which is another well-known constant. It would seem at first that somebody could guess your secret key by taking the logarithm of your public key, but *M* is so large that this is an infeasible computational task. To be secure, K_{AB} has too many bits to be used as a DES key, so 56 bits are extracted from it to form the DES key.

Both the public and the secret keys are stored indexed by `netid` in the NIS map publickey.byname. The secret key is DES-encrypted with your login password. When you log in to a machine, the `login` program grabs your encrypted secret key, decrypts it with your login password, and gives it to a secure

local keyserver to save for use in future RPC transactions. Users do not have to be aware of their public and secret keys. In addition to changing your login password, the `yppasswd` program randomly generates a new public/secret key pair as well.

The keyserver, `keyserv`, is an RPC service local to each machine that performs all three public key operations:

- `setsecretkey(`*secretkey*`)`
- `encryptsessionkey(`*servername, des_key*`)`
- `decryptsessionkey(`*clientname, des_key*`)`

`setsecretkey` tells the `keyserver` to store away your secret key SK_A for future use; it is normally called by `login`. The client program calls `encryptsessionkey` to generate the encrypted conversation key that is passed in the first RPC transaction to a server.

The keyserver looks up the *servername* public key and combines it with the client's secret key (set up by a previous `setsecretkey` call) to generate the key that encrypts *des_key*. The server asks the keyserver to decrypt the conversation key by calling `decryptsessionkey`.

Implicit in these procedures is the name of the caller, who must be authenticated in some manner. The keyserver cannot use DES authentication to do this, since it would create deadlock. The keyserver solves this problem by storing the secret keys by UID and only granting requests to local root processes.

The client process then executes `keyenvoy`, a `setuid` process owned by root, that makes the request on the part of the client, telling `keyserver` the real UID of the client. Ideally, the three operations described above would be system calls, and the kernel would talk to the `keyserver` directly, instead of executing the `setuid` program.

Calculating the common key

Public key systems are known to be slow, but secure NFS does not perform much actual public key encryption. Public key encryption only occurs in the first transaction with a service, and even then, caching speeds things up considerably. The first time a client program contacts a server, both it and the server must calculate the common key. The time it takes to compute the

common key is basically the time it takes to compute an exponential modulo M. You have to wait only the first time you contact a machine. Since the keyserver caches the results of previous computations, it does not have to recompute the exponential every time.

Restoring secret keys after a reboot

Consider the problem of a machine rebooting, for example after a power failure at some strange hour when nobody is around. All the secret keys that were stored get wiped out, and now no process will be able to access secure network services. The important processes at this time are usually root processes, so things would work if root's secret key were stored away; however, nobody is around to type the password that decrypts the key.

The solution is to store root's decrypted secret key in a file readable `keyserver`. This works well for machines that can store the secret key on a physically secure local disk; this is not secure for diskless machines, whose secret key must be stored across the network. If you tap the net when a diskless machine is booting, you will find the decrypted key, although such tapping is not easy.

Another booting problem is the single-user boot. In the single-user booting mode, a root login shell appears on the console. The problem here is that single-user mode does not require a password.

Another problem arises from the fact that diskless machine booting is not totally secure. It is possible for someone to impersonate the boot-server and boot a devious kernel that, for example, makes a record of your secret key on a remote machine. The present system is set up to provide protection only after the kernel and the keyserver are running. Before that, there is no way to authenticate the replies given by the boot server. Use of this method is unlikely, because anyone creating such a kernel must have access to kernel source code. Also, the crime is not without evidence. If you polled the net for boot servers, you would discover the illegal boot server's location.

`setuid` programs not owned by root may pose some problems. For example, if a `setuid` program is owned by dave, who has not logged into the machine since it booted, then the program will not be able to access any secure network services as dave. Since root's secret key is always stored at boot time, root owned programs behave as they always have.

Using secure NFS

The secure Network File System (NFS) is an application of DES authentication. To successfully access files on an NFS file system mounted with the `-secure` option, each user must have a pair of keys in the key database of the server machine. Users and remote hosts are identified by a *netname* unique for that user across the entire network; a netname looks like:

`unix.1927@convex1.convex.com`	for users
`unix.p4@convex1.convex.com`	for remote hosts

A netname is a string of printable characters; it is these netnames that are authenticated. The public and secret keys are stored on a per-netname, rather than per-username, basis. The NIS map `netid.byname` maps the netname into a local UID and group-access-list; some environments may map the netname into something else.

Netnames are globally unique. This is far easier then choosing globally unique UIDs. In the CONVEX environment, user names are unique within each NIS domain. Netnames are assigned by concatenating the operating system type and user ID with the NIS and ARPA domain names. For example, a ConvexOS system user with a user ID of 508 in the domain eng.convex.COM would be assigned the netname unix.508@eng.convex.COM. A good convention for naming domains is to append the ARPA domain name (COM, EDU, GOV, MIL) to the local domain name. Thus, the NIS domain eng within the ARPA domain convex.COM becomes eng.convex.COM

Netnames are assigned to machines as well as to users. A machine's netname is formed much like a user's. For example, a UNIX machine named hal in the domain eng.convex.COM has the netname unix.hal@eng.convex.COM. Proper authentication of machines is very important for diskless machines that need full access to their home directories over the net.

Other environments have other ways of generating netnames, but this does not preclude them from accessing the secure network services of the CONVEX environment. To authenticate users from any remote domain, you must make entries for them in two NIS databases. One entry is for their public and secret keys, the other is for their local UID and group-access-list mapping. Once these entries exist, users in the remote domain will be able to access all of the local network services, such as the NFS and remote logins.

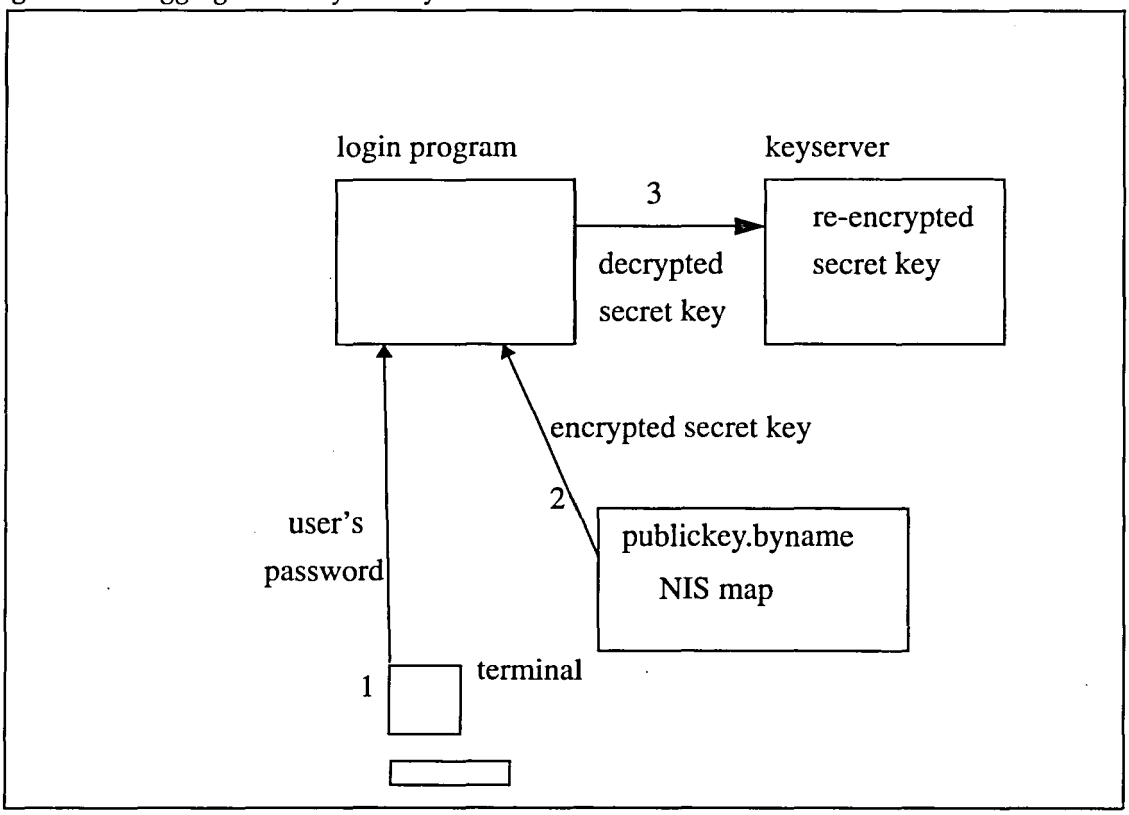
Illustrating the stages of secure NFS

To review, each user on the network has two DES keys, one public and one secret. Both keys are stored in a network-wide database called an NIS map. The name of this map is 'publickey.byname'. The secret key is stored in the map encrypted with the user's password.

Step 1 **User logs on at a client machine.**

When a user logs on to a system, the login program obtains the user's secret key from the NIS map and, using the user's typed password, decrypts the key. The key is then passed to a background process or daemon called the `keyserver`. The keyserver reencrypts the secret key using the secret key for 'root' on the system and stores the reencrypted secret key in a table for the user. The keyserver can decrypt the user's secret key at anytime. The decryption routine used by the login program is statically linked-in to the executable module and is not available to any other software on the system.

Figure 155 Logging on to a system by an NFS client



Step 2 **Remote file system is mounted to use secure NFS.**

Before a remote file system can be accessed by the user, it must be mounted by a system administrator. If the remote file system is to be accessed using secure NFS, the administrator supplies the
`-o secure` option on the mount command line. The mount command would read similar to

```
mount -o secure remotename:/filesys /mountpoint
```

Now the `mount` command performs the mount system call which creates a record of the mount in the kernel. This record contains information that indicates the name of the remote server, its address, and that accesses to the remote file system must be done using secure NFS authentication.

Step 3 **User accesses a remote file for the first time.**

For the first time since logging in and mounting the remote file system with secure, the user performs an operation which accesses a file on the remote server system. At this point, the client system must build a DES key to be used by the client and server for encrypting and decrypting the time stamps in the verifiers. This key is called the *conversation key.* The client must pass this key over the network to the server. Of course, the key cannot be simply sent over the network because it must be known to only the client and server. The conversation key is encrypted with another key called a *common key.* The common key can be derived by both the client and the server using a combination of the public and secret keys from the NIS map publickey.byname.

Choosing the conversation key

The user at the client system makes a request for a file on the remote system by issuing a system call for a file on the remote system. This transfers control to the kernel. The kernel detects that the file is a remote file and passes the request to the NFS system. The NFS system detects that the remote file system is secure and must first build an authorization record for the user that will be used on this and subsequent accesses to the remote file system. The first step in building the authorization record is to choose a conversation key. The NFS software in the kernel makes a request to the keyserver to choose the conversation key. This is done by NFS making a loopback remote procedure call. That is, a call to a remote procedure that is actually in the local machine. This is also known as a kernel call-out.

Encrypting the conversation key

Now that the conversation key has been chosen, it must be encrypted so that it can be sent over the network to the server. This is done using another key called the common key. On the client, the common key is derived using the secret key for the user and the public key for the remote system. On the server, the common key is derived using the secret key for the server and the public key for the user. All public keys are derived from their corresponding secret keys, so that the common keys calculated on the client and server will be the same.

In order to encrypt the conversation key, the NFS system makes another RPC cal lout to the keyserver. This time it passes the userID of the user with the conversation key. The keyserver encrypts the conversation key using the common key as described above. Remember, the keyserver knows the secret key for a user from when the user logged-in. The encrypted conversation key is then returned to the NFS system in the kernel.

Note that the encryption routine used by the `keyserver` is statically linked-in and is not available to any other software on the system.

Building the first credential and verifier

The client has the conversation key in encrypted and decrypted forms. It must now build the first credential and verifier that will be sent to the server along with the first NFS request. The first credential contains the user's network name, the encrypted conversation key, and a value called the *window*, that tells the server how much tolerance to allow for time stamp discrepancies; that is, how close the client's verifier time stamp must be to the server's time. The window value sent by the client is encrypted with the conversation key. The first verifier contains a time stamp encrypted with the conversation key and the window value plus one that is again encrypted with the conversation key. In order to do the encryption this time, the NFS system directly calls the DES encryption routine in the kernel. This copy of the encryption routine is accessible only to system software in the kernel and cannot be called through any system call.

After the credential is built, the client sends the first NFS request to the server with the credential and verifier.

Step 4 **Server receives the first request for the user.**

When the server receives the first request for the specified user, it knows it has not seen a request from the user before because of the user's network name contained in the credential. This indicates to the server that it must decrypt the conversation key contained in the credential and build an authorization record for the user. In order to decrypt the conversation key, the NFS server code makes an RPC callout to the keyserver running on the server system. NFS passes to the keyserver the network name of the user and the encrypted conversation key. The keyserver uses the public key for the user and the secret key for the server system to derive the common key. As discussed earlier, this common key will be equal to the one used by the client in encrypting the conversation key. The keyserver uses the

common key to decrypt the conversation key and passes the decrypted conversation key back to the NFS system in the kernel. Again, the decryption routine used by the keyserver is statically linked-in and not available to any other software on the system.

Step 5 **Server decrypts information in the initial request.**

After the NFS system in the server has the decrypted conversation key, it can use it to decrypt the remaining information in the initial request. The remaining information consists of a timestamp, the session's window value and the window value + 1. To do this, it passes the conversation key and the data to be decrypted to a DES decryption routine in the kernel. As before, this decryption routine is only available to system software in the kernel and cannot be called through any system call.

After the server decrypts the timestamp, it compares the value to its current time and using the window value, determines if the timestamp is valid. If the credential is accepted, the server processes the NFS file system request.

Step 6 **Server sends response to client.**

After the server has processed the client's NFS request, it sends a response containing the status of the request and any requested data. The server also returns an encrypted verifier so the client will be able to tell that the NFS response is from the real server. The verifier sent by the server consists of the timestamp from the client's request minus one and then encrypted using the conversation key. The server's response also contains an ID or 'nickname' which the client can use in subsequent requests instead of sending the user's network name. The nickname is an index into the server's list of user authorization records.

When the client sends another request for the user, the server uses the nickname to look up the user's authorization record and obtain the conversation key.

Note that the DES encryption routine is only available to the system software in the kernel and is not accessible through any system call

Configuring secure NFS

To get secure NFS up and running, set up the key database and start the new daemons as outlined in the following steps:

Step 1 Ensure that 'root' has a password.

Step 2 Verify that /etc/inetd.conf contains these lines:

```
time    stream  tcp     nowait  root    internal
time    dgram   udp     nowait  root    internal
```

Step 3 Verify that /etc/services contains the following lines:

```
time            37/tcp          timserver
time            37/udp          timserver
```

Step 4 Edit /etc/rc.local so that the domainname is set to a valid domain and that the new Secure daemons, `rpc.ypupdated` and `keyserv`, are started up after `ypserv` and `ypbind`.

Step 5 Use `touch` to create empty /etc/netid and /etc/publickey files.

Step 6 Configure the key database for all local and remote users who will need access to Secure file systems and for all machines with which you will be sharing files. Enter a root key using

`newkey -h` *local_hostname*

Enter the root password at the prompt. To add more hosts, repeat the `newkey` command with the appropriate hostname and password. If all your hosts are on the same subnet, it may be easier to wait until the NIS update daemon is running and let the system managers of the individual machines run `newkey` themselves, since the keys will propagate.

To add local users' keys, use

`newkey -u` *username*

Step 7 Change your directory to /usr/etc/yp to set up NIS set up. If you are not currently running NIS, run `ypinit` and answer the prompts; if you are currently running NIS, update the maps with `make`.

Step 8 Add the secure option to any /etc/fstab entries you want to mount with Secure NFS, and add the secure option to any /etc/exports entries you want to force client machines to mount with Secure NFS. Note that you must currently have at least one root-access entry on a secure export; use `root=`*localhostname* if there is no other host that should have root access.

Step 9 Start the new `rpc.ypupdated` and `keyserv` daemons; this may be done by hand or by bringing the machine up from single-user mode. If you do start the daemons by hand, you may want to unmount and remount any file systems that should now be using Secure NFS.

At this point, Secure mounts should work. If the mount attempt succeeds but the key database is incorrect, any accesses of the Secure file system will fail, even if performed by root, unless anon=0 is set in the exports entry.

All users (except root) must now use `yppasswd` instead of `passwd` to keep their secret key synchronized with their login password (out of necessity, `passwd` reencrypts the secret key for root). As a consequence, entries for individual users in local /etc/passwd files. Furthermore, it might be a good idea to overwrite passwd with yppasswd, after saving passwd as opasswd.

Step 10 When you reinstall, move, or upgrade a machine, save /etc/keystore, /etc/publickey and /etc/.rootkey along with everything else you normally save.

If you login, rlogin or telnet to another machine, are asked for your password, and type it correctly, you've given away access to your own account. This is because your secret key is now stored in /etc/keystore on that remote machine. This is only a concern if you don't trust the remote machine. If this is the case, don't ever log in to a remote machine if it asks for your password. Instead, use NFS to remote mount the files you're looking for. At this point there is no keylogout command.

If somebody tries to use `su` to impersonate you, they will not be able to decrypt your secret key. Editing /etc/passwd will not help them either, because what they need to edit, your encrypted secret key, is stored in the NIS. If you log into somebody else's workstation and type in your password, then your secret key would be stored in their workstation and they could use `su` to impersonate you. To avoid this and other problems, do not give your password to a machine you do not trust. Someone on that machine could just as easily change login to save all the passwords it sees into a file.

Managing NFS and NIS daemons

25

Managing NFS and NIS daemons

Like many system-level programs, both NFS and NIS rely heavily on daemons. Under most circumstances, you do not notice the operation of these daemons. However, if they die or are killed accidentally, you need to know how to restart them. The listing below contains instructions for restarting these daemons. Always restart daemons in the order in which they appear in the /etc/rc.local file.

Frequently, certain daemons must be killed to start others. Pass over the instructions for killing NIS daemons if you are not running NIS—namely the instructions for `/usr/etc/ypbind` and `/usr/etc/ypserv`. These daemons are covered in more detail in the chapter on NIS.)

Step 1 `usr/etc/portmap`—Always start this daemon first in any sequence. There is no reason to ever kill it. If this daemon should die, kill then restart `/usr/etc/ypserv`, `/usr/etc/ypbind`, `/usr/etc/nfsd`, and `/usr/etc/inetd` according to the instructions given in the steps that follow.

Step 2 `/usr/etc/biod`—Start this daemon after /etc/portmap but before you mount file systems with NFS. Typically, you need to start four `biod` daemons. To kill and restart them, use the following command sequence:

```
# ps ax | grep biod
# kill -9  pid1, pid2, ...
# /usr/etc/biod 4
# /usr/etc/exportfs -a
```

where *pid1, pid2, ...* are the numbers found by the instruction above.

Step 3 `/usr/etc/ypserv`—Start this daemon only if your machine is an NIS server: slave or master machine. Start it after the domain name is set and after you start `usr/etc/portmap`. If it dies, restart it by entering

```
# ps ax | grep ypserv
# kill -9  pid1, pid2, ...
# /usr/etc/ypserv
```

where *pid1, pid2, ...* are the numbers found by the previous instruction.

Step 4 `usr/etc/ypbind`—Start this daemon only if your machine is running NIS, (i.e., if your machine is an NIS server or client). When this daemon is running, system programs use NIS services rather than reading the local files. If this daemon dies, restart it by entering

```
# ps ax | grep ypbind
# kill -9  pid1, pid2, ...
# /usr/etc/ypbind
```

where *pid1, pid2, ...* are the numbers found by the previous instruction.

Step 5 `usr/etc/nfsd`—Start this daemon after the other daemons in /etc/rc.local. Typically, you will start four nfsd daemons. To kill and restart them, use the following command sequence:

```
# ps ax | grep nfsd
# kill -9 pid1, pid2, ...
# /usr/etc/nfsd 4
```

where *pid1, pid2, ...* are the numbers found by the previous instruction.

Step 6 `usr/etc/inetd`—Start this daemon from the /etc/rc file. It should be sent a `SIGHUP` (`kill -1`) signal; this signal causes `inetd` to read its configuration file, /etc/inetd.conf. `inetd` reconfigures itself each time you kill it, sets up new sockets, and restarts itself.

To kill `inetd`, use the following command sequence:

```
# ps ax | grep inetd
# kill -1 process_number
```

where *process_number* is the number found by the previous instruction.

Setting up rexd

`rex` and its accompanying daemon, `rexd`, enable you to execute commands remotely, providing you with the opportunity to off-load `nroff`, `make`, and other jobs onto lightly loaded machines. `rex` and `rexd`, by default, are on standard NFS release distribution tapes and use a simple syntactical structure.

`rex` operates quickly, without discernible start-up time, similar to `rlogin` and `rsh`, but quicker. Unlike `rsh` and `rlogin`, `rex` neither starts a shell on the remote host, nor does it perform remote logins. Hence, users are limited to relatively simple command sequences because it cannot interpret the complex command sequences interpreted by `rsh`. Instead, `rex` accomplishes the remote execution task by mounting local file partitions on remote machines via NFS. In an open environment, `rex` can also be used to improve load balancing between machines.

Using the rexd daemon

To administer the remote-execution system, you must understand not only rex, but also rexd, its accompanying daemon. The following sections describe how to use and administer rexd and discuss how to avoid potential security problems.

rexd is the server-side daemon for remote program execution. It is started by inetd which is why its start-up command is an entry in the inetd configuration file, /etc/inetd.conf. If you encounter problems with rex, check inetd.conf for its start-up command line as shown in Figure 156.

Figure 156 Line in /etc/inetd.conf

```
rexd      stream   tcp   wait root 1      /usr/etc/rexd          rexd
```

If you do not find this line, add it, then restart inetd according to the instructions included in this chapter.

rexd is started when a user invokes rex—it does not run continuously. When pending rex requests have been completed, rexd times itself out, and inetd takes control of the socket rexd was using.

There is no obvious way for rexd to break. If rexd does not seem to be working, however, kill it using the following command sequence:

```
# ps ax | grep inetd
# kill -1 pid
```

Once inetd receives the signal, it reconfigures inetd.conf and restarts itself. The reconfiguration should eliminate problems with rexd.

Addressing security issues

Unlike other networking utilities, rexd enables users to mount directories. In theory, this flexible approach can lead to security problems. If, for example, a user with a home directory in /mnt invokes the following command sequence:

```
% rex convex4 sleep 10000
```

/mnt is mounted on a temporary partition on the remote host, convex4. The mount point stays active for 10000 seconds, giving users on convex4 access to /mnt.

You can avoid this type of problem, of course, by protecting sensitive partitions in /etc/exports. If, in the previous example, /mnt had not been included in the /etc/exports file, users would be unable to mount the partition. Instead, they would receive the error message:

```
cannot mount file system
```

Note that permission checking under `rex` is as strong as the permission checking used with `rsh` and `rlogin`. In all cases, the system checks not only for legitimate passwords, but also checks against the listings in /etc/host.equiv. All things considered, `rex` is a reasonable security risk in open environments, but if you operate a secure environment, you may want to reconsider distributing `rex` to your users. If you decide not to provide `rex`, you can turn it off by removing its entry from /etc/inetd.conf, then reconfiguring `inetd` by sending a `SIGHUP` (`kill -1`) signal.

Troubleshooting NFS and NIS 26

Before you start investigating any NFS-related problem, you should be familiar with the names and functions of the various daemons and database files. Some suggested readings are:

- man pages on:
  - mount(8), nfsd(8), biod(8), showmount(8), rpcinfo(8), mountd(8c), inetd(8c), fstab(5), mtab(5), and exports(5).

When tracking an NFS problem, keep in mind that, like all network services, the three main points of failure are

- Server
- Client
- Network

Your approach— isolate each individual component to find the one that isn't working. The following sections provides general pointers and describe common problems.

Tracking causes, where to begin looking

If a client is having NFS trouble, make sure the server is up and running. From a client, to see if the server is up, enter

```
% /usr/etc/ping server_name
```

If the server responds, enter

```
% /usr/etc/rpcinfo -p server_name
```

The server should print a list of program, version, protocol, and port numbers that resembles the output shown inFigure 157.

Figure 157 Sample rpcinfo output

```
program vers proto   port
100000    2   tcp    111  portmapper
100000    2   udp    111  portmapper
100004    2   udp   1026  ypserv
100004    2   tcp   1027  ypserv
100004    1   udp   1026  ypserv
100004    1   tcp   1027  ypserv
100007    2   tcp   1028  ypbind
100007    2   udp   1034  ypbind
100007    1   tcp   1028  ypbind
100007    1   udp   1034  ypbind
100009    1   udp   1023  yppasswdd
100003    2   udp   2049  nfs
100002    1   udp   1120  rusersd
100002    2   udp   1120  rusersd
100001    1   udp   1126  rstatd
100001    2   udp   1126  rstatd
100001    3   udp   1126  rstatd
100008    1   udp   1131  walld
100005    1   udp   1134  mountd
100011    1   udp   1137  rquotad
100012    1   udp   1140  sprayd
100017    1   tcp   1055  rexd
100026    1   udp   1145  bootparam
```

- Use `rpcinfo` to determine whether the `mountd` server is running by entering

  `% /usr/etc/rpcinfo -u server_name 100005 1`

  where **`100005 1`** comes from the `mountd` entry in the `rpcinfo` output above. The response should return as: `proc 100005 version 1 ready and waiting`

- If attempts to get the server to respond fail, log into the server's console and see if it is running.
  - If the server is alive but your machine cannot reach it, check the Ethernet connections between your machine and the server.

– If the server is fine and the network is fine, enter `ps` to check your client daemons. `portmap` and several `biod` daemons should be running. If NIS is running, `ypbind` must also be running.

Entering

```
% ps ax
```

at a client running NIS, should result in output similar to this:

```
32 ? I    1:07 /etc/portmap
38 ? I    0:42 /usr/etc/ypbind
61 ? S    0:45 (biod)
62 ? S    0:36 (biod)
63 ? S    0:30 (biod)
64 ? S    0:27 (biod)
```

The four subsections ahead discuss the most common types of failures. The first tells what to do if your remote mount fails; the next three describe servers not responding after you have file systems mounted.

Note

Never mount or unmount a file system while one of its directories is exported. The proper sequence is to unexport a directory, unmount the filesystem, re-mount the same or a different filesystem, then reexport the directory. If you do not follow this procedure, it is possible to have a directory that `exportfs` claims is exported, but that clients cannot mount. Reexporting the directory corrects this inconsistency.

Handling failed remote mount operations

If `mount` fails for any reason, read through this section for specifics about what to do. They are arranged according to where they occur in the mounting sequence and are labeled with the error message you are likely to see.

`mount` can get its parameters from either the command line or from the fstab file (refer to the mount(8) man page). The example below assumes command line arguments, but the same debugging techniques work if fstab is used in a `mount -a` command. The following mount command, entered on an NFS client,

```
# mount krypton:/usr/src /krypton.src
```

asks the server machine krypton to return a file handle for the directory /usr/src. This file handle is then passed to the kernel in the mount system call. The kernel looks up the directory /krypton.src and, if conditions are right, it ties the file handle to the directory in a mount record. From now on, all file system requests to that directory and below go through the file handle to the server krypton.

Mounting—failure types and their messages

Any one of the steps listed in the previous section can fail, some of them in more than one way. Possible cause and what you can attempt in order to correct a failure condition accompany the text message are listed below.

- `mount: ... already mounted`

  The file system that you are trying to mount is already mounted or it has a bogus entry in /etc/mtab.

- `mount: ... Block device required`

  You probably omitted `krypton:` from the command line
  `# mount krypton:/usr/src /krypton.src`

  The mount" command assumes you are doing a local mount unless it sees a colon in the file system name or the file system type is NFS in /etc/fstab.

- `mount: ... not found in /etc/fstab`

  If you see this message, it means the argument you gave mount was not in any of the entries in /etc/fstab. If mount is called with a directory or file-system name but not both, it looks in /etc/fstab for an entry whose file system or directory field matched the argument. For example:

  ```
  # mount /krypton.src
  ```

  searches /etc/fstab for a line that has a directory name field of /krypton.src. If it finds an entry, such as

  ```
  krypton:/usr/src /krypton.src nfs rw,hard 0 0
  ```

  it does the mount as if you had typed

  ```
  # mount krypton:/usr/src /krypton.src
  ```

- `/etc/fstab: No such file or directory`

  mount tried to look up the name in /etc/fstab but there was no /etc/fstab.

- `... not in hosts database`

  This means NIS could not find the hostname you gave it in the /etc/hosts database or that ypbind is dead on your

machine. (Note that if you are not running NIS, no hostname entry is in the local /etc/hosts file.) First check the spelling and the placement of the colon in your mount call. If those look right, make sure that ypbind is running by entering

```
# ps ax | grep ypbind
```

Try rlogin or rpc to some other machine. If this also fails, your ypbind is probably dead or hung. If you get this message only for one host name, it means that the /etc/hosts entry on NIS server needs to be checked. See the section on debugging NIS later in this chapter.

- `mount: directory path must begin with '/'`

  The second argument to mount is the path of the directory to be covered. This must be an absolute path starting at "/".

- `mount: ... server not responding: RPC: Port mapper failure - RPC: Timed out`

  Either the server you are trying to mount from is down, or its portmapper is dead or hung. Try logging in to that machine. If you can log in, enter

  ```
  # rpcinfo -p
  ```

  to get a list of the registered programs.

  If you do not get a listing, restart portmapper on the server according to the instructions in the "Configuring portmap" on page 158. Restarting portmapper requires that you kill and restart both inetd and ypbind.

  If you cannot rlogin to the server but the server is up, check your Ethernet connection by trying to rlogin to some other machine. Also check the server's Ethernet connection.

- `mount: ... server not responding: RPC: Program not registered`

  This means that mount got through to the portmapper but the NFS mount daemon (rpc.mountd) was not registered. Go to the server and be sure that /usr/etc/rpc.mountd exists and that there is an entry in /etc/inetd.conf exactly like the one shown in Figure 158.

Figure 158 Line in /etc/inetd.conf

```
mount      dgram    udp     wait root 1        /usr/etc/mount              mountd
```

Finally, use ps to be sure that inetd is running. If you had to change /etc/inetd.conf, you must kill inetd and restart it.

- `mount: ...: No such file or directory`

  Either the remote directory or the local directory does not exist. Check spelling. Try to `ls` both directories.

- `mount: not in export list for ...`

  Your machine name is not in the export list for the file system you want to mount from the server. You can get a list of the server's exported file systems by entering

  `# showmount -e hostname`

  If the file system you want is not in the list, or your machine name or netgroup name is not in the user list for the file system, log in to the server and check the /etc/exports file for the correct file system entry. A file system name that appears in the /etc/exports file, but not in the output from `showmount`, indicates a failure in `mountd`. Either it could not parse that line in the file, or it could not find the file system, or the file system name was not a locally-mounted file system. See the exports(5) man page for more information. If exports seems okay, check the server's `ypbind` daemon: it may be dead or hung.

- `mount: ...: Permission denied`

  This message is a generic indication that some authentication failed on the server. It could simply be that you are not in the export list (see above), or the server couldn't figure out who you are (`ypbind` dead). Or, it could be that the server does not believe that you are who you say you are. Check the server's /etc/exports and `ypbind`. Here, you can change your hostname with hostname and retry the mount.

- `mount: ...: Not a directory`

  Either the remote path or the local path is not a directory. Check spelling and try to `ls` both directories.

- `mount: ...: Not owner`

  You have to do the mount as root on your machine because it affects the file system for the whole machine, not just you.

- `mount: ...: Cannot mount a directory on top of itself`

  Self-explanatory.

When the system hangs at startup

If your machine comes part way up after a boot, but hangs where it would normally be doing remote mounts, probably one or more servers is down or your network connection is bad. Refer to the two previous subsections.

To avoid a hung system, add the `bg` flag to the NFS partitions listed in /etc/fstab. For example, an entry would look like this:

```
krypton:/usr/man /usr/man nfs rw,soft,bg 0 0
```

When remote file access seems slow

If access to remote files seems unusually slow, enter

```
# ps aux
```

on the server to be sure it is not being clobbered by a runaway daemon, bad tty line, etc. If the server seems okay and other people are getting good response, make sure your block I/O (`biod`) daemons are running; enter

```
ps ax
```

and look for `biod`. If they are not running or are hung, you can find the process ID's by entering

```
ps ax | grep biod
```

and kill them with

```
# kill -9 pid1 pid2 pid3 pid4
```

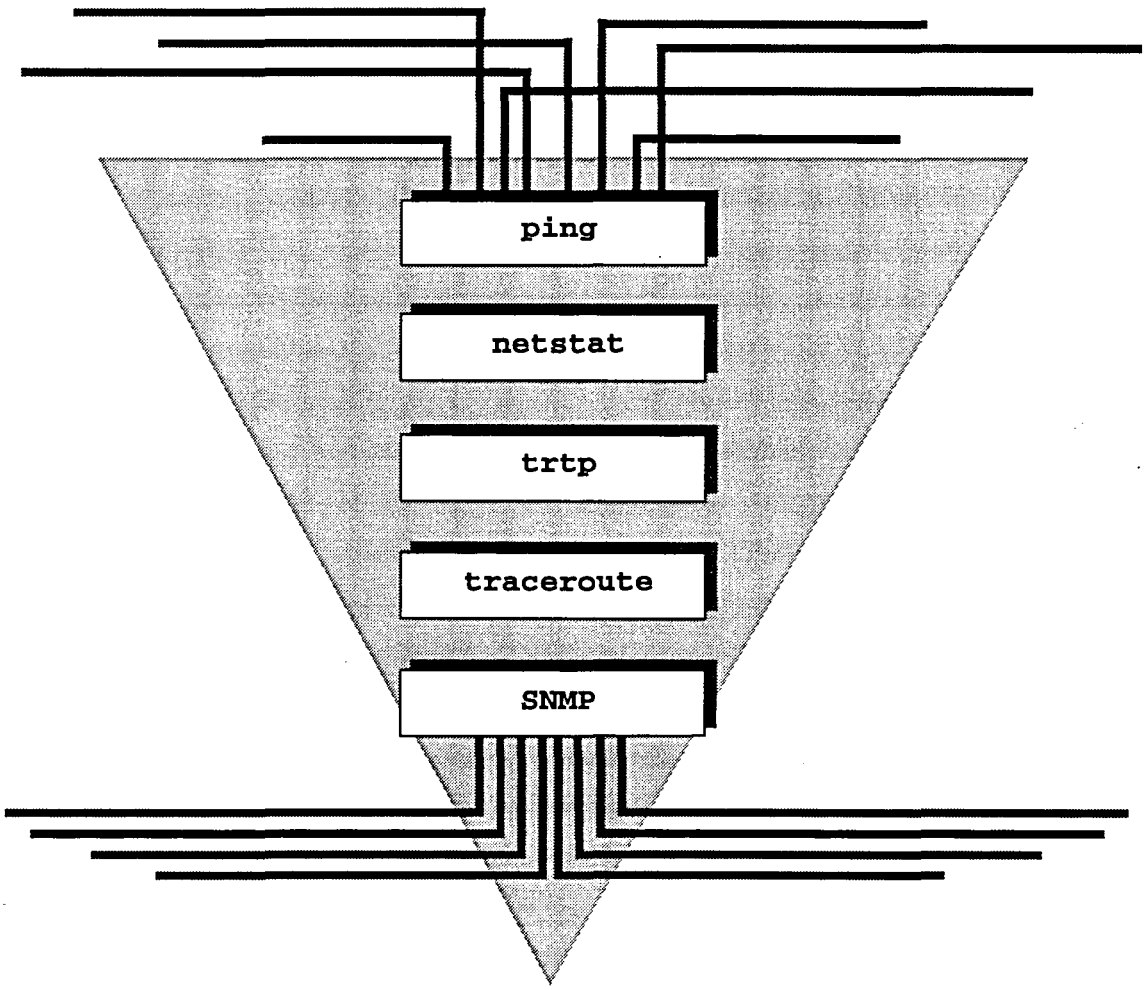
Restart them with

```
# /usr/etc/biod 4
```

To determine whether they are hung, enter `ps` as above, copy a large remote file, then enter `ps` again. If the `biod`s do not accumulate CPU time, they are probably hung.

If `biod` is okay, check your Ethernet connection. The command `netstat -i` tells you if you are dropping packets. Also, you can use `nfsstat -c` and `nfsstat -s` to lookup retransmission rates for the client or server. A retransmission rate of 5% is considered high. Excessive retransmission usually means a bad Ethernet board, a bad Ethernet tap, a mismatch between board and tap, or a mismatch between your Ethernet board and the server's board.

Part 5 Testing and troubleshooting the network



Introduction

27

Once you have properly configured the network and its supporting software, you should experience few serious problems. Often, you find that the problems reported are largely the result of user error. A user trying to connect to a nonexistent host, for example, may reach the conclusion that `ftp` is broken when the connection times out. Or, the user may decide the network is down when trying to connect to a host that is not running. Most other problems—runaway jobs, missing daemons, lack of kernel memory, bad connections, unplugged transceivers—can be located by using the procedures discussed in this part.

Begin troubleshooting by following the general procedure outlined below.

Step 1 Use `ping` to test the network. (Refer to Chapter 28, "Using ping," page 333.)

Step 2 Check running processes with the `ps` command. You may notice a network process at the top of the list. If so, that process is probably a runaway flooding the network with "renegade" packets. During a crisis of this sort, users may experience problems ranging from sluggish response to a lack of memory buffers for network programs. As a check, you may want to run `syspic` to see if the CPU is being consumed by the runaway process. You may also want to run `strstat` to see if the networking buffers are all in use. Of course, when a runaway process is identified, the solution is obvious and simple: kill the process, then use `netstat` to make sure that the network is again running normally.

Step 3 As a second step, run `netstat` and `strstat`. Because the options available enable you to look at all aspects of the network from memory use to I/O throughput, `netstat` and `strstat` are your best ways to gain information about how the network is running. (Instructions for using `netstat` are included in Chapter 29, "Using netstat," page 337.)

Step 4 If `netstat` provides clues but no solutions, run `ping` to isolate problems. If you cannot successfully reach another host by using `ping`, you can be confident that there is a problem in the transmission media or a software problem on either the remote host or the local host.

Step 5 If you suspect software problems on either the local host or the remote host,

- Verify contents of the /etc/hosts, /etc/networks, and /etc/rc.local files.
- Check /etc/rc.local to make sure that all the daemons needed are present.
- Make sure `inetd` is running and verify the contents of the /etc/inetd.conf file.
- If you have installed an UltraNet network, ensure that `unetd` is running and verify the contents of the /etc/unetd.conf file.

Step 6 If you are able to isolate problems to a particular daemon or protocol, `trpt` (explained in Chapter 30, "Using trpt," page 347) may be useful. `trpt` is used primarily for socket-level debugging, the process of tracing packets as they move through various layers of the network. Although `trpt` is normally used for debugging individual network programs, it may prove useful if you are experienced with socket-level debugging.

If you have worked through this procedure and you still cannot pinpoint the problem, call the CONVEX Technical Assistance Center (TAC).

Using `ping`

28

The simplest way to test the network is to use `ping`. `ping` tests connections with other hosts by sending them a particular type of datagram, called an ICMP (Internet Control Message Protocol) ECHO. These datagrams work well for testing connections because they are simply bounced back by the other host once they are received.

To test a connection to another host, start the transmission stream by invoking `ping`, and then check to make sure that packets are being returned. If at least 95 percent of the packets are being returned, you know that the network is functioning properly.

Use `ping` as follows:

`/usr/etc/ping` *remote_host_name*

where *remote_host_name* is the name of the remote machine with which you want to communicate. For example, to check whether the connection between `veeger` and `jupiter` is viable, run `ping` from `veeger` as follows:

```
/usr/etc/ping jupiter
```

Sample output is shown in Figure 159.

Figure 159 Sample `ping` output

```
PING jupiter: 56 data bytes
64 bytes from 140.50.0.1: icmp_seq=0. time=40. ms
64 bytes from 140.50.0.1: icmp_seq=1. time=40. ms
64 bytes from 140.50.0.1: icmp_seq=2. time=20. ms
64 bytes from 140.50.0.1: icmp_seq=3. time=20. ms
64 bytes from 140.50.0.1: icmp_seq=4. time=20. ms
64 bytes from 140.50.0.1: icmp_seq=5. time=30. ms
64 bytes from 140.50.0.1: icmp_seq=6. time=30. ms
64 bytes from 140.50.0.1: icmp_seq=7. time=30. ms
----veeger-il PING Statistics----
8 packets transmitted, 8 packets received, 0% packet loss
round-trip (ms) min/avg/max = 20/29/40
```

In this example, eight packets were received from network address 140.50.0.1 with transmission times ranging from 20 to 40 milliseconds. No packets were lost; therefore, you can assume that the link between the hosts is viable. If no packets had been returned, or if there had been a high rate of packet loss, you would have been correct in assuming a problem in the line between hosts. Return-trip-time statistics included in the output allow you to determine whether transmission times are prohibitive.

You can specify the size and number of packets sent over the network. If you specify the number of packets, you must also specify the size. For example,

```
ping jupiter 24 3
```

sends 3 packets with a data byte size of 24 to the remote host `jupiter`. Sample output is shown in Figure 160.

Figure 160 Sample `ping` output with different packet size and number

```
PING jupiter.convex.com [130.168.71.5]: 24 data bytes
32 bytes from 140.50.0.1: icmp_seq=0. time=13. ms
32 bytes from 140.50.0.1: icmp_seq=1. time=18. ms
32 bytes from 140.50.0.1: icmp_seq=2. time=8. ms

----jupiter.convex.com PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip (ms) min/avg/max = 8/13/18
```

The -v option displays verbose output (ICMP packets other than ECHO are listed). For example,

```
ping -v jupiter 24 3
```

displays output similar to that shown in Figure 161.

Figure 161 Sample `ping -v` output

```
PING jupiter.convex.com [140.50.0.1]: 24 data bytes
32 bytes from 140.50.0.1: icmp_seq=0. time=7. ms
36 bytes from 140.51.1.2: icmp_type=3 (Dest Unreachable)
x 0: x45000024
x 4: xa4630000
x 8: xfe010000
x c: x82a84601
x10: x82a84701
x14: x 3032d16
x18: x 0
x1c: x4500004c
x20: xda070000
x24: x1d110000
x28: x82a84701
x2c: x82a84601
icmp_code=3
32 bytes from 140.50.0.1: icmp_seq=1. time=7. ms
32 bytes from 140.50.0.1: icmp_seq=2. time=8. ms

----jupiter.convex.com PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
```

You can send just one ICMP ECHO packet to a remote host with the `pong` command. For example,

```
pong jupiter
```

displays the message

```
jupiter.convex.com is alive
```

if the remote host replies.

Using `netstat`

29

`netstat` displays statistical "snapshots" of various aspects of network operation. These snapshots enable you to monitor network use and efficiency during periods of normal operation, and to quickly track down problems when the network is malfunctioning. A few of the most useful `netstat` options are discussed here. For information about other `netstat` options, refer to the netstat(1C) man pages.

When invoked with no options, `netstat` displays the status of current network transmissions. Network addresses are shown as host names. To use `netstat`, enter

```
netstat
```

Output is similar to Figure 162.

Figure 162 Sample `netstat` output

```
Active internet connections
Proto      Local Address    Foreign Address        (state)
tcp        lynx.1498        fang.jpx               ESTABLISHED
tcp        lynx.1564        tiger.jpx              ESTABLISHED
udp        lynx.domain      *.*
udp        lynx-f.domain    *.*
udp        lynx.domain      *.*
Active UNIX domain sockets
Type       Socket ID        Bind Max   Flags       Addr
dgram      0x0000           0          0000
dgram      0x0001           0          0000        /dev/log
stream     0x0015           5          0000        /etc/cronsock
stream     0x0016           0          0060
dgram      0x0017           0          0000
stream     0x0018           5          0000        /dev/printer
```

Displaying status of autoconfigurec interfaces

`netstat -i` is useful for quick checks on network status and for checking the fate of outgoing `ping` packets. Status information is displayed for all network interfaces autoconfigured at boot time. To use this option, enter

```
netstat -i
```

Typical output is shown in Figure 163.

Figure 163 Sample `netstat -i` output

```
Name  Mtu   Network    Address   Ipkts     Ierrs  Opkts    Oerrs  Collis
eth0  1500  syssw-net  lynx      14637073    5    13549146  0      0
fddi0 4352  fddi2-net  lynx-f    255918      0    151417    0      0
```

The output displayed in Figure 163 shows interfaces that are up (`syssw-net` and `fddi2-net`), the name and addresses of the interfaces, and the maximum size of packets that can be transmitted across each network (listed under the column headed by "Mtu").

`netstat -i` displays statistics on the number of incoming packets (*Ipkts*), incoming packet errors (*Ierrs*), outgoing packets (*Opkts*), outgoing packet errors (*Oerrs*), and "collisions" on the network (*Collis*). (Collisions occur when two computers try to transmit packets across the network at exactly the same instant.)

You can show the number of dropped packets for network interfaces when you use the `-i` option with the `-d` option. For example,

```
netstat -id
```

displays output similar to that shown in Figure 164.

Figure 164 Sample `netstat -id` output

```
Name  Mtu   Network    Address  Ipkts     Ierrs  Opkts    Oerrs  Collis Dropped
eth0  1500  syssw-net  lynx     16224365    5    15003217  0      0      138
fddi0 4352  fddi2-net  lynx-f   284618      0    176882    0      0      0
```

Information about incoming and outgoing packets enables you to judge the status of the network at a glance. Furthermore, if you suspect trouble with a particular remote host, you can use

`netstat -i` to watch the network in real time. Use the following procedure.

Step 1 Run `ping` to start a packet stream directed toward the host in question. For example:

```
/usr/etc/ping obewan > & /dev/null &
```

Step 2 Start `netstat -i` with an appended "update" argument. For example:

```
netstat -i 3
```

The appended argument causes `netstat` to update the statistics and redisplay them every *n* seconds. (In this example, the display is updated every three seconds.) Statistics are displayed only for the most active interface.

Step 3 Check to see if packets are being sent and received by the local host.

Displaying addresses numerically

`netstat -n` displays the status of current network transmissions. Internet addresses are displayed instead of host names. The `-n` option may be used with any display format. Enter:

```
netstat -n
```

Typical output is displayed in Figure 165.

Figure 165 Sample `netstat -n` output

```
Active Internet connections
Proto      Local Address    Foreign Address       (state)
tcp        100.0.28.2.1324  100.0.1.13.25         TIME_WAIT
tcp        100.0.28.2.1021  100.0.1.13.513        ESTABLISHED
tcp        100.0.28.2.513   100.0.1.13.1022       FIN_WAIT_2
tcp        100.0.28.2.513   100.0.1.13.1023       FIN_WAIT_2
tcp        100.0.28.2.1022  100.0.1.13.513        ESTABLISHED
tcp        100.0.28.2.1014  100.0.1.13.1022       ESTABLISHED
```

The output shown in Figure 165 enables you to determine the status of the current network connections. The `Proto` field shows you which protocols are being used, and the `state` codes displayed show the state of the protocol when `netstat` was run.

Displaying routing information

This section describes how to display routing information with `netstat`.

Displaying routing tables

Use `netstat -r` to display available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. Enter

```
netstat -r
```

Typical output is shown in Figure 166.

Figure 166 Sample `netstat -r` output

```
Routing tables
Destination  Gateway    Flags   Refs      Use       Interface
localhost    localhost  UH      0         0         lo0
macd2-net    lion-73    UG      0         0         eth0
twinkie-net  lion-73    UG      0         1287      eth0
default      lion-73    UG      0         15924     eth0
cebit-net    lion-73    UG      0         0         eth0
syssw-net    lynx       U       48        234181    eth0
fddi2-net    lynx-f     U       0         7578      fddi0
termi-net    lion-73    UG      1         15360     eth0
```

Direct routes are created for each interface attached to the local host; the `Gateway` field for such entries shows the address of the outgoing interface.

The flags field shows the state of the route:

- `U`—route is up
- `G`—route is to a gateway
- `H`—route is for a particular host
- `D`—route was created dynamically by a redirect
- `M`—route was modified by a redirect

The `Refs` field gives the current number of active uses of the route.

Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The `Use` field provides a count of the number of packets sent using that route.

The `Interface` entry indicates the network interface used for the route.

Displaying routing statistics

When you use the `-r` option with the `-s` option, `netstat` displays routing statistics. For example,

```
netstat -rs
```

displays information similar to that shown in Figure 167

Figure 167 Sample `netstat -rs` output

```
routing:
        0 bad routing redirects
        0 dynamically created routes
        0 new gateways due to redirects
        24 destinations found unreachable
        22206 uses of a wildcard route
```

Displaying protocol statistics

`netstat -p` displays information about a particular protocol. Enter

`netstat -p` *protocol*

where *protocol* is either the protocol's well-known name or an alias specified in /etc/protocols. The two most common protocols used by CONVEX Internet Services are `tcp` and `udp`.

`netstat -s` displays statistics about all protocols in use. Typical output is displayed in Figure 168.

Figure 168 Sample `netstat -s` output

```
ip:
      1369846 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
       142517 fragments received
       0 fragments dropped (dup or out of space)
       465 fragments dropped after timeout
       1 packet forwarded
       0 packets not forwardable
       0 redirects sent
icmp:
       372 calls to icmp_error
       0 errors not generated because old message was icmp
       Output histogram:
               echo reply: 721
               destination unreachable: 372
       0 messages with bad code fields
       0 messages < minimum length
       0 bad checksums
       0 messages with bad length
       Input histogram:
               echo reply: 35
               destination unreachable: 2831
               echo: 721
       721 message responses generated
tcp:
       409545 packets sent
               322930 data packets (62559395 bytes)
               3170 duplicate acks
               0 acks for unsent data
               283718 packets (50257483 bytes) received in-sequence
```

Figure 168 Sample `netstat -s` output **(continued)**

```
                671 completely duplicate packets (46601 bytes)
                224 packets with some dup. data (2610 bytes duped)
                2236 out-of-order packets (1052663 bytes)
                4 packets (0 bytes) of data after window
                0 window probes
                1784 window update packets
                0 packets received after close
                1 discarded for bad checksum
                0 discarded for bad header offset fields
                0 discarded because packet too short
                1060 connection requests
        1438 connection accepts
        2464 connections established (including accepts)
        318 connections closed (including 10 drops)
        36 embryonic connections dropped
        282010 segments updated rtt (of 283618 attempts)
        1923 retransmit timeouts
                12 connections dropped by rexmit timeout
        3 persist timeouts
        150 keepalive timeouts
                50 keepalive probes sent
                28 connections dropped by keepalive
        0 segments dropped by PAWS
        144492 acks found by header prediction
        168268 data segments found by header prediction
udp:
        0 incomplete headers
        0 bad data length fields
        0 bad checksums
        0 receive buffer overflows
```

As you can see, a typical LAN uses many different protocols. Transmission Control Protocol (TCP) is used by the standard utilities for packet transmission. User Datagram Protocol (UDP) is used by `rwhod` for the transmission of user datagrams. Internet Control Message Protocol (ICMP) is the protocol used by `ping`. Internet Protocol (IP) is the underlying protocol layer associated with TCP, UDP, ICMP, and other protocols.

As you look over the output from `netstat -s`, check the checksums to see how many transmissions have had errors. (Checksums are validity checks on the packets transmitted.) If the percentage of bad checksums is high, you more than likely have problems with the LAN cabling, transceiver, or connections. Unless the percentage is high, though, don't worry—networks are designed to handle errors.

The sysgen parameter, udpcksum, enables checksumming of User Datagram Protocol (UDP) datagrams. UDP checksumming results in additional overhead, because each transmitted and received UDP datagram is checksummed when the parameter is turned on.

If you are running Serial Line Internet Protocol (SLIP), you must enable checksumming.

Using `trpt`

30

Socket-level debugging

`trpt` is a protocol-tracing daemon typically used to debug network programs. Although several options are available, the *-p* option, which prints the addresses of protocol control blocks, is the most useful for debugging the network itself. `trpt` should be used under the following conditions:

- When you have tracked problems to a particular daemon, and want a closer look at how the daemon is working.
- When you want to check various network daemons as you bring up the network.
- You use a similar procedure for both cases. Assume that you have noticed problems with `ftp`. First, make sure that no other users are using `ftp`, then shut down the `ftp` daemon by using this procedure.

Step 1 Log in as `superuser`.

Step 2 Modify the /etc/inetd.conf file to replace the line similar to

```
ftp stream tcp nowait root /usr/etc/in.ftpd  ftpd
```

with

```
ftp stream tcp nowait root  /usr/etc/in.ftpd  ftpd -d
find inetd; kill -Hup inetd
```

Note

Remember to change the entry in /etc/inetd.conf back to the original line when you finish debugging.

Step 3 Begin an `ftp` to `localhost`, log in, and start a file transfer; then immediately stop the transfer using **CTRL-Z** as follows:

```
% ftp localhost

Connected to localhost.

220 convex FTP server (Version 4.98 Mon Mar 23
21:27:10 CST 1987) ready.
Name (localhost:): watson
Password (localhost:watson): comehere
331 Password required for watson.
230 User watson logged in.
ftp> get program.c
CTRL-Z (Press before the transfer is complete)
```

Step 4 Issue the command:

```
netstat -A
```

Output similar to that shown in Figure 169 is produced.

Figure 169 Sample `netstat -A` output

```
Active connections
PCB      Proto  Recv-Q  Send-Q  Local Address  Foreign Address(state)
ff58a0c  tcp    0       0       acme-il.login  veeger-ex.1016 ESTABLISHED
ff5360c  tcp    0       0       convex.login   convex.1018    ESTABLISHED
ff4be0c  tcp    0       0       convex.1018    convex.ftpd    ESTABLISHED
ff60e0c  tcp    0       0       convex.login   convex.1019    ESTABLISHED
ff55a0c  tcp    0       0       convex.1019    convex.login   ESTABLISHED
ff5ae0c  tcp    0       0       acme-il.login  veeger-ex.1018 ESTABLISHED
ff5aa0c  tcp    0       0       acme-il.login  veeger-ex.1019 ESTABLISHED
ff52c0c  tcp    0       1       acme-il.login  veeger-ex.1020 ESTABLISHED
```

Step 5 Run `trpt -p` with the address of the protocol control block in question:

```
trpt -p ff4be0c
```

This produces a listing of the trace records associated with the protocol control block in question. Sample output is shown in Figure 170.

Figure 170 Sample `trpt -p` output

```
8061d98c:
405 CLOSED:user ATTACH -> CLOSED
405 LISTEN:input 877901@0(win=2048)<URG> -> SYN_RCVD
405 SYN_RCVD:output [877941..877945)@877902(win=2048)<URG> -> SYN_RCVD
406 SYN_RCVD:input 877902@877942(win=2048)<ACK> -> ESTABLISHED
406 ESTABLISHED:user ACCEPT -> ESTABLISHED
410 ESTABLISHED:user CONTROL -> ESTABLISHED
414 ESTABLISHED:output [877942..87798c)@877902(win=2048)<ACK> ->
ESTABLISHED
414 ESTABLISHED:user SEND -> ESTABLISHED
416 ESTABLISHED:input 877902@87798c(win=1974)<ACK> -> ESTABLISHED
418 ESTABLISHED:user CONTROL -> ESTABLISHED
912 ESTABLISHED:user PROTOSEND -> ESTABLISHED
912 ESTABLISHED:input 877902@87798c(win=2048)<ACK> -> ESTABLISHED
```

Although correct interpretation of these trace records depends on familiarity with protocol-level debugging, they may be useful to you or to someone on your staff. As you look at the output, notice that after an initial `LISTEN`, protocol connections were opened, or `ESTABLISHED`, without problem.

Using traceroute

31

traceroute

`traceroute` is a diagnostic aid for network management. This program attempts to trace the route an IP packet follows to an Internet host by sending UDP probe packets with a short "time to live" and listening for an ICMP response from each gateway along the path to the specified host. `traceroute` is useful for determining the actual routes taken in the network and can also be used to identify breaks in the normal route.

The only mandatory parameter is the destination host name or IP number. For example, to trace the route a packet takes to the remote host tulips.de, enter

```
traceroute tulips.de
```

Output similar to that shown in Figure 171 is displayed.

Figure 171 Sample `traceroute` output

```
traceroute to tulips.de (150.168.115.1), 30 hops max, 40 byte packets
 1 lily-73 (150.168.73.254) 12 ms 6 ms 6 ms
 2 iris-f (150.168.85.253) 6 ms 6 ms 7 ms
 3 130.168.251.5 (150.168.251.5) 647 ms 647 ms 647 ms
 4 tulips.de (150.168.115.1) 667 ms 658 ms 652 ms
```

The output lists each gateway along the route to the destination host (the number in the first field indicates the number of hops or gateways to the destination host). `traceroute` sends 3 packets to each gateway and prints the number of milliseconds (up to a default maximum of 3 seconds) it takes for each packet to reach the gateway.

Very large numbers or timeouts (shown by an asterisk) indicate host problems or network congestion. Inconsistant numbers indicate spurious network traffic. A large time difference between hops (for example, between hops 2 and 3 in Figure 171) identify a slow link in the route.

For information about `traceroute` options and more examples of using `traceroute`, refer to the traceroute(8) man page.

Using SNMP

32

SNMP

SNMP (Simple Network Management Protocol) is a standard for management of TCP/IP-based internets. SNMP is defined in the following RFCs:

- RFC1155—Defines the Structure of Management Information (SMI), the mechanisms used for describing and naming objects for the purpose of management
- RFC1212—Defines a more concise description mechanism consistent with the SMI
- RFC1213—Defines the second version of the Management Information Base (MIB-II) for use with network management protocols in TCP/IP- based internets
- RFC1157—Defines SNMP, the protocol used for network access to managed objects

SNMP consists of a collection of network management stations (NMS) and network elements. Convex provides the management agent, `snmpd`, but not the NMS. The NMS, which has a user interface to SNMP, is available through third-party vendors and works with the agent that CONVEX supplies. Upon receipt of a request from the NMS, `snmpd` authenticates the request, attempts the operation, and returns a response.

For more information on SNMP, refer to the RFCs listed above and to the snmpd(8) man page.

Using strstat

33

This chapter explains how to use the `strstat` utility to troubleshoot problems with STREAMS resources.

Monitoring STREAMS resources

The amount of system resources used by STREAMS can be monitored by using the `strstat` utility. STREAMS resources are allocated at boot-time and can be controlled by modifying the boot-time parameter associated with that resource.

Figure 172 shows an example of `strstat` output.

Figure 172 `strstat` output

RESOURCE	CONFIG	IN USE	FREE	TOTAL USED	HIGH WATER	FAILED
dblks(0)	1024	0	1024	32	1	0
dblks(4)	1024	1	1023	1176	49	0
dblks(16)	1024	1	1023	1697	4	0
dblks(64)	1024	40	984	11877	70	0
dblks(128)	1024	1	1023	2833	14	0
dblks(256)	1024	0	1024	2049	8	0
dblks(512)	1024	1	1023	25	2	0
dblks(1k)	1024	0	1024	40	2	0
dblks(2k)	1024	6	1018	2520	12	0
dblks(4k)	1024	0	1024	0	0	0
dblks(64k)	0	0	0	0	0	0
mblks	10240	52	10188	23795	115	0
queues	4096	288	3808	1456	296	0
streams	1024	72	952	364	74	0
event cells	256	0	256	0	0	0
timeout cells	256	4	252	41172	5	0

Each row represents a different resource. Resources that can be displayed are

`dblks`	STREAMS data blocks. A separate entry is displayed for each class. The size of the data blocks is in parentheses.
`mblks`	STREAMS message blocks.
`queues`	STREAMS queue structures.
`streams`	System limit on maximum number of Streams.
`event cells`	STREAMS structure used in in bufcall.
`timeout cells`	STREAMS structure used in timeout routines.

For each resource, a column entry displays a particular characteristic. Column headings have the following meanings:

`RESOURCE`	Resource name.
`CONFIG`	Number configured within the system.
`IN USE`	Number currently allocated.
`FREE`	Number currently available.
`TOTAL USED`	Total number allocated since system boot.
`HIGH WATER`	Maximum number allocated, at one time, since system boot.
`FAILED`	Failed allocation attempts.

The `strstat` utility has several command line options that enable you to select the resources for which you want to display statistics. If no options are selected, statistics are displayed for all resources. For a complete list of `strstat` options, refer to the strstat(1) man page.

The `-C` option causes `strstat` to repeatedly monitor STREAMS resources. When followed by an integer, the `-C` option causes `strstat` to repeatedly monitor resources for the interval specified.The default is ten seconds. For example, the command

```
strstat -C5 |dis
```

displays STREAMS statistics continuously, updating the statistics every 5 seconds. The `strstat` command is piped through `dis` to refresh the screen rather than redisplay the output.

Troubleshooting resource problems

The values assigned to STREAMS boot-time parameters depend on the following factors:

- Contents of the STREAMS protocol stack (as specified in the net.conf file)
- Number of socket-based connect points and expected use of Internet Services (volume and pattern of network traffic)
- Number of UNIX domain connections (for example, the printer daemon)

The default values for all STREAMS boot-time parameters are optimal for the type of usage expected with CONVEX networking and should not need to be modified. If you experience failures in establishing networking or UNIX domain connections, use the `strstat` utility to determine if the problem is not enough STREAMS resources.

If you see repeated allocation failures in a resource (indicated by the number shown in the `FAILED` column of the `strstat` output) you may need to raise the value of the STREAMS boot-time parameter associated with that resource. Be aware, however, that STREAMS resources are physical memory resources that are available only to STREAMS processes. Increasing these resources may affect the ability of other processes in the system to allocate memory.

Table 6 lists STREAMS boot-time parameters with descriptions and default, minimum, and maximum values.

To override the default value for any of these parameters, you must add an entry for the parameter specifying the new value in the /mnt/os/bootcmd.local file on the Service Processor Unit (SPU). The new value for the modified parameter takes effect the next time the machine is booted. Refer to *Managing ConvexOS: Configuration Guide* for the procedure to modify the bootcmd.local file on the SPU.

Table 6 STREAMS boot-time parameters

Parameter	Description	Minimum	Maximum	Default
`str_n_queue`	Maximum number of queues	0	8192	4096
`str_n_stream`	Maximum number of streams	0	2048	1024
`str_n_event`	Maximum number of event cells (for STREAMS bufcalls)	0	512	256
`str_n_tevent`	Maximum number of timeout cells (for STREAMS timeouts)	0	512	16
`str_dblk_4096`	Maximum number of 4096 byte datablocks	0	2048	1024
`str_dblk_2048`	Maximum number of 2048 byte datablocks	0	2048	1024
`str_dblk_1024`	Maximum number of 1024 byte datablocks	0	2048	1024
`str_dblk_512`	Maximum number of 512 byte datablocks	0	2048	1024
`str_dblk_256`	Maximum number of 256 byte datablocks	0	2048	1024
`str_dblk_128`	Maximum number of 128 byte datablocks	0	2048	1024
`str_dblk_64`	Maximum number of 64 byte datablocks	0	2048	1024
`str_dblk_16`	Maximum number of 16 byte datablocks	0	2048	1024
`str_dblk_4`	Maximum number of 4 byte datablocks	0	2048	1024
`str_dblk_0`	Maximum number of 0 byte datablocks	0	2048	1024
`str_n_sockets`	Maximum number of sockets	0	1700	850
`str_dblk_64k`	Maximum number of 64K datablocks (only used by HIPPI)	0	2048	0
`str_n_mblk`[1]	Maximum number of message blocks	0	25000	10240

Table 6 STREAMS boot-time parameters **(continued)**

Parameter	Description	Minimum	Maximum	Default
`str_lo_pct`[2]	percentage at which a BPRI_LO allocb will fail	0	100	60
`str_med_pct`[2]	percentage at which a BPRI_MED allocb will fail	0	100	80
`str_n_muxlink`[2]	Maximum number of links (lower multiplexor connections)	0	512	100
`str_n_push`[2]	Maximum number of modules (I_PUSH ioctls)	0	512	100
`str_msg_sz`[2]	Maximum STREAMS data message size (in bytes)	16384	65536	16384
`str_ctl_sz`[2]	Maximum STREAMS control message size (in bytes)	16384	65536	16384

1str_n_mblk must be at least the total of the values of the datablocks.
[2]These parameters should not be changed unless recommended by the TAC.

Table 7 lists the STREAMS resources displayed by `strstat` and the STREAMS boot-time parameter associated with that resource.

Table 7 STREAMS resources and corresponding boot-time parameters

STREAMS resource	STREAMS boot-time parameter
`dblks(4)`	`str_dblk_4`
`dblks(16)`	`str_dblk_16`
`dblks(64)`	`str_dblk_64`
`dblks(128)`	`str_dblk_128`
`dblks(256)`	`str_dblk_256`
`dblks(512)`	`str_dblk_512`
`dblks(1k)`	`str_dblk_1024`
`dblks(2k)`	`str_dblk_2048`
`dblks(4k)`	`str_dblk_4096`

Table 7 STREAMS resources and corresponding boot-time parameters **(continued)**

STREAMS resource	STREAMS boot-time parameter
dblks(64k)	str_dblk_64k
mblks	str_n_mblk
queues	str_n_queue
streams	str_n_stream
event cells	str_n_event
timeout cells	str_n_tevent

The STREAMS boot-time parameters str_lo_pct, str_med_pct, str_n_muxlink, str_n_push, str_msg_sz, and str_ctl_sz do not have corresponding STREAMS resources displayed by strstat.

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

P

R

S

T

U

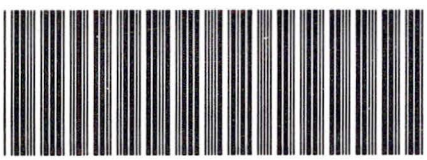
W

Y

Z

ORDER NUMBER
DSW-108

DOCUMENT NUMBER
710-023930-000



CONVEX PRESS